applied sciences

MDPI

Article

Optimization of Truck-Drone Parcel Delivery Using Metaheuristics

Sarab AlMuhaideb *, Taghreed Alhussan, Sara Alamri, Yara Altwaijry, Lujain Aljarbou and Haifa Alrayes

Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh 11362, Saudi Arabia; 438201646@student.ksu.edu.sa (T.A.); 438202311@student.ksu.edu.sa (S.A.); 438201218@student.ksu.edu.sa (Y.A.); 438200719@student.ksu.edu.sa (L.A.); 438200619@student.ksu.edu.sa (H.A.)
* Correspondence: salmuhaideb@ksu.edu.sa; Tel.: +966-11-805-2836

Abstract: This research addresses a variant of the traveling salesman problem in drone-based delivery systems known as the TSP-D. The TSP-D is a combinatorial optimization problem in which a truck and a drone collaborate to deliver parcels to customers, with the objective of minimizing the total delivery time. Determining the optimal solution is NP-hard; thus, the size of the problems that can be solved optimally is limited. Therefore, metaheuristics are used to solve the problem. Metaheuristics are adaptive and intelligent algorithms that have proved their success in many similar problems. In this study, a solution to the TSP-D problem using the greedy, randomized adaptive search procedure with two local search alternatives and a self-adaptive neighborhood selection scheme is presented. The proposed approach was tested on 200 instances with different properties from the publicly available "Instances of TSP with Drone" benchmark. Results were evaluated against state-of-the-art algorithms. Non-parametric statistical tests concluded that the proposed approach has comparable performance to the rival algorithms ($p = 0.074$) in terms of tour duration. The proposed approach has better or similar performance in instances where the drone and truck have the same speed ($\alpha = 1$).

Keywords: GRASP; TSP-D; optimization; routing; metaheuristics; drone

Citation: AlMuhaideb, S.; Alhussan, T.; Alamri, S.; Altwaijry, Y.; Aljarbou, L.; Alrayes, H. Optimization of Truck-Drone Parcel Delivery Using Metaheuristics. *Appl. Sci.* **2021**, *11*, 6443. https://doi.org/10.3390/app11146443

Academic Editors: Stylianos Pappas and Sokratis Katsikas

Received: 21 June 2021
Accepted: 9 July 2021
Published: 13 July 2021

1. Introduction

Over the last few years, drones have been adopted in many sectors [1,2], especially the commercial sector [3,4]. Drones are now deployed to support parcel delivery, an area that has traditionally been handled by trucks. Moreover, the COVID-19 pandemic has taken a significant toll on people all over the world [5]. Drones have helped people to comply with social distancing rules, as they provide contact-free delivery services. In addition, drones help with transportation logistics in many ways, such as avoiding traffic congestion, allowing for faster deliveries, and providing lower transportation costs [6]. However, there is an upper limit for the parcel weight, as drones cannot carry heavy parcels [7]. Additionally, drones are battery-powered, thereby limiting their delivery ranges. In contrast, truck delivery can work over a longer range, as it is fuel-based, and trucks can carry heavy and large parcels. Nonetheless, traditional truck delivery is slow and has high transportation costs [6]. Combining both truck and drone delivery could minimize the total operational costs and time, thereby enhancing the quality of service. This variant of the traveling salesman problem (TSP) [8,9] is known as the traveling salesman problem with drone (TSP-D) [6,10].

1.1. Problem Description

The TSP-D is defined on a graph $G = (V, A)$, where A is a set of arcs, each of which links two nodes in V. $V = \{0, \ldots, n+1\}$, where nodes 0 and $n+1$ represent the start point and the depot, duplicating the start and return points; the nodes $1, \ldots, n$ are customer locations. The set of customers is represented by $N = \{1, \ldots, n\}$. The set of customers that are served by a drone is denoted by $V_d \subseteq N$ (Figure 1) [11].

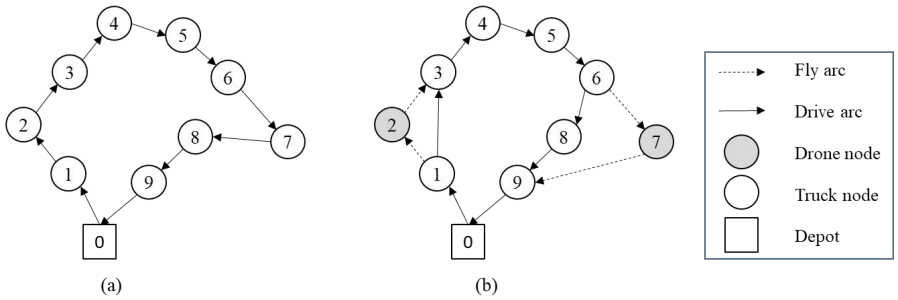


Figure 1. TSP vs. TSP-D for a graph of size $n = 9$: (**a**) displays an optimal TSP route without the aid of a drone; and (**b**) displays a TSP-D route, where the drone is launched from a truck, delivering parcels to two customers.

If a customer is served by a truck, it is called a truck delivery, whereas if a customer is served by a drone, it is called a drone delivery. The drone delivery is represented as a tuple $< i, j, k >$, in which i is the launch node (i.e., the node where the truck launches the drone, as in nodes 1 and 6 in Figure 1b), j is the drone node (i.e., the node that the drone delivers the parcel to, as in nodes 2 and 7 in Figure 1b), and k is the rendezvous node (i.e., where the truck and drone meet again, as in nodes 3 and 9 in Figure 1b); k can either be a customer location or the depot. A sortie refers to a sequence of nodes between a launch node and a rendezvous node. For example, the sequences $< 1, 2, 3 >$ and $< 6, 7, 8, 9 >$ represent two distinct sorties in Figure 1b. The drone has a constant level of endurance, defined as the maximum time that the drone can fly without needing to be charged. $< i, j, k >$ is a feasible solution for a drone delivery when $i \neq j, j \neq k, k \neq i$, $t_{ij} + t_{jk} \leq e$, where t_{ij} is the time taken by the drone to move from i to j, t_{jk} is the time taken by the drone to move from j to k, and e is the drone's endurance. The drone path follows the road on the network for safety. The objective of the TSP-D is to find the tour with the shortest delivery time such that all customer locations are served by either the truck or the drone. The pickup, delivery, and recharging times of the drone are neglected.

There are additional constraints that need to be considered:

- Both vehicles must start from and return to the depot;
- Customers can only be served once, either by the drone or the truck;
- The drone does not have a rendezvous node before the delivery drone node;
- The drone cannot serve more than one customer between nodes i and j.

A very similar problem to the TSP-D is the flying sidekick traveling salesman problem (FSTSP) [12]. The FSPSP is also a collaboration between a single truck and a single drone to deliver parcels to customers. The difference between the TSP-D and FSTSP is that in the FSTSP, two metrics are used for the distance—one for the truck (Manhattan distance) and one for the drone (Euclidean distance)—as the drone can fly directly from the launch point to the destination, thereby ignoring road restrictions. In the TSP-D, however, the drone must follow the road network.

The TSP-D is considered an NP-hard problem [13]. Exact methods have been used to solve the TSP-D [13–16]; however, due to computational limitations, these methods are only feasible on instances of limited sizes. Heuristics and metaheuristics are generally used to minimize the tour costs. Several taxonomies have been proposed for use in metaheuristics [17,18], among which solution and population-based approaches are the most common classifications.

The route-first, partition-second approach has been used in all reviewed studies. In this approach, the TSP route is constructed first, as if there were no drone. Following that, the drone nodes are selected and removed from the truck route. The Concorde TSP solver [19–21] is an exact algorithm which is able to find the optimal solution for a traditional TSP problem that includes instances with a large number of nodes. Many studies have used Concorde for the routing step [10,22–24].

1.2. The Original Solution to the TSP-D

Agatz et al. [10] have proposed several heuristics that are able to provide fast and acceptable solutions for instances of reasonable sizes. First, they constructed a truck-only tour using either the Concorde TSP solver or Kruskal's minimum spanning tree algorithm. Next, two approaches were proposed to classify some nodes as drone nodes. The first approach was a fast greedy heuristic, and the second approach was an exact partitioning algorithm based on dynamic programming. In both approaches, the order in which the nodes are visited remains unchanged. Both the greedy heuristic and the exact partitioning algorithm depend on the initial TSP tour. Therefore, the authors decided to start with a variety of initial tours, in order to determine whether they could obtain a better solution. They applied several heuristics based on the local search (LS) method, which modified the initial tour. Then they iteratively sought improvements. Several neighborhood functions were considered, which yielded several versions. In one of the versions, all neighborhoods were examined and the best neighbor generated in each iteration was selected.

The authors chose to run their experiments using the most challenging instances, where there were no limitations on the drone range and all locations could be visited by both drone and truck [23]. The greedy approach solved all instances with 100 nodes in a few seconds, but with limited savings in time relative to the original TSP tour. In contrast, the run time of the exact partitioning heuristics method increased tremendously with the number of nodes.

1.3. Solution-Based Metaheuristic Approaches to TSP-D

Solution-based metaheuristics have been used in previous studies to solve the TSP-D [6,11,25] and related FSTSP problems [12,22–24]. An initial tour for the simulated annealing algorithm [26] was implemented by Ponza [22] using a nearest-neighbor heuristic. To make a move (i.e., selecting a drone node for a sortie), several neighbors were randomly generated, and the roulette-wheels selection method [27] was used to choose a neighbor. De Freitas and Penna [23] first found the optimal solution for the TSP using the Concorde solver. The initial solution for the FSTSP was then defined, using a heuristic utilizing a greedy approach. This procedure removes a truck customer and assigns it to a drone, effectively partitioning the truck route into sub-routes. The initial solution was optimized using the variable neighborhood search [28]. Seven different neighborhood structures were used and randomly selected, resulting in a change in the original sequence of node visits. Similarly, de Freitas and Penna [23,29] used a randomized variable neighborhood descent heuristic with five different neighborhoods. Ha et al. [11] attempted to solve the TSP-D using two heuristics, TSP-LS and GRASP. The aim of this paper was to minimize the total operational cost of the two vehicles. In each iteration, a new giant tour (TSP tour) was generated using three different heuristics: k-nearest neighbors, k-cheapest insertion, and random insertion. A split algorithm built the minimum cost (min-cost) TSP-D solution from the giant tour solution by substituting a truck customer with a drone. After building the min-cost solution, the LS improved the solution using four move operators. The tests conducted in this study showed that the k-nearest neighbors generated a better min-cost TSP, when compared with k-cheapest insertion and random insertion. The quality of solutions obtained with GRASP outperformed those obtained with TSP-LS. However, TSP-LS was faster than GRASP. The proposed GRASP method has been modified by Marinelli et al. [6], such that the drone can launch from and join the truck not only at a customer node, but also along the route.

1.4. Population-Based Metaheuristic Approaches to TSP-D

Özoğu et al. [30] proposed a solution to the FSTSP: a hybrid approach combining a genetic algorithm (GA) [31] and Clarke and Wright's savings algorithm (CW) [32]. In this approach, the GA is used to assign the drone and truck to the customers. Following that, the sequence of the customers is determined using the CW. The aforementioned process continues until a specified number of iterations is reached. The authors proposed multiple

specific mathematical assumptions for the problem. The most consequential assumption is that the drone must launch and return to the truck at the same node, resulting in an increase in the waiting time of the truck. Ha et al. [33] used another hybrid GA, called the HGA, to solve the TSP-D with both minimum time (min-time) and minimum cost (min-cost) objectives. To improve the HGA's quality, the authors educated their offspring using LS methods. The authors designed a hill-climbing and first-improvement LS to address the min-time and min-cost objectives. In addition, the authors developed a restoration method to "educate" the TSP-D solution. Ferrandez et al. [34] used K-means clustering to help to find the truck stops, which were also the drone launch points. A GA was used to solve the truck route as a TSP. Houseknecht [35] implemented an extension of the classical ant system [36]. Using ant pairs is essential for the TSP-D, as a truck route and a drone route are needed every time the tour is constructed. When constructing its own path, every ant in the pair lays down a pheromone which is different from those the others. Every ant type is attracted to its own pheromone.

1.5. Discussion of Previous Approaches

Solution-based metaheuristic methods can be less destructive than population-based methods. For instance, genetic operators can drastically change the characteristics of the current solution. The use of generic genetic operators may lead to infeasible solutions. Restorative steps can repair the obtained solution, in some cases. Solution-based methods are more inclined to undergo intensification. Notably, several studies have utilized a considerable number of neighborhood structures (e.g., seven [23] or five [10,28]). The neighborhood structure for the next move has been chosen randomly or pseudo-randomly.

Population-based methods also work much more slowly than solution-based methods, as population-based methods process an entire population of solutions in each iteration. For instance, HGA was found to be 1.5 times slower than GRASP [33]. Notably, population-based metaheuristic approaches have been applied to this problem less frequently than single solution-based methods. The aforementioned approaches can achieve good results and provide the opportunity to further reduce the total route cost, compared with the exact cost in small to medium instances [10]. We considered GRASP [37,38], as it is a simple LS heuristic that utilizes a so-called restricted candidate list, which is a convenient way to store the set of candidate drone nodes. The neighborhood structure, (or equivalently, the next move to be applied) can be viewed as a search parameter. The optimal parameter choice largely depends on the problem type or the problem instance [39]. Self-adaptation has been shown to be beneficial for the selection of a genetic mutation operator for permutation encoding [40]. The search is guided by self-adaptively selected move operators: e.g., if there is a larger probability that moves with higher success rates, the instance in consideration will be identified and exploited, thereby improving the search outcome. This study aims to apply self-adaptive selection when searching the neighborhood in GRASP.

2. Proposed Greedy Randomized Adaptive Search Procedure for TSP-D

The route-first, partition-second approach was followed. The representative solution and the required data structures are described in Section 2.1. The details of the algorithm are presented in Section 2.2. Finally, the objective function is presented in Section 2.3.

2.1. Representative Solution and Supportive Data Structures

The candidate solution, *tspd*, is represented using an array that contains both truck and drone deliveries. The first and last indices refer to the depot. The elements in the array are ordered by the visit sequence. Each element in the array consists of a node *id* and node *label* (*t*: truck node, *d*: drone node, and *c*: combined node, where the location is visited by both the truck and drone). An example of a solution is shown in Figure 2, where Figure 2a illustrates the representative solution of the actual graph presented in Figure 2b. As part of the algorithm's initialization, the node IDs are initialized with the solution of the TSP tour. The node labels are initialized to *t*.

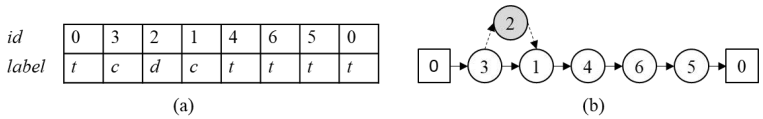


Figure 2. Representative solution: (**a**) represents the solution corresponding to the graph presented in (**b**).

The locations of the customer and the depot are stored in a location matrix, L. The column index represents the node *id*. The first row contains the x-coordinates, and the second row contains the y-coordinates. The values of the x and y-coordinates are read from the input data set.

In addition, two cost (adjacency) matrices are used to specify the cost over time (seconds): a drone cost matrix (C_D) and a truck cost matrix (C_T). Two cost matrices are used, as the drone and truck differ in terms of speed (s_D and s_T, respectively). The drone and truck speeds are read as inputs from the dataset. First, the Euclidean distance d_{ij} between nodes i and j for each i; j in L is computed using the x and y-coordinates. Following that, $C_{D_{ij}}$ and $C_{T_{ij}}$ are respectively computed using the following equations: $C_{D_{ij}} = \frac{d_{ij}}{s_D}$ and $C_{T_{ij}} = \frac{d_{ij}}{s_T}$.

2.2. Algorithm Details

The proposed GRASP approach is presented as Algorithm 1. *minTime* and *bestTour* are initialized to ensure the minimum time and the best attained TSP-D solution found so far, respectively. In line 3, the optimal TSP tour is obtained using the Concorde TSP solver. The following steps are repeated *maxTrials* times. The output of $optimalTspTour()$ is passed to $randomizedInitTspd()$, in order to construct an initial TSP-D solution. This is explained in detail in Section 2.2.1. Following that, $localSearch()$ is used to improve the acquired TSP-D solution, which is detailed in Section 2.2.2. $CompCost()$ then computes the cost (objective function) of the current TSP-D solution as *currentTime* (Section 2.3). If there are positive savings in cost, with respect to *minTime*, the current TSP-D solution is saved as the best solution found thus far, *bestTour*.

Algorithm 1: The GRASP algorithm

```
1  minTime = ∞;
2  bestTour = null ;
3  tspTour = optimalTspTour();
4  i = 0;
5  while (i < maxTrials) do
6      tspd = randomizedInitTspd(tspTour);
7      tspd = localSearch(tspd, swapSR, twoOptSR, P_m);
8      currentTime = CompCost(tspd);
9      if (currentTime < minTime) then
10         minTime=currentTime;
11         bestTour=tspd;
12     end
13     i = i + 1;
14 end
15 return bestTour,minTime;
```

2.2.1. Building the Randomized Initial TSP-D Tour

The main function of the initial TSP-D procedure (Algorithm 2) is to perform the initial partitioning of the available truck route into a truck route and a drone route, using the restricted candidate list (RCL).

First, the time saving obtained for all nodes (except the depot) when a node j is transformed from a truck node to a drone node is computed. The procedure identifies the

minimum amount of saved time for all nodes c_{min}, the maximum amount of saved time among all nodes c_{max}, and an array *savingTime* which contains the time saved by each node. The saved time is computed using Equation (1) and illustrated in Figure 3, where i is the node ID of the launch node, j is the node ID of the drone node, and k is the node ID of the rendezvous node. Calculating the time savings obtained for all nodes takes $\mathcal{O}(n)$ time.

Algorithm 2: *randmizedInitTSPD* Algorithm

1 $c_{min}, c_{max}, savingTime = calcSavings(tspTour)$;
2 Generate δ value, $\delta \in (0, 1)$;
3 $tau = c_{max} - \delta \times (c_{min} + c_{max})$;
4 $RCL = buildRCL(tau, savingTime)$;
5 $tspd = tspTour$;
6 **while** $(len(RCL) \neq 0)$ **do**
7 $\quad RCL = buildTspd(tspd, savingTime)$;
8 **end**

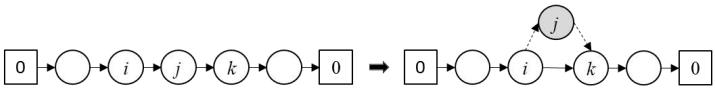


Figure 3. Transforming node j into a drone node.

$$saving = C_T[i][j] + C_T[j][k] - max(C_D[i][j] + C_D[j][k], C_T[i][k]) \quad (1)$$

The threshold value, *tau*, is computed using Equation (2) [41], where δ is a randomly generated value, $\delta \in (0, 1)$. The RCL is accordingly built by adding the customers that may be serviced by the drone; that is, all nodes i with $savingTime[i] \geq tau$. The *buildRCL* procedure requires $\mathcal{O}(n)$ time. Next, nodes are iteratively and randomly selected and removed from the RCL to be transformed into drone nodes, using the *BuildTspd* algorithm explained in Algorithm 3.

$$tau = c_{max} - \delta \times (c_{min} + c_{max}) \quad (2)$$

buildTspd was inspired by the greedy partitioning heuristic presented by Agatz et al. [10]. In that method, one of the three moves (*MakeFly*, which converts a truck node to a drone node, $PushLeft$, and *PushRight*) is randomly selected in each iteration. The selected move is applied to an unprocessed node. However, in this implementation, as presented in Algorithm 3, a candidate node, *candidateElement*, is chosen randomly and removed from the RCL, in order to be converted into a drone node. *candidateElement* can be a drone node if it has a predecessor and a successor, meaning that it is not a boundary node. In addition, it must not lie in a sortie. Once this move is successful, the feasibility of the $PushLeft$ and *PushRight* moves for the sortie under consideration is examined. If both moves can be feasibly implemented while improving the cost, then one of them is greedily selected, such that the cost saving is maximized. Otherwise, the feasible move is applied if it improves the cost.

$PushLeft$ inserts the truck node to the left of the launch node into the sortie, as shown in Figure 4. The move is only feasible if the node preceding the launch node is a truck node other than the depot and is not part of a sortie. The time taken by the sortie before applying $PushLeft$, $beforeLeft$, is calculated using Equation (3). Equation (4) computes the time taken by the sortie after applying $PushLeft$, $afterLeft$. The time saved is computed by taking the difference between $beforeLeft$ and $afterLeft$. If the procedure results in time savings greater than zero, the move is accepted. The *PushRight* operator is symmetric with respect to $PushLeft$. Converting the *candidateElement* into a drone node in lines 4–10 of Algorithm 3 requires $\mathcal{O}(1)$ time. The complexity of both moves, including the work

required to compute the savings obtained by the moves, is $\mathcal{O}(1)$ time. However, the time required by the *CompCost* step at line 1 of the algorithm is $\mathcal{O}(n)$ (Section 2.3), resulting in a time of $\mathcal{O}(n)$ for *buildTspd*. The *buildTspd* algorithm is called a number of times equal to the length of the RCL as in line 6 from Algorithm 2. Accordingly, the time complexity required by the *randomizedInitTspd* algorithm is $\mathcal{O}(n^2)$.

Algorithm 3: *buildTspd* algorithm

```
1  currentTime = CompCost(tspd);
2  candidateElement =randomly select an element from RCL;
3  Find the position i of candidateElement in tspd;
4  if (candidateElement can be a drone node) then
5      launch = tspd[i − 1];
6      drone = tspd[i];
7      rendezvous = tspd[i + 1];
8      Set labels for nodes tspd[i − 1], tspd[i], and tspd[i + 1] to 'c', 'd', and 'c',
         respectively;
9      currentTime = currentTime − savingTime[i]
10 end
11 if (both PushLeft and PushRight are feasible) then
12     Greedily select and apply the move that maximizes cost savings to tspd;
13 else
14     if (PushLeft is feasible) then
15         Apply PushLeft to tspd;
16     else
17         if (PushRight is feasible) then
18             Apply PushRight to tspd;
19         end
20     end
21 end
22 return RCL, tspd
```

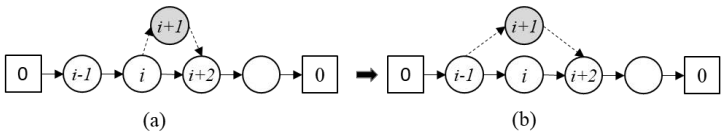


Figure 4. *PushLeft* move: (**a**) displays the current solution before applying *PushLeft*; and (**b**) displays the solution after applying *PushLeft*.

$$beforeLeft = C_T[i-1][i] + max(C_D[i][i+1] + C_D[i+1][i+2], C_T[i][i+2]) \quad (3)$$

$$afterLeft = max(C_D[i-1][i+1] + C_D[i+1][i+2], C_T[i-1][i] + C_T[i][i+2]) \quad (4)$$

2.2.2. Optimizing the Obtained Solution with Local Search

The LS procedure helps to improve the initial TSP-D solution by applying two different moves: *Swap* and *TwoOpt* [10]. The LS procedure is explained, followed by a description of the two moves.

A general structure for the LS procedure is shown in Algorithm 4. After the initialization step, an iterative process starts. In each iteration, either the *Swap* or *TwoOpt* move is selected in a self-adaptive manner, based on the move probability, p_m, as follows (lines 7–12). A random number, $rand \in [0, 1]$, is generated. If the value of *rand* is less than p_m, the *Swap* move is selected; otherwise, the *TwoOpt* move is selected. The selected move is applied to the current solution, *tspd*. If the resulting neighbor has a lower cost, the move is accepted. Otherwise, the acceptance criterion is tested. The success ratio of the chosen

move is updated. The procedure is repeated until the termination condition or *convergence* is reached.

Algorithm 4: *localSearch* Algorithm

```
1  j = 0;
2  initialize loop control parameters;
3  while (termination condition not satisfied and j < convergence) do
4      if (windowSize iterations have elapsed) then
5          update p_m based on Equation (5);
6      end
7      rand=generate a random number ∈ [0,1];
8      if (rand < p_m) then
9          apply swap move to tspd and update its success ratio;
10     else
11         apply twoOpt move to tspd and update its success ratio;
12     end
13     examine move acceptance criteria and accept move accordingly;
14     if (the move was not successful) then
15         increment j;
16     else
17         reset j;
18     end
19     update iteration control parameters;
20 end
```

The move selection probability, p_m, is updated consistently after a certain number of iterations, *windowSize*, based on the success ratio of the *Swap* and *TwoOpt* moves: *swapSR* and *twoOptSR*, respectively (lines 4–6 in Algorithm 4). The success ratio is computed as the number of times that the move is successfully applied, divided by the total number of times that the move was applied during the past *windowSize*. If *swapSR* is better than *twoOptSR*, p_m is updated such that *Swap* has a higher probability of being chosen and vice versa, as shown in Equation (5), where $\beta \in (0,1)$ is a learning rate parameter.

$$p_m = \begin{cases} p_m + \beta \times p_m, & \text{if } swapSR \geq twoOptSR. \\ p_m - \beta \times p_m,, & \text{otherwise.} \end{cases} \quad (5)$$

Moreover, in order to control the computational time, the convergence is examined and the LS procedure is stopped if it fails to improve the solution after a specific number of iterations (*convergence*), as shown in lines 1 and 14–18.

Two variants of GRASP were considered, according to the LS procedure used: hill-climbing local search (HCLS) [42] and LS with simulated annealing (SA) [26]. These variants differ in terms of the initialization step and the acceptance criteria (lines 2 and 13 in Algorithm 4, respectively). In the HCLS variant, the loop control parameter is a simple loop counter i, which is initialized to zero. The termination condition is satisfied when i reaches the maximum number of allowed iterations, *maxIter*. In this variant, a move is accepted only if the resulting neighbor has a lower cost than that of the current solution.

In the second variant, SA, the loop is controlled by a temperature parameter, T_i. T_i is initialized to an initial temperature, T_0, and updated using a geometric cooling schedule, as shown in Equation (6), where $\gamma \in (0,1)$ [26]. The acceptance criterion in SA differs from that in the HCLS variant for inferior moves. A sub-optimal move can be accepted, based on a certain probability $P(\Delta E, T)$ that it follows the Boltzmann distribution, as shown in Equation (7), where ΔE represents the difference between the objective function (cost) of the current solution and that of the resulting neighbor. A random number, *randP*, is generated. If the value of *randP* is less than the probability $P(\Delta E, T)$, the move is accepted; otherwise,

the move is not accepted. The termination condition in the SA variant is satisfied when the temperature T_i exceeds the final temperature T_f.

$$T_i = \gamma \times T_i \tag{6}$$

$$P(\Delta E, T) = e^{\frac{\Delta E}{T_i}} \tag{7}$$

The *Swap* Move: In the *Swap* move, two distinct nodes are randomly selected and swapped (Figure 5). A swapped node may be a truck node, a drone node, or a combined node, but it cannot be the depot [22].

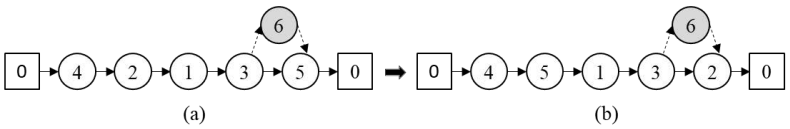


Figure 5. Example of the *Swap* move: (**a**) displays the TSP-D solution, *tspd*, before applying *Swap*; and (**b**) displays the *tspd* after nodes 2 and 5 have been swapped.

The *TwoOpt* Move: In the *TwoOpt* move, two edges are removed from the tour and replaced with another set of edges, in order to obtain a valid tour. The removed edges must be between truck or combined nodes [22]. The first step in applying *TwoOpt* is to select the two origin nodes of the selected edges randomly. In Figure 6a, nodes 4 and 2 are selected. Two edges are thus removed: $(4, 6)$ and $(2, 0)$. Then, two new edges are created, as shown in Figure 6b: $(4, 2)$ and $(6, 0)$. As a result, the sortie path is reversed, as shown in Figure 6b.

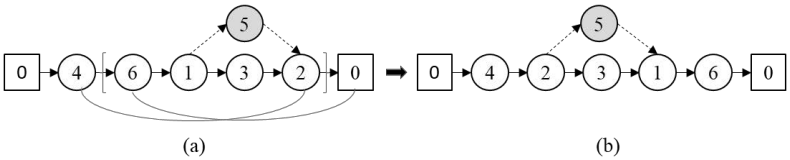


Figure 6. An example of a *Swap* move: (**a**) displays the TSP-D solution, *tspd*, before applying *Swap*; and (**b**) displays the *tspd* after nodes 2 and 5 have been swapped.

As twoOpt randomly selects two edges, there are several invalid cases that must be taken into consideration, as follows:

1. If the selected origin node or its destination node happen to be inside a sortie, as shown in Figure 7;

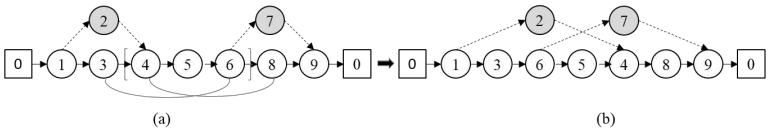


Figure 7. An example of an invalid *TwoOpt* move applied to the graph in (**a**), where origin node 3 is a truck node inside a sortie and the origin node 6 is a launch node for a different sortie, resulting in an invalid tour in (**b**).

2. If both selected origin nodes are launch nodes and their destinations are rendezvous nodes, as shown in Figure 8;

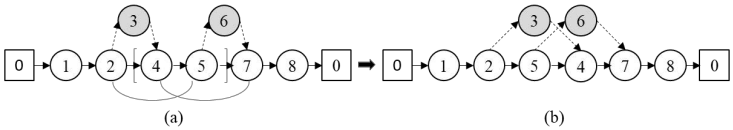


Figure 8. An example of an invalid *TwoOpt* move applied to the graph in (**a**), where origin nodes 2 and 5 are launch nodes and their destination nodes 4 and 7 are rendezvous nodes, resulting in an invalid tour in (**b**).

3. If the selected origin or its destination node happens to be a common node between different sorties, as shown in Figure 9.

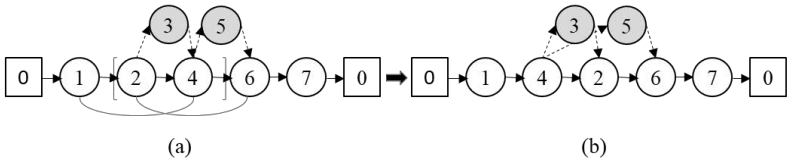


Figure 9. An example of an invalid *TwoOpt* move applied to the graph in (**a**), where destination node 4 is a launch and rendezvous node for different sorties, resulting in an invalid tour in (**b**).

The complexity of *Swap* and *TwoOpt* moves, including the work required to compute the savings obtained by the moves, is $\mathcal{O}(n)$.

2.3. Objective Function

We aimed to minimize the total delivery time required to serve all customers starting from and returning to the depot, as shown in the objective function (8), where customer i is where the sub-route starts and customer k is where the sub-route ends. Customer j is served by the drone [23].

$$minimize \sum_{s \in subroute} max\left(C_D[i,j] + C_D[j,k], \sum_{i}^{k-1} C_T[i,i+1]\right) \tag{8}$$

The *CompCost* procedure, presented in Algorithm 5, computes the cost of the full *tspd* tour. Firstly, the *currentTime* is initialized to 0. Following that, the nodes in the tour are examined in sequence. If the node is a launch node, the sortie time is computed. A launch location i and a rendezvous location k are identified. The sortie time is calculated by taking the maximum time between the *droneTime* and *truckTime*. The *droneTime* is calculated by adding the time taken by the drone to move from the launch location to the drone location, to the time taken by the drone to move from the drone location to the rendezvous location. The *truckTime* is computed by calculating the time taken by the truck to move from the launch location to the rendezvous location. In line 16, the value of the counter i is updated to that of the rendezvous location k, as the entire sortie has been processed. Otherwise, if the location is not in a sortie, it is visited by the truck and the cost required by the truck is computed by calculating the time taken by the truck to move from its current location to the following location.

Algorithm 5: *CompCost* algorithm

```
currentTime = 0;
i = 0;
while (i < len(tspd)) do
    if (tspd[i] is launch location for drone) then
        truckTime = 0;
        launch = tspd[i];
        drone = tspd[j];
        rendezvous = tspd[k];
        droneTime = C_D[launch, drone] + C_D[drone, rendezvous];
        t = i;
        while (t < k − 1) do
            truckTime = truckTime + C_T[t, t + 1];
            t = t + 1;
        end
        sortieTime = max(droneTime, truckTime);
        currentTime = currentTime + sortieTime;
        i = k;
    else
        truckTime = C_T[i, i + 1];
        currentTime = currentTime + truckTime;
        i = i + 1
    end
end
return currentTime
```

3. Experimental Setup

The benchmark dataset is described in Section 3.1. The algorithm was implemented in Python. The algorithm parameters are explained in Section 3.2, and the model evaluation is presented in Section 3.3. We used an Ubuntu 16.04 LTS desktop powered by a 3.60 GHz × 8 Intel Xeon(R) CPU E5-1620 with 31.3 GiB of RAM and a 427.9 GB SSD.

3.1. Benchmark Dataset

We used the publicly-available dataset "Instances for the TSP with Drone" [43]. This has been used in several studies [10,23]. The dataset categorizes instances based upon the distribution type, instance size, and vehicle speed configuration. The categories are as follows. We used two different distribution types, based on how each instance was generated in the benchmark dataset. In a uniform distribution, the coordinates are generated independently and uniformly; in a single-center distribution, the coordinates are generated using an angle and a distance drawn from a normal distribution. The dataset contains a wide variety of instance sizes, ranging from 5 to 500 customer locations. The experiments in this study were applied using instances of sizes $10, 20, 50, 75$, and 100. As for the vehicle speed configuration (α), we utilized two different relative speed configurations for the vehicles. With $(\alpha = 1)$, the truck and drone have the same speed; with $(\alpha = 2)$, the drone is two times faster than the truck.

For each (type, size, α) combination, the dataset contains 10 different instances. In each instance, the vehicle speed configuration (α), the total number of nodes (including the depot), and for each node, the customer ID and location specified using the x and y-coordinates are defined.

3.2. Parameter Initialization

Table 1 summarizes the initial values of the parameters used in the proposed GRASP approach. Several parameters were adaptively set using the instance size, n. For example, the maximum number of iterations in the GRASP algorithm (*maxTrials*) and in the HCLS

variant (*maxIter*); the window size used to update the move probability parameter in LS, *windowSize*; and the *convergence* used to signal LS convergence. When n was less than 50, *maxTrials* was set to 50×10. In addition, p_m was initialized to 0.5, in order to allow for equal chances of *Swap* and *TwoOpt* moves occurring at the beginning of the search. The learning rate, β, was initialized to 0.02, a common default value for standard artificial neural networks [44].

Table 1. Algorithm parameter initialization.

Parameter	Initial Value	Description
maxTrials	$n \times 10$	Max. number of iterations in GRASP
p_m	0.5	Move selection probability in LS
β	0.02	Learning rate for updating p_m
windowSize	$\frac{n}{2}$	Window size for updating p_m
maxIter	n	Max. number of iterations in HCLS
convergence	$n \times 0.3$	Max. number of iterations without improvement in LS
T_0	500	Initial temperature in SA
T_f	0.01	Final temperature in SA
γ	0.95	SA cooling schedule coefficient

3.3. Model Evaluation

The outcomes of the proposed GRASP approach were compared with the results of two studies. Agatz et al. [10] solved the TSP-D problem by proposing a route-first, cluster-second heuristic based on the local search (LS) method. On the other hand, de Freitas et al. [23] first used a mixed-integer linear-programming solver, and then applied a variable neighborhood search metaheuristic to construct sorties and improve the solution. The effectiveness of the model was measured using the complete tour duration performance measure (in seconds, *secs*). The efficiency of the model was measured using the run-time performance measure (in *secs*). As GRASP is a stochastic algorithm, the run for each instance was repeated ten times, and the average value over each of the obtained performance measures was reported.

To obtain a solid statistical basis for the model comparisons, non-parametric statistical tests were used. The Wilcoxon signed-rank test was used to test the median differences among paired data points (pairwise comparisons) [45]. The Friedman test was used to compare more than two related data samples [46]. The significance level was set to 5%. The statistical analysis was conducted using the SPSS version 1.0.0.1012 software.

4. Results

This section reports the results for the two variants proposed: the GRASP HCLS variant and the GRASP SA variant (Section 4.1). The better-performing GRASP variant was then compared with two the rival algorithms [10,23]

4.1. Results of the Proposed GRASP Algorithm for TSP-D

The GRASP HCLS variant experiment used 200 test instances, $(10, 20, 50, 75, 100)$ in size, of uniform and single-center type, and with a relative truck/drone speed of $\alpha = 1$ or $\alpha = 2$. All parameters were initialized to the settings stated in Table 1.

Table 2 summarizes the results obtained using the proposed GRASP approach with HCLS. The values reported in the table represent average values of the best-known solutions obtained over the ten runs for each of the ten different instances for each of the (type, size, α) combinations. The table is divided according to the distributions; that is, uniform followed by single-center. In each distribution, the results are sectioned by the relative drone/truck speed (α). In the table, the column Size represents the instance size, the column Cost represents the average trip duration of the obtained TSP-D solution (in *secs*), the column Runtime represents the average running time (in *secs*), the column

Swap represents the average number of times in which a *Swap* move was applied, the column *TwoOpt* represents the average number of times that a *TwoOpt* move was applied, the column *Swap* Success represents the average number of times that a *Swap* move was successful (i.e., it improved the TSP-D solution and decreased its cost), and finally, the column *TwoOpt* Success represents the average number of times that a *TwoOpt* move was successful. Moreover, instances of size 100 were run on different machines; hence, the run time is not reported (NA).

Table 2. Results obtained with the proposed GRASP approach, HCLS variant.

	Size	**Cost**	**Runtime**	***Swap***	***TwoOpt***	***Swap* Success**	***TwoOpt* Success**
Uniform Distribution							
$\alpha = 1$	10	281.46 ± 1.38	4.05	5107.89	4709.60	338.84	0.02
	20	363.83 ± 1.64	11.13	4141.19	4058.06	89.94	0.13
	50	554.10 ± 0.57	57.01	3848.66	3847.08	23.01	0.07
	75	647.18 ± 1.12	121.85	3804.58	3784.04	9.60	0.13
	100	731.32 ± 0.78	NA	12,867.77	12,851.60	13.75	0.17
	Avg.	516.99 ± 1.10	48.51	5954.02	5850.07	95.03	0.10
$\alpha = 2$	10	224.74 ± 3.67	2.32	4246.95	4010.12	105.2	0.00
	20	306.39 ± 4.55	4.98	3963.70	3888.41	43.81	0.04
	50	472.88 ± 2.90	18.89	3807.67	3806.22	11.17	1.43
	75	551.92 ± 3.12	32.69	3795.46	3791.46	7.98	1.42
	100	632.30 ± 2.41	NA	15,166.70	15,204.00	19.08	3.36
	Avg.	437.65 ± 3.33	14.72	6196.10	6140.04	37.45	1.25
Single Center							
$\alpha = 1$	10	385.49 ± 8.14	4.235	5500.51	5027.52	484.10	1.00
	20	546.13 ± 3.91	11.19	4172.03	4085.86	101.39	0.44
	50	776.97 ± 1.28	56.69	3838.70	3835.01	21.20	0.11
	75	1039.74 ± 1.47	121.95	3789.25	3812.06	11.97	0.10
	100	1238.28 ± 1.60	NA	15,177.63	15,201.60	23.89	0.10
	Avg.	797.32 ± 3.28	48.52	6495.62	6392.45	128.51	0.35
$\alpha = 2$	10	277.00 ± 6.17	2.39	4406.14	4143.45	158.16	5.07
	20	402.92 ± 7.09	4.53	3920.74	3856.99	33.13	2.35
	50	604.79 ± 6.22	13.22	3787.35	3792.57	7.09	1.19
	75	851.01 ± 7.03	23.03	3774.44	3775.38	3.83	0.83
	100	995.72 ± 9.10	NA	15,145.04	15,128.24	10.71	5.29
	Avg.	626.29 ± 7.12	10.80	6206.74	6139.33	42.58	2.95

Table 2 shows that the *Swap* move was used more successfully in smaller-sized instances than in larger-sized instances, while the *TwoOpt* move was more successful in larger-sized instances. Nonetheless, the *Swap* move was more successful than the *TwoOpt* move overall. The standard deviation of the solution cost in instances with a uniform distribution tended to be lower than that observed in instances with a single-center distribution. In addition, the standard deviation of the solution cost in instances in which $\alpha = 1$ tended to be lower than instances in which $\alpha = 2$. Concerning the computational time, when $\alpha = 1$, the algorithm ran for longer. The run time did not differ between uniform and single-center instances.

Notably, the moves were often not accepted in the GRASP HCLS variant, which suggests that it became stuck in a local optimum. To address this problem, the GRASP SA variant for LS, in which sub-optimal moves can be accepted, was examined. In this experiment, 160 test instances of sizes $(10, 20, 50, 75)$, with both uniform and single-center types and vehicle speed configurations of $\alpha = 1$ and $\alpha = 2$ were utilized. All parameters were initialized to the settings stated in Table 1.

Table 3 summarizes the results obtained using the proposed GRASP approach with the SA variant. The values reported in the table represent the average values of the best known solutions obtained over the ten runs for each of the ten different instances in each of the (type, size, α) combinations. The column definitions are the same as those used in Table 2.

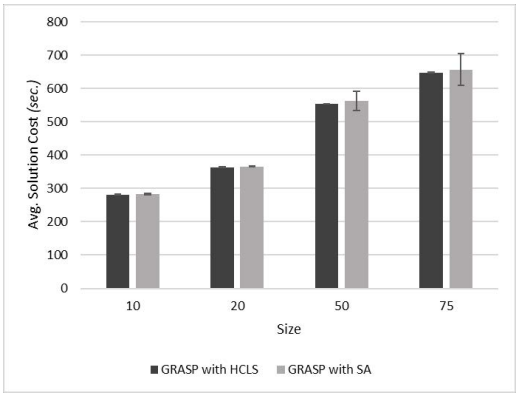
Table 3. Results obtained with the proposed GRASP approach, SA variant.

	Size	Cost	Runtime	*Swap*	*TwoOpt*	*Swap* Success	*TwoOpt* Success
Uniform Distribution							
$\alpha = 1$	10	283.71 ± 1.91	3.37	3185.49	2948.50	1173.77	21.21
	20	366.08 ± 0.74	15.94	12,194.30	10,935.50	4031.44	140.95
	50	561.93 ± 58.04	87.76	25,107.27	23,404.50	4050.01	221.54
	75	656.76 ± 94.22	282.62	42,641.81	40,542.60	4415.07	269.36
	Avg.	467.12 ± 38.73	97.42	20,782.23	19,457.80	3417.57	163.27
$\alpha = 2$	10	242.09 ± 9.60	2.54	4597.52	4152.01	1590.50	349.87
	20	321.88 ± 5.34	12.25	15,570.18	13,879.04	4604.19	711.19
	50	568.37 ± 94.31	60.01	30,890.69	26,462.18	7064.08	2919.46
	75	648.30 ± 113.28	164.62	54,327.14	52,851.94	5357.96	4451.74
	Avg.	445.16 ± 55.58	59.86	26,346.38	24,336.29	4654.18	2108.07
Single Center							
$\alpha = 1$	10	380.46 ± 7.74	3.29	3557.43	3272.93	1174.02	269.90
	20	556.99 ± 3.09	16.33	11,851.55	10,657.87	3878.31	137.46
	50	787.78 ± 64.34	85.13	25,643.55	23,858.43	4372.77	219.97
	75	1058.13 ± 89.51	244.23	42,661.80	40,628.15	4396.45	262.65
	Avg.	695.84 ± 41.17	87.24	20,928.58	19,604.30	3455.39	222.50
$\alpha = 2$	10	295.49 ± 9.24	2.51	4231.81	3831.19	1437.96	293.26
	20	428.09 ± 12.55	12.33	15,760.87	14,192.32	4427.88	2114.55
	50	723.88 ± 107.93	58.21	32,542.07	31,949.98	4799.59	4409.37
	75	1030.92 ± 104.36	154.65	55,666.13	57,962.20	5123.6	6297.56
	Avg.	619.60 ± 58.52	56.92	27,050.22	26,983.92	3947.26	3278.69

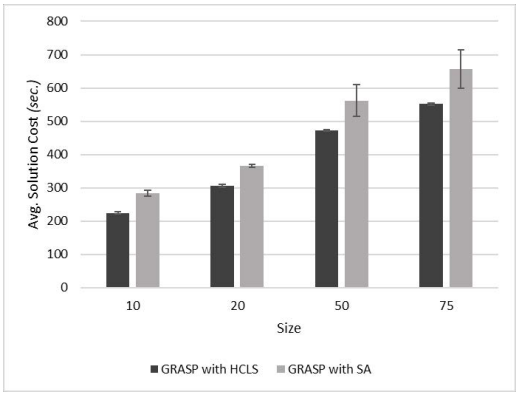
Table 3 shows that, as the instance size increased, the success ratios of both moves also increased. Nonetheless, the *Swap* move was still more successful than the *TwoOpt* move in all cases, except for the case (type = single-center, size = 75, $\alpha = 2$). The standard deviation of the solution cost in uniform instances tended to be slightly lower than that for single-centered instances. We also observed that the standard deviation of the solution cost in instances in which $\alpha = 1$ tended to be lower, compared to that for instances having $\alpha = 2$.

As can be seen in Figure 10, overall, the GRASP SA variant performed worse than or approximately equal to the GRASP HCLS variant. The overall average solution cost for the models obtained using the GRASP HCLS variant was 517.79 *sec*s, compared to 556.90 *sec*s for the models generated using the GRASP SA variant. The standard deviation of the solution cost obtained with the GRASP SA models was higher than that of the GRASP HCLS model. A possible explanation for the large gap between runs of instances of the same size in the former is that the LS with SA accepts sub-optimal moves.

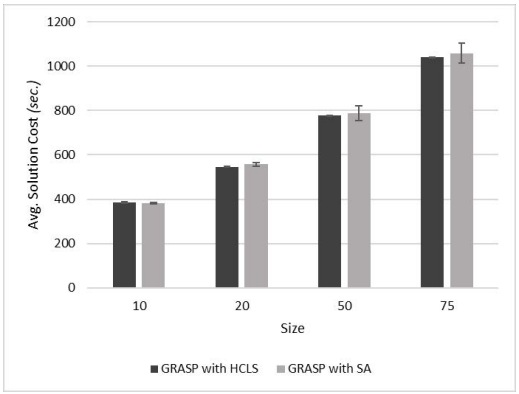
The Wilcoxon signed-rank test was used to compare the models obtained with the two GRASP variants. For a valid comparison, 160 instances were used for each model. A statistically significant difference was detected between the means of the two models ($p < 0.001$). Based on this, the HCLS variant was considered to perform better than the SA variant.



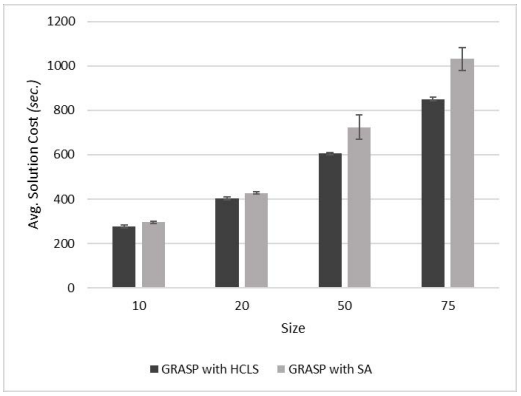
(**a**) Uniform, $\alpha = 1$.



(**b**) Uniform, $\alpha = 2$.



(**c**) Single-center, $\alpha = 1$.



(**d**) Single-center, $\alpha = 2$.

Figure 10. Comparison of the average solution cost obtained with the two GRASP variants–GRASP with HCLS vs. GRASP with SA: (**a**) Uniform distribution, $\alpha = 1$. (**b**) Uniform distribution, $\alpha = 2$. (**c**) Single-centered distribution, $\alpha = 1$. (**d**) Single-centered distribution, $\alpha = 2$.

4.2. Evaluation against Rival Algorithms

Based on the conclusion presented in Section 4.1, the GRASP HCLS variant outperformed the SA variant. The better performing GRASP variant was compared with the rival algorithms, LS [10] and HGVNS [23], for the TSP-D problem, where the endurance of the drone was set to infinity and the service time for the drone launch and return was set to zero. The parameter settings for both rival algorithms can be found in the study of de Freitas and Penna [23].

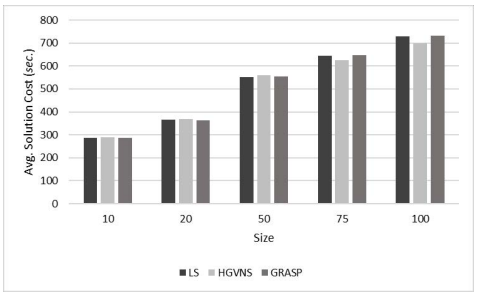
The experiment used 200 instances, as described in Section 3.1. Table 4 reports the average values (in *sec*s) for the best-known solutions for the ten instances for each (type, size, α) combination. The column definitions are the same as those used in Table 2, and are sectioned according to the algorithm addressed. Figure 11 compares the average solution cost results. Similarly to the findings presented by de Freitas and Penna [23], the algorithms generally performed better for instances with a uniform distribution, compared to instances with a single-center distribution, for all values of α. The proposed GRASP approach performed better in instances where $\alpha = 1$; however, in instances where $\alpha = 2$, that size was found to play a role. Accordingly, the proposed GRASP approach had a better or approximately equal performance in instances with small size (i.e., 10 and 20). Conversely, in instances with large size (i.e., 50 and 75), our approach had a worse or approximately equal performance.

Table 4. Performance comparison of the models obtained by the LS [10], a TSP-D version of the HGVNS [23], and the proposed GRASP HCLS variant, showing the average solution cost in *sec*s.

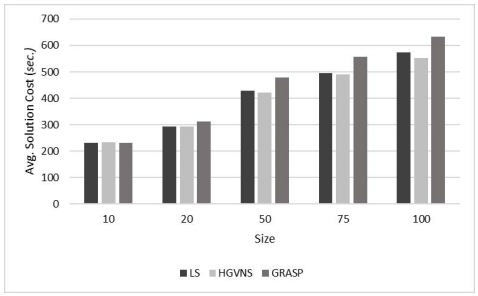
	Size	**Cost**		
		LS [10]	**HGVNS [23]**	**GRASP**
Uniform Distribution				
$\alpha = 1$	10	286.82	289.82	285.69
	20	365.38	368.5	364.54
	50	550.38	559.20	554.94
	75	645.55	624.32	647.18
	100	729.40	698.42	731.10
	Avg.	515.51	508.05	515.16
$\alpha = 2$	10	231.29	233.20	230.75
	20	293.59	293.60	312.82
	50	428.63	420.80	478.31
	75	495.90	490.40	557.15
	100	572.53	553.43	632.30
	Avg.	404.39	398.29	437.65
Single Center				
$\alpha = 1$	10	379.29	364.90	399.55
	20	529.15	553.53	553.79
	50	763.28	784.32	779.33
	75	1017.04	978.32	1042.00
	100	1203.42	1193.95	1238.28
	Avg.	778.44	775.00	797.32
$\alpha = 2$	10	278.22	291.36	287.13
	20	384.87	364.00	416.47
	50	554.58	593.50	617.43
	75	741.38	754.40	861.21
	100	891.28	900.10	995.72
	Avg.	570.07	580.67	626.29

The average solution costs over all instances (in *sec*s) were as follows: The HGVNS scored 565.52, closely followed by the LS (567.10) and GRASP (594.11). The Friedman test applied under the null hypothesis of equal performance of all algorithms obtained a p-value of ($p = 0.074$). Since the p-value was larger than the significance level (0.05), the null hypothesis is accepted, indicating that no statistically significant differences were detected among all the means of solution costs obtained from the three algorithms on multiple datasets.

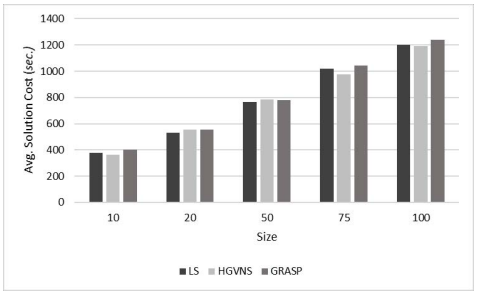
Considering the efficiency of the three algorithms under consideration, all three algorithms start by constructing the initial TSP tour using the Concorde TSP solver. Both of the rival algorithms, LS and HGVNS, adapt the best-improvement search strategy, which exhaustively explores the neighborhoods and returns the solution having the minimum cost. The proposed GRASP algorithm, on the other hand, employs the first-improvement strategy, accepting the first neighbor having a cost that is lower than that of the current solution. Excluding the time required for obtaining the initial TSP tour, a single iteration of the LS procedure requires $\mathcal{O}(n^3 \log n)$ [10]. A single iteration of the HGVNS algorithm runs in $\Theta(n^2|\mathcal{N}^2|)$ [23], where $|\mathcal{N}|$ is the number of neighborhoods considered. On the contrary, and considering the value of $maxIter$ in HCLS to be equal to n, a single iteration of GRASP requires $\mathcal{O}(n^2)$ time. Accordingly, the proposed algorithm has lower computational requirements than the two rival algorithms, LS and HGVNS.



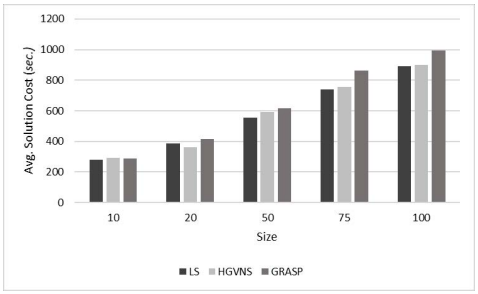
(**a**) Uniform, $\alpha = 1$.



(**b**) Uniform, $\alpha = 2$.



(**c**) Single-center, $\alpha = 1$.



(**d**) Single-center, $\alpha = 2$.

Figure 11. Comparison of the average solution cost (in *secs*) obtained with the HCLS variant of the proposed GRASP, HGVNS [23], and LS [10] for instances of size 10, 20, 50, 75, and 100: (**a**) Uniform distribution, $\alpha = 1$. (**b**) Uniform distribution, $\alpha = 2$. (**c**) Single-centered distribution. $\alpha = 1$ (**d**) Single-centered distribution, $\alpha = 2$.

5. Discussion

The GRASP HCLS variant focuses on improving the obtained TSP-D solution using relatively optimal moves only, whereas in the SA variant, sub-optimal moves with a given probability can be accepted. Overall, the GRASP SA variant was shown to have a worse or approximately equal performance to the HCLS variant. Setting the initial temperature to a high value may have played a role. Furthermore, setting γ to 0.95 and using the geometric schedule to update the temperature may have resulted in the temperature decreasing faster than needed. It is possible that changing the values of the SA parameters could result in better performance.

Comparing the performance of the proposed GRASP HCLS variant with the two rival algorithms, LS and HGVNS, the proposed approach performed better in instances where $\alpha = 1$ and instances of smaller sizes where $\alpha = 2$. Overall, the Friedman test showed that the proposed GRASP was comparable with the two state-of-the-art rival algorithms specified earlier in terms of solution cost. It is possible that the number of iterations in the LS procedure was not sufficient to effectively improve the solution. Furthermore, the random δ value heavily affects the size of the RCL. Thus, a small δ value may result in a small and inadequate RCL. However, the proposed algorithm had lesser computational complexity than the other two.

6. Conclusions

In this study, we addressed the TSP-D problem and its variants. Various algorithms, including population-based metaheuristic algorithms and solution-based metaheuristic algorithms, have been proposed for TSP-D and its variants, such as the FSTSP. We used GRASP as a single-solution metaheuristic algorithm. The proposed algorithm starts by constructing the entire truck route as a classical TSP. The obtained route is then partitioned into truck nodes and drone nodes, yielding an initial TSP-D solution. Following that, the proposed GRASP algorithm is applied to minimize the total delivery time for the initial TSP-D solution. Two versions of the LS procedure in GRASP were used: HCLS and LS with SA. Both versions use a self-adaptive neighborhood. The proposed approach was tested

on the "Instances of TSP with Drone" benchmark dataset. Generally, the HCLS variant outperformed the SA variant of GRASP. Subsequently, the performance of the proposed GRASP HCLS variant was compared with two state-of-the-art algorithms: LS [10] and HGVNS [23]. Overall, the proposed approach was found to have a performance comparable to those of the two rival algorithms. Nonetheless, under some (type, size, α) combinations, our approach outperformed the rival algorithms.

The main contribution of this study is the self-adaptive selection of the search neighborhood, which was implemented for two neighborhoods (associated with *Swap* and *TwoOpt* moves), but which could be easily extended to a larger number of neighborhoods. The self-adaptive selection of the search neighborhood for the TSP-D problem may also inspire other methods, which in turn may significantly outperform existing ones. We recommend that the proposed GRASP approach be further improved by increasing the number of iterations in the LS procedure and tuning the δ value. Moreover, it would be interesting to investigate different parameters for the SA variant and to add more neighborhood alternatives into the search.

Author Contributions: Conceptualization, S.A. (Sarab AlMuhaideb); methodology, S.A. (Sarab AlMuhaideb) and T.A.; software, T.A., S.A. (Sara Alamri). Y.A., L.A. and H.A.; validation, T.A., S.A. (Sara Alamri), Y.A., L.A. and H.A.; writing—original draft preparation, S.A. (Sarab AlMuhaideb), T.A., S.A. (Sara Alamri) and Y.A.; writing—review and editing, S.A. (Sarab AlMuhaideb); visualization, L.A. and H.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in this study are openly available in Instances for the TSP with Drone at doi:10.5281/zenodo.1204676, reference number [43].

Acknowledgments: The authors would like to thank the anonymous reviewers for their constructive comments. Special thanks to Heba Kurdi, Department of Computer Science, King Saud University, for her valuable advice that helped improve the quality of this work.

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

Abbreviations

The following abbreviations are used in this manuscript:

CW	Clarke and Wright's Savings Algorithm
FSTSP	Flying Sidekick Traveling Salesman Problem
GA	Genetic Algorithm
GRASP	Greedy Randomized Adaptive Search Procedures
GVNS	General Variable Neighborhood Search
HCLS	First-choice Hill-Climbing Local Search
HGA	Hybrid Genetic Algorithm
HGVNS	Hybrid General Variable Neighborhood Search
IP	Integer Programming Model
LS	Local Search
NP-Hard	Non-Deterministic Polynomial-Time Hardness Problem
RCL	Restricted Candidate List
RVND	Randomized Variable Neighborhood Descent
SA	Simulated Annealing
TSP	Traveling Salesman Problem
TSP-D	Traveling Salesman Problem with Drone
VND	Variable Neighborhood Descent
VNS	Variable Neighborhood Search

References

1. Scott, J.; Scott, C. Drone delivery models for healthcare. In Proceedings of the 50th Hawaii International Conference on System Sciences, Hawaii, HI, USA, 4–7 January 2017.
2. Frachtenberg, E. Practical Drone Delivery. *Computer* **2019**, *52*, 53–57. [CrossRef]
3. McNabbon, M. Walmart Drone Delivery: Investment in DroneUp Could Change the Race to Dominate the Last Mile. Available online: https://dronelife.com/2021/06/17/walmart-drone-delivery-investment-in-droneup-could-change-the-race-to-dominate-the-last-mile/ (accessed on 17 June 2021).
4. Straight, B. Workhorse Now Making Residential Deliveries with HorseFly Drone. Available online: https://www.freightwaves.com/news/technology/drone-delivery-test-underway-in-cincinnati (accessed on 18 May 2018).
5. Ioannidis, J.P. Global perspective of COVID-19 epidemiology for a full-cycle pandemic. *Eur. J. Clin. Investig.* **2020**, *50*, e13423. [CrossRef]
6. Marinelli, M.; Caggiani, L.; Ottomanelli, M.; Dell'Orco, M. En route truck–drone parcel delivery for optimal vehicle routing strategies. *IET Intell. Transp. Syst.* **2018**, *12*, 253–261. [CrossRef]
7. Khoufi, I.; Laouiti, A.; Adjih, C. A survey of recent extended variants of the traveling salesman and vehicle routing problems for unmanned aerial vehicles. *Drones* **2019**, *3*, 66. [CrossRef]
8. Lawler, E.L.; Lenstra, J.; Kan, A.R.; Shmoys, D.B. *The Traveling Salesman Problem*; John Wiley & Sons, Incorporated: Chichester, UK, 1985; Volume 12.
9. Gutin, G.; Punnen, A.P. *The Traveling Salesman Problem and Its Variations*; Springer Science & Business Media: Boston, MA, USA, 2006; Volume 12.
10. Agatz, N.; Bouman, P.; Schmidt, M. Optimization approaches for the traveling salesman problem with drone. *Transp. Sci.* **2018**, *52*, 965–981. [CrossRef]
11. Ha, Q.M.; Deville, Y.; Pham, Q.D.; Hà, M.H. On the min-cost traveling salesman problem with drone. *Transp. Res. Part C Emerg. Technol.* **2018**, *86*, 597–621. [CrossRef]
12. Murray, C.C.; Chu, A.G. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transp. Res. Part C Emerg. Technol.* **2015**, *54*, 86–109. [CrossRef]
13. Tang, Z.; van Hoeve, W.J.; Shaw, P. A study on the traveling salesman problem with a drone. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*; Springer: Cham, Switzerland, 2019; pp. 557–564.
14. Bouman, P.; Agatz, N.; Schmidt, M. Dynamic programming approaches for the traveling salesman problem with drone. *Networks* **2018**, *72*, 528–542. [CrossRef]
15. Schermer, D.; Moeini, M.; Wendt, O. A branch-and-cut approach and alternative formulations for the traveling salesman problem with drone. *Networks* **2020**, *76*, 164–186. [CrossRef]
16. Roberti, R.; Ruthmair, M. Exact methods for the traveling salesman problem with drone. *Transp. Sci.* **2021**, *55*, 315–335. [CrossRef]
17. Blum, C.; Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. CSUR* **2003**, *35*, 268–308. [CrossRef]
18. Stegherr, H.; Heider, M.; Hähner, J. Classifying Metaheuristics: Towards a unified multi-level classification system. *Nat. Comput.* **2020**, 1–17.
19. Applegate, D.; Bixby, R.; Chvátal, V.; Cook, W. *Finding Cuts in the TSP (A Preliminary Report)*; Technical Report 5; Rutgers University: New Brunswick, NJ, USA, 1995.
20. Applegate, D.; Bixby, R.; Chvátal, V.; Cook, W. The Traveling Salesman Problem. In *Combinatorial Optimization: Theory and Algorithms*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 527–562. [CrossRef]
21. Cook, W. Concorde TSP Solver. Available online: http://www.math.uwaterloo.ca/tsp/concorde/index.html (accessed on 29 October 2020).
22. Ponza, A. Optimization of Drone-Assisted Parcel Delivery. Master's Thesis, Universita Degli Studi Di Padova, Padova, Italy, 2015.
23. de Freitas, J.C.; Penna, P.H.V. A variable neighborhood search for flying sidekick traveling salesman problem. *Int. Trans. Oper. Res.* **2020**, *27*, 267–290. [CrossRef]
24. Crişan, G.C.; Nechita, E. On a cooperative truck-and-drone delivery system. *Procedia Comput. Sci.* **2019**, *159*, 38–47. [CrossRef]
25. González-R, P.L.; Canca, D.; Andrade-Pineda, J.L.; Calle, M.; Leon-Blanco, J.M. Truck-drone team logistics: A heuristic approach to multi-drop route planning. *Transp. Res. Part C Emerg. Technol.* **2020**, *114*, 657–680. [CrossRef]
26. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
27. Lipowski, A.; Lipowska, D. Roulette-wheel selection via stochastic acceptance. *Phys. A Stat. Mech. Appl.* **2012**, *391*, 2193–2196. [CrossRef]
28. Mladenović, N.; Hansen, P. Variable neighborhood search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100. [CrossRef]
29. de Freitas, J.C.; Penna, P.H.V. A randomized variable neighborhood descent heuristic to solve the flying sidekick traveling salesman problem. *Electron. Notes Discret. Math.* **2018**, *66*, 95–102. [CrossRef]
30. Özoğlu, B.; Çakmak, E.; Tuğçe, K. Clarke & Wright's savings algorithm and genetic algorithms based hybrid approach for flying sidekick traveling salesman problem. *Avrupa Bilim Teknol. Derg.* **2019**, 185–192.
31. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*; MIT Press: Cambridge, MA, USA, 1992.

32. Clarke, G.; Wright, J.W. Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.* **1964**, *12*, 568–581. [CrossRef]
33. Ha, Q.M.; Deville, Y.; Pham, Q.D.; Hà, M.H. A hybrid genetic algorithm for the traveling salesman problem with drone. *J. Heuristics* **2020**, *26*, 219–247. [CrossRef]
34. Ferrandez, S.M.; Harbison, T.; Weber, T.; Sturges, R.; Rich, R. Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *J. Ind. Eng. Manag. JIEM* **2016**, *9*, 374–388. [CrossRef]
35. Houseknecht, J. An ACO-Inspired, Probabilistic, Greedy Approach to the Drone Traveling Salesman Problem. Bachelor's Thesis, School of Engineering and Computational Sciences Senior Honors, Liberty University, Lynchburg, VA, USA, 2019.
36. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybernet. Part B Cybernet.* **1996**, *26*, 29–41. [CrossRef]
37. Feo, T.A.; Resende, M.G. Greedy randomized adaptive search procedures. *J. Glob. Optim.* **1995**, *6*, 109–133. [CrossRef]
38. Feo, T.A.; Resende, M.G. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.* **1989**, *8*, 67–71. [CrossRef]
39. Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computing*; Springer: Berlin/Heidelberg, Germany, 2003; Volume 53.
40. Serpell, M.; Smith, J.E. Self-adaptation of mutation operator and probability for permutation representations in genetic algorithms. *Evol. Comput.* **2010**, *18*, 491–514. [CrossRef] [PubMed]
41. Resende, M.G.; Ribeiro, C.C. Greedy randomized adaptive search procedures. In *Handbook of Metaheuristics*; Springer: Boston, MA, USA, 2003; pp. 219–249.
42. Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*; Prentice Hall: Upper Saddle River, NJ, USA, 2002.
43. Agatz, N.; Bouman, P.; Schmidt, M. Instances for the Tsp with Drone. Available online: https://zenodo.org/record/1204676#.YIc_y30zYTU (accessed on 30 October 2020).
44. Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 437–478.
45. Oyeka, I.C.A.; Ebuh, G.U. Modified Wilcoxon signed-rank test. *Open J. Stat.* **2012**, *2*, 172–176. [CrossRef]
46. Theodorsson-Norheim, E. Friedman and Quade tests: BASIC computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples. *Comput. Biol. Med.* **1987**, *17*, 85–99. [CrossRef]