

THE TRAVELING SALESMAN PROBLEM WITH DRONE

QUANG MINH HA

*Thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Applied Science in Engineering*

November 2019

Institute of Information and Communication Technologies,
Electronics and Applied Mathematics
Louvain School of Engineering
Université Catholique de Louvain
Louvain-la-Neuve
Belgium



Thesis Committee

Yves Deville (director)

Quang Dung Pham (director)

Pierre Schaus

Birger Raa

Christine Solnon

Charles Pecheur (president)

UCLouvain, Belgium

SOICT, HUST, Vietnam

UCLouvain, Belgium

Ghent University, Belgium

INSA Lyon, France

UCLouvain, Belgium

ABSTRACT

Over the past few years, unmanned aerial vehicles (UAV), also known as drones, have been adopted as part of a new logistic method in the commercial sector called "last-mile delivery". In this novel approach, they are deployed alongside trucks to deliver goods to customers to improve the quality of service and reduce the transportation cost. This approach gives rise to a new variant of the traveling salesman problem (TSP), called TSP with drone (TSP-D). A variant of this problem that aims to minimize the time at which truck and drone finish the service (or, in other words, to maximize the quality of service) was studied in the work of Murray and Chu (2015).

In this thesis, we propose two new variants of the TSP-D: the Min-Cost TSP-D and the Multi-Visits TSP-D. With each of this variant, a mathematical model – a MIP formulation – is presented. Furthermore, with the development of three metaheuristics (GRASP, HGA, ILS) based on local search, we can solve the problem in both min-time and min-cost objective efficiently under an acceptable time.

The computational results show that our proposed methods could outperform the ones in the previous work and could significantly improve the solution quality when comparing to the optimal TSP solution. Furthermore, the extensive analysis of the components of the heuristic as well as the parameters of the variants, we can understand more about the algorithm's sensitivity as well as the nature of the problem.

ACKNOWLEDGEMENTS

First of all, I would like to warmly thank Yves Deville and Quang Dung Pham for being my supervisors for the last 4 years. Yves, who is a very demanding supervisor, taught me lessons about always taking care of the tiniest details and make everything self-contained and easy to understand. His criticisms and straightforward comments during the research works helped me to improve the quality of the papers and this thesis. Dung, my second supervisor, gave me a lot of valuable and detailed comments and research questions. I also learn a lot from his programming skills to make implementations become efficient.

I'm also grateful to Minh Hoang Ha, with whom I wrote several papers. His experience in the field helped me to save much time in finding directions during my research.

I would like to give my sincere thanks to the member of the jury who accepted to read and give comments to this document.

My thanks are also for all the BeCool (and now AIA) members, especially Bui Quoc Trung, Khong Minh Thanh, Michael and Francois, for all the good / crazy time we've had together.

I thank all my Vietnamese friends in Louvain-la-Neuve for everything that we've shared together. I send my thanks to Bui Quoc Trung, Dao Minh Vu for helping me during my very first days in Belgium. Thanks brother Ho Trong Viet for creating a great community here in Louvain-la-Neuve. Many thanks to Bui Tuan Anh, Bui Duc Viet for sharing with me all the moments and experiences during the last 6 years.

I wish to thank my parents for their support in my life and for always encouraging me to pursuit the way of studying and doing research. Without them, I would not have such day as today.

Last but not least, I would like to thank my wife Bao for always being with me, supporting me in every circumstances. Without Bao, I would never have been able to start and finish this work. I'm also grateful to my daughters Mit, Chip and Ca who gave me the motivation in everything and who remind me of balancing between work and life.

CONTENTS

1	INTRODUCTION	3
1.1	The Traveling Salesman Problem with Drone	4
1.2	Contributions	5
1.3	Outline	6
1.4	Publications	7
2	BACKGROUND	9
2.1	The Traveling Salesman Problem	9
2.2	Greedy Randomized Adaptive Search Procedure	9
2.3	Genetic Algorithm	11
2.4	Iterated Local Search	13
3	THE TRAVELING SALESMAN PROBLEM WITH DRONE	17
3.1	Introduction	17
3.2	Definition and notations	18
3.3	Solution representation	20
3.4	Constraints	20
3.5	Objectives	21
3.6	Related works	22

Contributions

4	ON THE MIN-COST TRAVELING SALESMAN PROBLEM WITH DRONE	29
4.1	Mixed Integer Linear Programming Formulation	29
4.2	A Greedy Randomized Adaptive Search Procedure (GRASP) for TSP-D	34
4.3	TSP-LS heuristic	43
4.4	Experiment setup	47
4.5	Results	48
4.6	Conclusion	59
5	A HGA ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM WITH DRONE	61
5.1	A Hybrid Genetic Algorithm for TSP-D (HGA)	61
5.2	Computational Results	74
5.3	Conclusion	84
6	THE MULTI-VISITS TRAVELING SALESMAN PROBLEM WITH DRONE	85
6.1	Problem description	85
6.2	Mathematical model	86
6.3	A split algorithm for TSP-mVD	90
6.4	Iterated Local Search for TSP-mVD	92
6.5	Computational Results	99
6.6	Conclusion	104
7	CONCLUSION	107

7.1	Results	107
7.2	Future works	108

BIBLIOGRAPHY

LIST OF FIGURES

Figure 3.1	Optimal solution: TSP vs. min-cost TSP-D. TSP Objective = 1500, min-cost TSP-D Objective = 1420.53. The solid arcs are truck's path. The dash arcs are drone's path	19
Figure 4.1	Auxiliary graph for TSP tour in Figure 3.1	38
Figure 4.2	A truck relocation move operator : $relocate_T((TD, DD), 1, 2)$	42
Figure 4.3	A drone relocation move operator : $relocate_D((TD, DD), 1, 5, 4)$	42
Figure 4.4	A drone removal move operator : $remove_D((TD, DD), 2)$	42
Figure 4.5	A two-exchange move operator in which a drone node is exchanged with a truck node : $two_exchange((TD, DD), 3, 2)$	43
Figure 5.1	Transformations between giant-tour chromosome and TSP-D solution	64
Figure 5.2	DX Crossover for TSP-D.	67
Figure 5.3	Illustrations of $\mathcal{N}_1$ to $\mathcal{N}_6$	69
Figure 5.4	Illustrations of $\mathcal{N}_7$ to $\mathcal{N}_{12}$	70
Figure 5.5	Illustrations of $\mathcal{N}_{13}$ to $\mathcal{N}_{16}$	71
Figure 5.6	Reinsertion in restore method. Truck travels the solid lines and drone travels the dashed lines.	72
Figure 5.7	Crossovers' performance - Min-Time objective.	82
Figure 5.8	Crossovers' performance - Min-Cost objective.	82
Figure 6.1	Illustration of TSP-mVD solution	90

LIST OF TABLES

Table 4.1	Instances of min-cost TSP-D	47
Table 4.2	Performance of GRASP on TSP-tour construction heuristics in min-cost TSP-D	50
Table 4.3	Comparison with the min-cost TSP-D optimal solution.	51
Table 4.4	Performance of heuristics on 50-customer instances in the min-cost TSP-D	52
Table 4.5	Performance of heuristics on 100-customer instances in the min-cost TSP-D	53
Table 4.6	Performance of heuristics with different cost-ratio settings in min-cost TSP-D.	54
Table 4.7	Comparison between GRASP and the best solutions found by [MC15]	55
Table 4.8	Performance of heuristics of min-time TSP-D	57
Table 4.9	Trade-off between min-cost and min-time in terms of performance ratio	58
Table 4.10	The frequency of using drone in two objective functions	58
Table 5.1	Comparison of Instance Set 1 under Min-time objective	77
Table 5.2	Comparison with GRASP under Min-cost objective - Instance set 2	78
Table 5.3	Comparison with GRASP under Min-time objective - Instance set 2	79
Table 5.4	Comparison with HGVNS in [FP18] under min-time objective.	80
Table 5.5	Crossover performance comparison - Min-cost and Min-time objective	81
Table 5.6	Sensitivity analysis of key components	84
Table 6.1	Compare the solutions of MILP formulation and ILS on small instances $q = 2$	100
Table 6.2	Compare the solutions of MILP formulation and ILS on small instances $q = 1$	100
Table 6.3	Comparison with small instances from Murray and Chu's work.	101
Table 6.4	Performance of ILS on TSP-D on larger instances (50 and 100-customer).	102
Table 6.5	TSP-mVD vs. TSP-D solution quality under 50-customer instances.	103
Table 6.6	TSP-mVD vs. TSP-D solution quality under 100-customer instances.	104

LIST OF ALGORITHMS

1	Greedy Randomized Adaptive Search Procedure	10
2	Local search phase	11
3	Genetic Algorithm	12
4	Iterated Local Search	14
5	Greedy Randomized Adaptive Search Procedure (GRASP) for min-cost TSP-D	35
6	Split_Algorithm_Step1(s): Building the auxiliary graph and finding shortest path	36
7	Split_Algorithm_Step2(P,V,T): Extract_TSPD_Solution .	39
8	Split_Algorithm_Step1(s): Min-time adaptation	40
9	TSP-LS heuristic	44
10	calcSavings(j)	44
11	relocateAsTruck(j, subroute, savings)—Calculates the cost of relocating the customer j into a different position in the truck's route	45
12	relocateAsDrone(j, subroute, savings) - Calculates the cost of relocating customer j as a drone node	46
13	applyChanges function	47
14	HGA for TSP-D	62
15	Crossover DX for TSP-D	66
16	Cost calculation in TSP-mVD split	92
17	Iterated Local Search for TSP-mVD	94
18	InitialSolution	95
19	Perturbation technique	97
20	Restore method for TSP-mVD	98
21	AcceptanceCriterion	98

INTRODUCTION

Companies always tend to look for the most cost-efficient methods to distribute goods across logistic networks [Riz+07]. Traditionally, trucks have been used to handle these tasks and the corresponding transportation problem is modelled as a traveling salesman problem (TSP). However, a new distribution method has recently arisen in which small unmanned aerial vehicles (UAV), also known as drones, are deployed to support parcel delivery. On the one hand, there are four advantages of using a drone for delivery: (1) it can be operated without a human pilot, (2) it avoids the congestion of traditional road networks by flying over them, (3) it is faster than trucks, and (4) it has much lower transportation costs per kilometre [Woh14]. On the other hand, because the drones are powered by batteries, their flight distance and lifting power are limited, meaning they are restricted in both maximum travel distance and parcel size. In contrast, a truck has the advantage of long range travel capability. It can carry large and heavy cargo with a diversity of size, but it is also heavy, slow and has much higher transportation cost.

Consequently, the advantages of truck offset the disadvantages of drones and — similarly — the advantages of drones offset the disadvantages of trucks. These complementary capabilities are the foundation of a novel method named "last mile delivery with drone" [Ban13], in which the truck transports the drone close to the customer locations, allowing the drone to service customers while remaining within its flight range, effectively increasing the usability and making the schedule more flexible for both drones and trucks. Specifically, a truck departs from the depot carrying the drone and all the customer parcels. As the truck makes deliveries, the drone is launched from the truck to service a nearby customer with a parcel. While the drone is in service, the truck continues its route to further customer locations. The drone then returns to the truck at a location different from its launch point.

From the application perspective, a number of remarkable events have occurred since 2013, when Amazon CEO Jeff Bezos first announced Amazon's plans for drone delivery [New13], termed "a big surprise." Recently, Google has been awarded a patent that outlines its drone delivery method [Mur16]. In detail, rather than trying to land, the drone will fly above the target, slowly lowering packages on a tether. More interestingly, it will be able to communicate with humans using voice messages during the delivery process. Google initiated this important drone delivery project, called Wing, in 2014, and

it is expected to launch in 2017 [Gro16]. A similar Amazon project called Amazon Prime Air ambitiously plans to deliver packages by drone within 30 minutes [Pog16]. Other companies worldwide have also been testing delivery services using drones. In April 2016, Australia Post successfully tested drones for delivering small packages. That project is reportedly headed towards a full customer trial in late 2016 [Cut16]. In May 2016, a Japanese company—Rakuten—launched a service named "Sora Kaku" that "delivers golf equipment, snacks, beverages and other items to players at pickup points on the golf course" [New16]. In medical applications, Matternet, a California-based startup, has been testing drone deliveries of medical supplies and specimens (such as blood samples) in many countries since 2011. According to their CEO: it is "much more cost-, energy- and time-efficient to send [a blood sample] via drone, rather than send it in a two-ton car down the highway with a person inside to bring it to a different lab for testing," [Fre15]. Additionally, a Silicon Valley start-up named Zipline International began using drones to deliver medicine in Rwanda starting in July, 2016 [Too16].

From the academic perspective, the routing problem in which truck(s) and drone(s) are cooperated with each other is well studied in the last few years with hundreds of publications. Although the authors have given this problem many names, it is commonly known as the "Traveling Salesman Problem with Drone" with its generalized problem known as the "Vehicle Routing Problem with Drones".

1.1 THE TRAVELING SALESMAN PROBLEM WITH DRONE

The Traveling Salesman Problem with Drone (TSP-D) is basically concerned with finding a way to a set of customers using a truck and a drone in which each customer can only be visited once by either the truck or the drone. Both vehicles (truck and drone) starts at the depot. When in service, the drone could be launched directly from the depot or from the truck to customer location(s). After it has finished its task, the drone will return to the truck at a rendezvous point which is another customer location. When not in service, the drone is carried by the truck while its battery is recharged to prepare for the next launch. The truck, after launching the drone, could continue to visit other customer locations before joining with the drone at rendezvous location. Typically, a set of operational constraints must be fulfilled. It therefore limits the possible locations that the drone could visit. One important constraint is the drone's capacity. This capacity is measured differently based on the variant of the TSP-D and is a major aspect to investigate in the literature. Other constraint could limit the number of customer locations that a drone could visit before returning to the truck. The objective of the TSP-D is to either minimize the completion time of two vehicles or the their total operational cost.

We are well aware of many other variants of the TSP-D which may be different from the TSP-D defined above. For example, one variant could allow the truck to carry more than one drone. Other could allow the truck to stay at the same customer location, waiting for the drone to return etc.,.

In this thesis, we focus on the TSP-D with the hypothesis firstly proposed in the work of Murray and Chu [MC15] and then extend it to allow the drone to visit multiple customer locations before rejoining the truck.

1.2 CONTRIBUTIONS

The main contributions of this thesis are as follows.

ON THE MIN-COST TSP-D

- We introduce a new variant of TSP-D called min-cost TSP-D, in which the objective is to minimize the operational costs.
- We propose a model together with a MILP formulation for the problem which is an extended version of the model proposed in [MC15] for min-time TSP-D.
- We develop two heuristics for min-cost TSP-D: TSP-LS and GRASP. which contain a new split procedure and local search operators. We also adapt our solution methods to solve the min-time problem studied in [MC15].
- We introduce various sets of instances with different numbers of customers and a variety of options to test the problem.
- We conduct various experiments to test our heuristics on the min-cost as well as min-time problems. We also compare solutions of both objectives. The computational results show that GRASP outperforms TSP-LS in terms of quality of solutions with an acceptable running time. TSP-LS delivers solution of lower quality, but very quickly.

A HYBRID GENETIC SEARCH FOR THE TSP-D

- We propose an efficient hybrid genetic algorithm that includes a new crossover, a set of 16 local search operators, and a penalization and restore mechanism to solve the TSP-D under both min-cost and min-time objectives.
- We conduct extensive computational experiments to evaluate the performance of HGA under instance sets from [MC15] and [Ha+18b]. The proposed method outperforms existing approaches in terms of solution quality and can improve a number of best known solutions.

- We analyze the efficiency and importance of the new components to the performance of the overall algorithm.

THE MULTI-VISITS TSP-D

- We define a new variant of TSP-D with a mathematical model (a MILP formulation).
- We propose a new split method to transform a TSP tour to a TSP-mVD solution optimally (given the relative orders of the nodes are fixed).
- We introduce 4 new local search operators to specifically handle the "multi visits" property of the drone in TSP-mVD.
- We propose an Iterated Local Search (ILS) with new perturbation technique that could effectively solve the TSP-mVD.
- We conduct several experiments to test the result of our newly proposed elements as described above. The impact of the limit of drone-visits to the solution quality of TSP-mVD is also conducted to better understand the problem. The computational results show that our ILS delivers better results comparing to ones in the previous works in terms of solution quality under an acceptable time. Moreover, it also proves the effectiveness of allowing the drone to visit a certain number of customer locations before returning to the truck.

The choice of algorithms in this thesis is simply based on the fact that we want to test our approaches with different metaheuristics, not just a single one. Furthermore, for obtaining an optimal TSP-D or TSP-mVD solution, solving a MIP model is one possible approach. Other approaches also exist such as the use of dynamic programming. However, they are not in the scope of this thesis.

1.3 OUTLINE

The remainder of this thesis is organized as follows.

Chapter 2 briefly presents the Traveling Salesman Problem and the optimization techniques used in this thesis which are: Greedy Randomized Adaptive Search Procedure (GRASP), Genetic Algorithm (GA), Iterated Local Search (ILS).

Chapter 3 introduces the TSP-D where we define the problem and the notations used, the representation of the solution, the constraints and its objectives. Finally an extensive review of the related works in the field is presented.

Chapter 4 is about the first contribution of this thesis – the min-cost TSP-D – in which we propose a mathematical model for the problem, a split algorithm, two heuristics and many experiments.

Chapter 5 – this thesis second contribution – is the presentation of a Hybrid Genetic Search for TSP-D which could effectively solve the TSP-D under both objectives. The framework of the algorithm is introduced with a new crossover, a restore method and a number of local search operators. The computational results are also presented to prove the efficiency of this algorithm.

Chapter 6 proposes a new variant of the TSP-D called MVTSP-D where drone is allowed to visit multiple customer locations in one launch. The MVTSP-D is formally defined with a mathematical model. For the solution approach, a new split algorithm is introduced and is used in an Iterated Local Search (ILS) heuristic. We also propose a new perturbation method which can effectively solve the MVTSP-D with ILS. Computational results analyze the performance of ILS in comparison with the previous work and evaluate the effectiveness of the MVTSP-D over traditional TSP-D.

Finally, chapter 7 concludes the thesis and discuss the future research directions.

1.4 PUBLICATIONS

The content of this thesis leads to an accepted journal article, an article under review and a completed manuscript.

- Ha, Q. M., Deville, Y., Pham, Q. D., & Hà, M. H. (2018). On the min-cost traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies*, 86, 597-621.
- Ha, Q. M., Deville, Y., Pham, Q. D., & Hà, M. H. (2018). A Hybrid Genetic Algorithm for the Traveling Salesman Problem with Drone. *Journal of Heuristics*.
- The Multi-Visits Traveling Salesman Problem with Drone. Completed manuscript.

BACKGROUND

2.1 THE TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) is one of the most widely studied combinatorial optimization problems. With a simple definition, it still remains one of the most challenging problems in Operational Research [Lap92].

In detail, let $G = (V, A)$ be a graph where V is the set of n vertices and A is the set of arcs connecting the vertices in V . We denote c_{ij} the travel cost between two vertices i and j . The TSP consists of determining a tour passing through each vertices once and only once with a minimum travel cost.

In terms of mathematical model, the TSP can be defined under the form of an integer programming formulation, as firstly proposed by [MTZ60]:

$$\text{Min } \sum_{i \in A} \sum_{\substack{j \in A \\ i \neq j}} c_{ij} x_{ij} \quad (2.1)$$

$$\sum_{i=0, i \neq j}^n x_{ij} = 1 \quad \forall j = 1, \dots, n \quad (2.2)$$

$$\sum_{j=0, j \neq i}^n x_{ij} = 1 \quad \forall i = 1, \dots, n \quad (2.3)$$

$$u_i - u_j + nx_{ij} \leq n - 1 \quad \forall 1 \leq i \neq j \leq n \quad (2.4)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in A \quad (2.5)$$

where x_{ij} is a binary variable determining whether the arc ij is included in the final tour, u_i are arbitrary real numbers that is used in a subtour elimination constraint.

2.2 GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

In this section, we discuss the Greedy Randomized Adaptive Search Procedure [FR95], a multistart heuristic that is commonly used in many optimisation problems [GP10]. In theory, by enumerating all combinations of solutions and evaluate them under an objective function, one can finally conclude an optimal outcome. However, it is

impossible in practice to conduct this type of enumeration since the search space would be too large. Therefore, instead of exploring the whole search space, we could, for each iteration, start to explore a specific point of this space with the hope to find good (or optimal) solution. This is the idea behind GRASP.

In detail, there are two steps for each iteration of GRASP: (1) construction and (2) local search. The construction phase builds up a solution at first. This solution is guaranteed to be feasible by using a repair mechanism in case it is not initially feasible. In the second step, the constructed feasible solution is then used as the input in a local search procedure. The local search will try to improve this solution until it reaches a local minima. In the end, the best overall solution is kept and returned. Algorithm 1 presents the pseudo-code of GRASP.

Algorithm 1: Greedy Randomized Adaptive Search Procedure

Data: $maxIteration, randomSeed$
Result: $bestSolution$

```

1  $solution = null$  ;
2  $bestSolution = null$  ;
3 for  $i = 1, \dots, maxIteration$  do
4    $solution = constructionPhase(randomSeed)$ ;
5   if  $solution$  is not feasible then
6      $solution = repair(solution)$  ;
7    $solution = localsearch(solution)$  ;
8   if  $f(bestSolution) < f(solution)$  then
9      $bestSolution = solution$  ;
10 return  $bestSolution$  ;
```

In Algorithm 1, there are two data inputs which are the number of iterations that GRASP will be running, and a random seed to provide the randomness of the construction phase. The algorithm starts by defining the current solution and the best solution (Lines 1-2). It then starts the loop for the iterations in Line 3. For each iteration, a solution is first created by using a *constructionPhase* heuristic (Line 4). This solution is checked for feasibility and is repair in case of being infeasible (Lines 5-6). In the next step, the algorithm puts the solution to a local search method (*localsearch()*) with the hope that it could be improved. We can also call this step the "education" step. Once the solution is "educated", it will be compared with the current *bestSolution* to see if a new best solution is found. If this is the case, we update the *bestSolution* with the current *solution* (Lines 7-9). In the end, the *bestSolution* is returned.

The construction phase in Algorithm 1 uses a greedy strategy in which at each iteration an element is chosen to add into a partial solution in the way that the increment of solution cost is minimum. In

routing problems such as the Traveling Salesman Problem or the Vehicle Routing Problem, the construction phase is usually done by using heuristics like k-nearest neighbors, k-cheapest insertion [RSL77]. These heuristics start with a partial solution containing the depot. Then, for each iteration, it chooses a random node among k of its best neighbors following the its criteria (nearest nodes or nodes with cheapest insertion cost) to add to the partial solution.

The local search phase usually improves the constructed solution. It works iteratively by successively replacing current solution with better solution until a local minima is reached. The pseudo-code of such local search is described in Algorithm 2.

Algorithm 2: Local search phase

Data: *solution*

Result: *solution*

```

1 while solution is not locally optimal do
2    $N(\text{solution}) = \text{neighbors of solution ;}$ 
3   if  $\exists s' \in N(\text{solution}) : f(s') < f(\text{solution})$  then
4      $\text{soluton} = s' ;$ 
5 return solution;

```

The speed the effectiveness of the local search procedure depend mostly on the neighborhood structure, the constraint verification step, the objective function and the initial solution. That's why the construction phase also plays an important role in GRASP as it could provide a good starting point for the local search. The neighborhood search (Lines 2-4 of Algorithm 2) in a local search could be implemented in either *best improvement* or *first improvement* manner. In detail, a *best improvement* strategy search for all neighbors of the current solution that could be an improvement and select the best one. On the other hand, *first improvement* strategy moves immediately to the first neighbor solution that could improve it. In practice, both strategies seem to deliver similar solution quality, but first improvement strategy often runs in shorter time. Furthermore, the search is often trapped in bad local minima with best improvement strategy.

In the literature, GRASP has been used to solve both problems of TSP and VRP such as in these research works [MMP05b], [HPRMSG09], [MMP05a], [KB95], [Pri09a], [Duh+10], [Mar12].

2.3 GENETIC ALGORITHM

This section discusses the Genetic Algorithm (GA), a metaheuristic firstly introduced by Holland [Hol+92]. Its concept is mainly based on the use of mutation and selection which is the core of the neo-Darwinian theory of evolution.

In GA, we manage a population of strings (or *chromosomes*) as representation of different solutions. The algorithm then goes through many iterations, each of which consists of three steps: parents selection, crossover and mutation. The pseudo-code of this metaheuristic is described in Algorithm 3.

Algorithm 3: Genetic Algorithm

```

1 population = initialisePopulation() ;
2 while termination criterion are not satisfied do
3   select two parent chromosomes ;
4   offspring = perform crossover on two parents ;
5   if mutation condition has met then
6     choose a mutation point ;
7     perform mutation ;
8   evaluate fitness of offspring ;
9   population = population + offspring ;
10  if number of chromosomes in the population has reached a certain
    limit then
11    select a new population by eliminating "bad"
    chromosomes with respect to an evaluation function ;

```

In detail, the algorithm first initialises a population of chromosomes by calling the *initializePopulation()* (Line 1) procedure which could be implemented by using multiple strategies such as random string generations or constructive oriented approaches (i.e., nearest neighbors in TSP). After the population has been created, we start to go through a numerous iterations, in which we conduct the following steps:

- Parents selection (Line 3): two parent chromosomes are selected from the population by using a selection function. There exists also different strategies for selecting these two such as random, roulette-wheel, ranking and tournament selection [Pot96].
- Crossover (Line 4): after two parents have been selected, they are used as inputs in a process named crossover, where some "genes" (subsequence of a chromosome) of a parent will be replaced by ones from the other parent. For example: P1 = a1, a2, a3, a4, a5 and P2 = b1, b2, b3, b4, b5 A one point OX crossover would produce two offsprings like: O1 = a1, a2, b3, b4, b5 and O2 = b1, b2, a3, a4, a5. In general, a crossover strategy should be independent from the specification of the problem. However, there exists some researches that propose problem-dependent crossovers. This class of crossovers is called heuristic crossover.
- Mutation (Lines 5-7): After the offsprings are created from crossover step, they will have a certain probability of going

through a mutation process where some of their genes' positions are exchanged randomly. This step is often considered as a diversification step as it helps to diversify the population's chromosome and therefore avoid the search from being trapped in a local minima.

- Fitness evaluation (Line 8): At this step, the offspring is given a "score" called fitness where it evaluates the importance of this offspring with respect to the current chromosomes in the population. Different criterion could be used to evaluate this score such as the difference between this offspring and all other chromosomes in the population, the objective value of this offspring compare to the values in the population... The offsprings are then added into the population (Line 9).
- Selecting survivors for new population (Lines 10-11): At the end of every iteration, the population size is checked to decide whether the algorithm should continue with this population to the next iteration, or to eliminate some elements. This size limit is a parameter of the GA and should be tuned based on the specific problem. In practice, the population is normally sorted by the fitness function and the most *size – limit* worst chromosomes will be removed.

Genetic Algorithm has proven to effectively solve many variants of the Traveling Salesman Problem as well as the Vehicle Routing Problem such as in [Gre+85], [OSH87], [Lar+99], [Uld+90], [Moo+02], [FM96], [BA03], [ORHo6], [Ho+08] despite the debates on why does it work.

2.4 ITERATED LOCAL SEARCH

Iterated Local Search (ILS) is a metaheuristic that is based on the key idea that instead of searching the total space of solutions, the algorithm only focuses the search on a smaller subset by iteratively building a chain of solutions [GP10]. In detail, starting from an initial solution, ILS will apply a local search heuristic in the hope of improving this solution until it reaches a local minima. This local optimal solution is then gone through a process call "shaking" where it will be perturbed to get a new solution for the next iteration. The process continues until a certain stopping criterion has been met. By using this approach, we can avoid a total random restart in each iteration, leading to a total random search.

The general framework of ILS is described in Algorithm 4. Again, we first create the starting solution by calling the method *GenerateInitialSolution* (Line 1). There are multiple strategies for generating this solution, as we will discuss shortly below. This initial so-

lution will be improved by using a local search method (Line 2). The result solution is then used in the iterations of ILS (Lines 3 to 7).

Algorithm 4: Iterated Local Search

```

1  $s_0 = \text{GenerateInitialSolution}$  ;
2  $s^* = \text{LocalSearch}(s_0)$  ;
3 repeat
4    $s' = \text{Perturbation}(s^*, \text{history})$  ;
5    $s^{*'} = \text{LocalSearch}(s')$  ;
6    $s^* = \text{AcceptanceCriterion}(s^*, s^{*'}, \text{history})$  ;
7 until termination criterion met;
  
```

For each iteration (Lines 3-7), we first create a new solution s' by shaking the previous solution s^* in *Perturbation* method (Line 4). This perturbation strategy also takes into account a history of solutions in order to balance between diversification and intensification as the perturbed solution could be the one that is already considered (belongs to the history list). If this is the case, the perturbation method could choose, by using an evaluation function, whether to use this "visited" solution or restart the shaking process to obtain a new one.

The chosen, perturbed solution is then passed through a local search method with the hope to improve its quality (Line 5). Finally, we choose a solution to be used in the next iteration by calling the *AcceptanceCriterion* procedure where both previous solution s^* , newly improved solution $s^{*'}$ and solutions from the history are taken into consideration. Again, there exists multiple strategy for choosing the next solution. These strategies are often tuned by experiment, but we could sum up some of them as follows:

- Improvement only strategy: we only accept the new solution $s^{*'}$ if it is better than the previous solution s^* . If this is not the case, we will continue with s^* or one of the history solution in the next iteration.
- Simulated Annealing strategy: if $s^{*'}$ is better than s^* , we accept $s^{*'}$. Otherwise, $s^{*'}$ is accepted with a probability $p = e^{-\frac{f(s^{*'}) - f(s^*)}{T}}$ where T is a temperature that is set to a high value at the beginning and then goes smaller as the algorithm progresses.

The algorithm stops at Line 7 where stopping criterion has been met such as the running time limit, the number of iterations without any improvement, the number of total iterations etc., Finally, the best found solution during all iterations is returned.

One of the most important step in an ILS is the *Perturbation* method where a strategy is applied to modify the previous solution to obtain a new solution (new starting point). There are two possible bad cases of this perturbation step:

- The solution is not "shaken" good enough so the new obtained solution is not be able to exit the current local optimal and move to another region. In this case, it has too much intensification and too little diversification in terms of perturbation.
- The solution is shaken too much that it will jump to a total different solution. ILS then becomes a total random start algorithm where there is too much diversification.

The design of this perturbation varies between problems. For the Traveling Salesman Problem, there are multiple strategies for this method such as Multi-Swaps, Multi-Relocate, or some problem-specific moves. However, the most common method for the TSP is the Double Bridge method where four arcs are removed and then reconnected to form a new solution.

ILS has been also used to solve many variants of the TSP and VRP effectively as in [VT99], [KN99], [PSO13], [Iba+08].

THE TRAVELING SALESMAN PROBLEM WITH DRONE

3.1 INTRODUCTION

In this section, we provide a description of the problem and discuss a model for the TSP-D in a step-by-step manner. Here, we consider a list of customers to whom a truck and a drone will deliver parcels. To make a delivery, the drone is launched from the truck and later rejoins the truck at another location. Each customer is visited only once and is serviced by either the truck or the drone. Both vehicles must start from and return to the depot. When a customer is serviced by the truck, this is called a **truck delivery**, while when a customer is serviced by the drone, this is called a **drone delivery**. This is represented as a 3-tuple $\langle i, j, k \rangle$, where i is a launch node, j is a drone node (a customer who will be serviced by the drone), and k is a rendezvous node, as listed below:

- Node i is a launch node at which the truck launches the drone. The launching operation must be carried out at a customer location or the depot. The time required to launch the drone is denoted as s_L .
- Node j is a node serviced by the drone, called a "drone node". We also note that not every node in the graph is a drone node. Because some customers might demand delivery a product with size and weight larger than the capacity of the drone.
- Node k is a customer location where the drone rejoins the truck. At node k , the two vehicles meet again; therefore, we call it "rendezvous node". While waiting for the drone to return from a delivery, the truck can make other truck deliveries. The time required to retrieve the drone and prepare for the next drone delivery is denoted as s_R . Moreover, the two vehicles can wait for each other at the rendezvous point.

Moreover, the drone has an "endurance", which can be measured as the maximum time the drone can operate without recharging. A tuple $\langle i, j, k \rangle$ is called feasible if the drone has sufficient power to launch from i , deliver to j and rejoin the truck at k . The drone can be launched from the depot but must subsequently rejoin the truck at a customer location. Finally, the drone's last rendezvous with the truck can occur at the depot.

When not actively involved in a **drone delivery**, the drone is carried by the truck. Again, as in [MC15], we also assume that the drone

is in constant flight when waiting for the truck. Furthermore, the truck and the drone have their own transportation costs per unit of distance. In practice, the drone's cost is much lower than the truck's cost because it weighs much less than the truck, hence, consuming much less energy. In addition, it is not run by gasoline but by batteries. We also assume that the truck provides new fresh batteries for the drone (or recharges its batteries completely) before each drone delivery begins. When a vehicle has to wait for each other, a penalty is created and added to the transportation cost to form the total operational cost of the system. The waiting costs of truck and drone are calculated by:

$$\text{waiting cost}_{truck} = \alpha \times \text{waiting time}$$

$$\text{waiting cost}_{drone} = \beta \times \text{waiting time}$$

where α and β are the waiting fees of truck and drone per unit of time, respectively. The use of these coefficients is flexible to model a number of situations in reality. Typically, α can be used to represent parking fee and labour cost of the truck driver, while β can model costs resulted by battery energy consumption of drone. In some contexts where a delivery company does not have to pay waiting costs and wants to focus only on transportation cost, it can set both coefficients to null. In addition, if we do not allow the waiting of drone to protect it from theft or being shot down by strangers, we can set β to a very large number. Similarly for α , if we only have a very short time to park the truck at the customer locations.

The objectives of the TSP-D are either to minimize the total operational cost of the system (which includes the travel cost of truck and drone as well as their waiting costs) or to minimize the completion time of two vehicles. We denote the problem with the first objective as “**min-cost TSP-D**” and with the latter as “**min-time TSP-D**”.

Because the problem reduces to a TSP when the drone's endurance is null, it is NP-Hard. Examples of TSP and min-cost TSP-D optimal solutions on the same instance in which the unitary transportation cost of the truck is 25 times more expensive than that of the drone are shown in Figure 3.1.

We now develop the abstract model for the TSP-D. We first define basic notations relating to the graph, sequence and subsequence. Then, we formally define drone delivery and the solution representation as well as the associated constraints and objectives.

3.2 DEFINITION AND NOTATIONS

The TSP-D is defined on a graph $G = (V, A)$, $V = \{0, 1, \dots, n, n+1\}$, where 0 and $n+1$ both represent the same depot but are duplicated

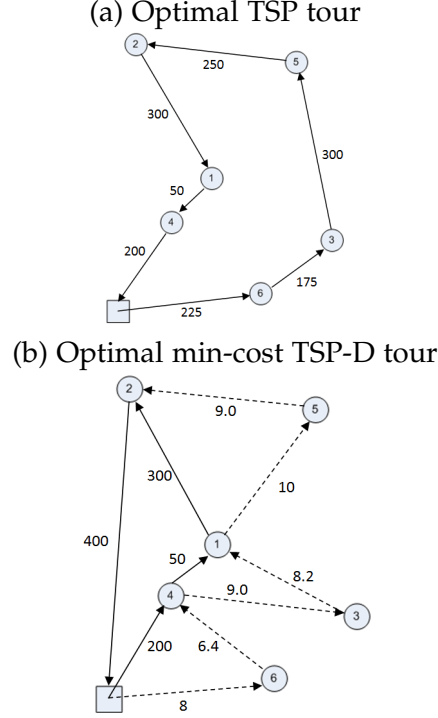


Figure 3.1: Optimal solution: TSP vs. min-cost TSP-D.

TSP Objective = 1500, min-cost TSP-D Objective = 1420.53. The solid arcs are truck's path. The dash arcs are drone's path

to represent the starting and returning points. The set of customers is $N = \{1, \dots, n\}$. Let $V_D \subseteq N$ denote the set of customers that can be served by drone. Let d_{ij} and d'_{ij} be the distances from node i to node j travelled by the truck and the drone, respectively. We also denote τ_{ij} , τ'_{ij} the travel time of truck and drone from i to j . Furthermore, C_1 and C_2 are the transportation costs of the truck and drone, respectively, per unit of distance.

Given a sequence $s = \langle s_1, s_2, \dots, s_t \rangle$, where $s_i \in V, i = 1 \dots t$, we denote the following:

- $V(s) \subseteq V$ the list of nodes of s
- $pos(i, s)$ the position of node $i \in V$ in s
- $next_s(i), prev_s(i)$ the next node and previous node of i in s
- $first(s), last(s)$ the first node and last node of s
- $s[i]$ the i^{th} node in s
- $size(s)$ the number of nodes in s
- $sub(i, j, s)$, where $\forall i, j \in s, pos(i, s) < pos(j, s)$, the subsequence of s from node i to node j
- $A(s) = \{(i, next_s(i)) | i \in V(s) \setminus last(s)\}$ the set of arcs in s

- $d_{i \rightarrow k}$ the distance traveled by truck from i to k in the truck tour
- $t_{i \rightarrow k}$ the time traveled by truck from i to k in the truck tour
- t'_{ijk} the time traveled by drone from i to j to k in a drone delivery
- $d_{i \rightarrow k}^{-j}$ the distance traveled by truck from i to k in the truck tour with j removed (j is between i and k)
- r_i is the arrival time of truck at node i
- r'_i is the arrival time of drone at node i
- $recover_{truck}$ is the time taken for the truck to recover the drone and possibly prepare it for the next launch. More specifically, if the truck just recovers the drone without relaunching it at the same location, then $recover_{truck} = s_R$. Otherwise, if the truck relaunches the drone at the same location, then $recover_{truck} = s_R + s_L$.
- $recover_{drone} = s_R$ is the time taken to recover the drone at the rendezvous point.

As mentioned above, we define a **drone delivery** as a 3-tuple $\langle i, j, k \rangle : i, j, k \in V, i \neq j, j \neq k, k \neq i, \tau'_{ij} + \tau'_{jk} \leq \epsilon$, where ϵ is a constant denoting the drone's endurance. We also denote $\mathbb{P}$ as the set of all possible drone deliveries on the graph $G = (V, A)$ that satisfies the endurance constraint as follows:

$$\mathbb{P} = \{ \langle i, j, k \rangle : i, k \in V, j \in V_D, i \neq j, j \neq k, k \neq i, \tau'_{ij} + \tau'_{jk} \leq \epsilon \}.$$

3.3 SOLUTION REPRESENTATION

A **TSP-D solution**, denoted as sol , is represented by two components:

- A truck tour, denoted as TD , is a sequence $\langle e_0, e_1, \dots, e_k \rangle$, where $e_0 = e_k = 0, e_i \in V, e_i \neq e_j$ and $i \neq j$.
- A set of drone deliveries DD such that $DD \subseteq \mathbb{P}$,

which can also be written as

$$sol = (TD, DD).$$

3.4 CONSTRAINTS

A solution (TD, DD) of the TSP-D must satisfy the following constraints:

(A) Each customer must be serviced by either the truck or the drone:

$$\forall e \in N : e \in TD \text{ or } \exists \langle i, e, k \rangle \in DD.$$

By definition, during a truck tour, a customer cannot be visited twice by the truck. The above constraint does not prevent a customer from being serviced by both the truck and the drone nor from being serviced twice by the drone.

(B) A customer is never serviced twice by the drone:

$$\forall \langle i, j, k \rangle, \langle i', j', k' \rangle \in DD : j \neq j'.$$

(C) Drone deliveries must be compatible with the truck tour:

$$\forall \langle i, j, k \rangle \in DD : j \notin TD, i \in TD, k \in TD, pos(i, TD) < pos(k, TD).$$

This constraint implies that a customer cannot be serviced by both the truck and drone.

(D) No interference between drone deliveries:

$$\forall \langle i, \cdot, k \rangle \in DD, \forall e \in sub(i, k, TD), \forall \langle i', j', k' \rangle \in DD : e \neq i'.$$

The above constraint means that when the drone is launched from the truck for a drone delivery, it cannot be relaunched before the rendezvous from that delivery. As a consequence, we cannot have any other rendezvous during that period either.

3.5 OBJECTIVES

There exists two objectives of the TSP-D: one is to minimize the completion time of two vehicles (when they return to the depot) and the other is to minimize the operational cost of two vehicles. We call the former the "**Min-cost TSP-D**" and the latter the "**Min-time TSP-D**".

Regarding the costs, the following notations are used:

- $cost(i, j, k) = C_2(d'_{ij} + d'_{jk})$, where $\langle i, j, k \rangle \in \mathbb{P}$ [cost of drone delivery $\langle i, j, k \rangle$]
- $cost_W^T(i, j, k) = \alpha \times \max(0, (t_{i \rightarrow k} - t'_{ijk}))$, where $\langle i, j, k \rangle \in \mathbb{P}$ [waiting cost of truck at k]
- $cost_W^D(i, j, k) = \beta \times \max(0, (t'_{ijk} - t_{i \rightarrow k}))$, where $\langle i, j, k \rangle \in \mathbb{P}$ [waiting cost of drone at k]
- $cost(TD) = \sum_{(i,j) \in A(TD)} C_1.d_{ij}$ [cost of a truck tour TD]
- $cost(DD) = \sum_{\langle i,j,k \rangle \in DD} cost(i, j, k)$ [total cost of all drone deliveries in DD]

- $cost_W(DD) = \sum_{\langle i,j,k \rangle \in DD} cost_W^T(i,j,k) + cost_W^D(i,j,k)$ [total waiting cost]
- $cost(TD, DD) = cost(TD) + cost(DD) + cost_W(DD)$ [cost of a solution]
- $cost(sub(i,k,s))$ the total cost for both truck and drone and their waiting cost (if any) in a subsequence $s \in TD$.

Regarding the times, the following extra notation is used:

- $time(TD, DD)$ or $time(sol = (TD, DD))$ = the time that two vehicles returns to the depot.

The min-cost objective is to minimize the total operational cost:

$$\min cost(TD, DD).$$

The min-time objective is to minimize the time return to depot of two vehicles:

$$\min time(TD, DD).$$

3.6 RELATED WORKS

In the literature, there are several publications that have investigated the routing problem related to the truck-drone combination for delivery. The past few years witnessed the vast development in this research field with hundreds of new publications. Many variants of this truck-drone routing problem has been proposed. In this section, we will first focus on reviewing the latest works that are directly related to the variant researched in this thesis and then briefly review the latest development in other variants.

As the very first work on TSP-D, [MC15] introduced the problem, calling it the "Flying Sidekick Traveling Salesman Problem" (FSTSP). A mixed integer liner programming (MILP) formulation and a heuristic are proposed. Basically, their heuristic is based on a "Truck First, Drone Second" idea, in which they first construct a route for the truck by solving a TSP problem and, then, repeatedly run a relocation procedure to reduce the objective value. In detail, the relocation procedure iteratively checks each node from the TSP tour and tries to consider whether it is suitable for use as a drone node. The change is applied immediately when this is true, and the current node is never checked again. Otherwise, the node is relocated to other positions in an effort to improving the objective value. The relocation procedure for TSP-D is designed in a "best improvement" fashion; it evaluates all the possible moves and executes the best one. The proposed methods are tested only on small-sized instances with up to 10 customers.

[ABS16], study a slightly different problem—called the "Traveling Salesman Problem with Drone" (TSP-D), in which the drone has to follow the same road network as the truck. Moreover, in TSP-D, the drone may be launched and return to the same location, while this is forbidden in the FSTSP. This problem is also modelled as a MILP formulation and solved by a "Truck First, Drone Second" heuristic in which drone route construction is based on either local search or dynamic programming. Recently, [BAS17] extended this work by proposing an exact approach based on dynamic programming that is able to solve larger instances. Furthermore, [Pon16] also extended the work of Murray and Chu [MC15] in his master's thesis to solve the FSTSP, proposing an enhancement to the MILP model and solving the problem by a heuristic method based on Simulated Annealing.

Shortly after, [CS17] delivers a research in which the authors also evaluate the efficiency of the TSP-D by providing numerous theoretical analysis as well as the computational results. The authors also named the problem the "horsefly routing problem" with the objective to minimize the completion time of the vehicles, which is similar to the variant proposed by [MC15] except that the number of drone is not strictly limited to one, but is generalized and become a parameter of the problem. Additionally, the study of the upper and lower bounds for the completion time of the TSP-D is also conducted. For the solution approach, a matheuristic is proposed in which it iteratively builds the TSP-D solution starting from the optimal TSP solution. From this work, the author concludes that the improvement in completion time due to the use of drone is proportional to the square root of the ratio of the speeds of the truck and the drone.

In a recent work, Freitas et al. [FP18] proposed a hybrid heuristic named HGVNS to solve two TSP-D variants by [MC15] and [ABS18] with the min-time objective. In detail, HGVNS first obtains the initial solution by using an MIP solver to solve the TSP optimally and then applies a heuristic in which some trucks' customers are removed and reinserted as drone customers. Next, the initial solution is used as the input for a general variable neighbourhood search in which eight neighbourhoods are shuffled and chosen randomly. The authors conducted the experiments on three instance sets from [Pon16], [ABS18] and TSPLIB. The computational results show that the proposed approach can decrease delivery time by up to 67.79%.

Apart from the works that investigate the original TSP-D variant where one truck and one drone are used for delivery, there exists many other variants of the TSP-D as well as its generalization – the Vehicle Routing Problem with Drone (VRP-D) – that have been studied in many other researches. We will now briefly discuss their details.

Poikonen [Poi18], in another work, is the first work that introduces a new variant of the TSP-D called the "Multi-visit Drone Routing Problem" which is also described in Chapter 6 of this thesis. In his

work, instead of being limited to carry one parcels at a time, the drone is now be able to carry and visit multiple customer locations before returning to the truck for recharging/taking new parcels. Additionally, the author also considers that the drone has a fixed energy capacity instead of a fixed flight time to cope with the variety of parcels' weight. For defining the problem, the author proposes a descriptive model and various theoretical analysis. For solving the problem more effectively, a local search method is introduced using a Swap-based operator. Extensive computational results have been conducted to analyse the performance and sensitivity of the factors (i.e. drone speed and battery). Finally, the author also extends this model to cope with more than one drone, which leads to a more general problem.

Liu et al. [Liu+19], in the recent work, proposes a similar variant of the TSP-D where a drone is able to service multiple customers before returning to the truck. In this work, the author gives the definition of the problem, proposes a two-stage, route-based modeling approach to optimize both truck's main route and the drone's flying schedule. To construct the initial solution, a hybrid heuristic combining a nearest-neighbor and cost-saving strategies is used. To improve this initial solution, the author applies the Simulated Annealing (SA) metaheuristic. Moreover, an experiment is conducted using randomly generated customer locations as well as a practical road network of in Changsha, China. The computational results analyse the overall performance of the SA as well as the sensitivity of different factors such as the ratio of light parcels, the drone's capacity, drone endurance.

Recently, Murray et al. [MR19] extends his original work – the FSTSP – to introduce a new problem called "The Multiple Flying Sidekick Traveling Salesman Problem" (mFSTSP) where multiple drones are used with a truck to make the deliveries. Again, the authors define the problem with a mixed integer programming formulation and propose a three-phase heuristic to solve the problem with various size upto 100 customers. The computational results show that this variant can improve the truck-only delivery (the TSP) problem by up to 23%.

The extension of TSP-D to the mTSP-D is introduced in the work of [Kit+19] where a fleet of trucks and drones are used for delivery. However, instead of fixing the pair of truck and drone (i.e. a truck only carries and works with one identical drone, not others), the problem allows the drones to return to any available trucks. The authors present a mixed integer programming model for the mTSP-D and develop a heuristic called "Adaptive Insertion Heuristic" (ADI) to solve the problem. The computational results show that there is a potential gain for the mTSP-D comparing to the traditional TSP-D.

Wang et al. [WPG16] introduced a more general problem that copes with multiple trucks and drones with the goal of minimizing the completion time. The authors named the problem "The vehicle routing

problem with drone" (VRP-D) and conducted the analysis on several worst-case scenarios, from which they propose bounds on the best possible savings in time when using drones and trucks instead of trucks alone. An further development of this research is studied in [PWG17] where the authors extend the worst-case bounds to more generic distance/cost metrics as well as explicitly consider the limitation of battery life and cost objectives.

Most recently, Wang et al. [WS19] introduces a further work of the VRP-D studied from 2016 by proposing a mixed integer programming model for the problem (an arc-based model) and develop a branch-and-price algorithm. The experiment was run and conducted under 10 and 15-customer instances. The results show that by using MIP solver with the proposed branch-and-price algorithm, the authors can deliver the optimal solutions of these instances in the average of 7 minutes and 4 hours for two types of instances above. Furthermore, it is reported that a VRP-D solution could save up to 20% of the cost comparing to the traditional VRP.

CONTRIBUTIONS

ON THE MIN-COST TRAVELING SALESMAN PROBLEM WITH DRONE

4.1 MIXED INTEGER LINEAR PROGRAMMING FORMULATION

The min-cost TSP-D defined in the previous section is represented here in a MILP formulation. This formulation is an extension from the one proposed by [MC15]. We extend it by proposing constraints where waiting time is captured in order to calculate the waiting cost of two vehicles. We first define two subsets of V , $V_L = \{0, 1, \dots, n\}$ and $V_R = \{1, 2, \dots, n+1\}$ to distinguish the nodes that from where the drone can be launched from and the one it returns to.

Variables

Let $x_{ij} \in \{0, 1\}$ equal one if the truck goes from node i to node j with $i \in V_L$ and $j \in V_R$, $i \neq j$. Let $y_{ijk} \in \{0, 1\}$ equal one if $\langle i, j, k \rangle$ is a **drone delivery**. We can denote $p_{ij} \in \{0, 1\}$ as equalling one if node $i \in N$ is visited before node $j \in N$, $j \neq i$, in the truck's path. We also set $p_{0j} = 1$ for all $j \in N$ to indicate that the truck always starts the tour from the depot. As in standard TSP subtour elimination constraints, we denote $0 \leq u_i \leq n+1$ as the position of the node i , $i \in V$ in the truck's path.

To handle the waiting time, let $t_i \geq 0, t'_i \geq 0, i \in V_R$ denote the arrival time of truck and drone at node i , $r_i \geq 0, r'_i \geq 0, i \in V_R$ the leaving time of truck and drone at node i . We also denote $w_i \geq 0, w'_i \geq 0, i \in V_R$ the waiting time of truck and drone at node i respectively. Finally, we have $t_0 = 0, t'_0 = 0, r_0 = 0, r'_0 = 0$ the earliest time of truck and drone starting from depot 0 and $w_0 = 0, w'_0 = 0$ the waiting time at the starting depot.

The MILP formulation is as follows:

$$\begin{aligned} \text{Min } C_1 \sum_{i \in V_L} \sum_{\substack{j \in V_R \\ i \neq j}} d_{ij} x_{ij} + C_2 \sum_{i \in V_L} \sum_{\substack{j \in N \\ i \neq j}} \sum_{\substack{k \in V_R \\ \langle i, j, k \rangle \in \mathbb{P}}} (d'_{ij} + d'_{jk}) y_{ijk} \\ + \alpha \sum_{i \in V} w_i + \beta \sum_{i \in V} w'_i \end{aligned} \quad (4.1)$$

$$\sum_{\substack{i \in V_L \\ i \neq j}} x_{ij} + \sum_{\substack{i \in V_L \\ i \neq j}} \sum_{\substack{k \in V_R \\ \langle i, j, k \rangle \in \mathbb{P}}} y_{ijk} = 1 \quad \forall j \in N \quad (4.2)$$

$$\sum_{j \in V_R} x_{0j} = 1 \quad (4.3)$$

$$\sum_{i \in V_L} x_{i,n+1} = 1 \quad (4.4)$$

$$u_i - u_j + 1 \leq (n+2)(1 - x_{ij}) \quad \forall i \in V_L, j \in \{V_R : i \neq j\} \quad (4.5)$$

$$\sum_{\substack{i \in V_L \\ i \neq j}} x_{ij} = \sum_{\substack{k \in V_R \\ k \neq j}} x_{jk} \quad \forall j \in N \quad (4.6)$$

$$2y_{ijk} \leq \sum_{\substack{h \in V_L \\ h \neq i}} x_{hi} + \sum_{\substack{l \in N \\ l \neq k}} x_{lk} \quad (4.7)$$

$$\begin{aligned} & \forall i \in N, j \in \{N : i \neq j\}, k \in \{V_R : \langle i, j, k \rangle \in \mathbb{P}\} \\ y_{0jk} & \leq \sum_{\substack{h \in V_L \\ h \neq k \\ h \neq j}} x_{hk} \quad j \in N, k \in \{V_R : \langle 0, j, k \rangle \in \mathbb{P}\} \end{aligned} \quad (4.8)$$

$$u_k - u_i \geq 1 - (n+2)(1 - \sum_{\substack{j \in N \\ \langle i, j, k \rangle \in \mathbb{P}}} y_{ijk}) \quad (4.9)$$

$$\forall i \in V_L, k \in \{V_R : k \neq i\}$$

$$\sum_{\substack{j \in N \\ j \neq i}} \sum_{\substack{k \in V_R \\ \langle i, j, k \rangle \in \mathbb{P}}} y_{ijk} \leq 1 \quad \forall i \in V_L \quad (4.10)$$

$$\sum_{\substack{i \in V_L \\ i \neq k}} \sum_{\substack{j \in N \\ \langle i, j, k \rangle \in \mathbb{P}}} y_{ijk} \leq 1 \quad \forall k \in V_R \quad (4.11)$$

$$u_i - u_j \geq 1 - (n+2)p_{ij} - M(2 - \sum_{\substack{h \in V_L \\ h \neq i}} x_{hi} - \sum_{\substack{k \in N \\ k \neq j}} x_{kj}) \quad (4.12)$$

$$\begin{aligned} & \forall i \in N, j \in \{V_R : j \neq i\} \\ u_i - u_j & \leq -1 + (n+2)(1 - p_{ij}) + M(2 - \sum_{\substack{h \in V_L \\ h \neq i}} x_{hi} - \sum_{\substack{k \in N \\ k \neq j}} x_{kj}) \end{aligned} \quad (4.13)$$

$$\forall i \in N, j \in \{V_R : j \neq i\}$$

$$u_0 - u_j \geq 1 - (n+2)p_{0j} - M(1 - \sum_{\substack{k \in V_L \\ k \neq j}} x_{kj}) \quad \forall j \in V_R \quad (4.14)$$

$$u_0 - u_j \leq -1 + (n+2)(1 - p_{0j}) + M(1 - \sum_{\substack{k \in V_L \\ k \neq j}} x_{kj}) \quad \forall j \in V_R \quad (4.15)$$

$$u_l \geq u_k - M \left(3 - \sum_{\substack{j \in N \\ j \neq l \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk} - \sum_{\substack{m \in N \\ m \neq i \\ m \neq k \\ m \neq l}} \sum_{\substack{n \in V_R \\ n \neq i \\ n \neq k \\ \langle l,m,n \rangle \in \mathbb{P}}} y_{lmn} - p_{il} \right) \quad (4.16)$$

$$\forall i \in V_L, k \in \{V_R : k \neq i\}, l \in \{N : l \neq i, l \neq k\}.$$

$$t_k \geq r_i + \tau_{ik} - M(1 - x_{ik}) \quad \forall i \in V_L, k \in V_R, i \neq j \quad (4.17)$$

$$t_k \leq r_i + \tau_{ik} + M(1 - x_{ik}) \quad \forall i \in V_L, k \in V_R, i \neq j \quad (4.18)$$

$$t'_j \geq r_i + \tau'_{ij} - M(1 - \sum_{\substack{k \in V_R \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall j \in V_D, i \in V_L, j \neq i \quad (4.19)$$

$$t'_j \leq r_i + \tau'_{ij} + M(1 - \sum_{\substack{k \in V_R \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall j \in V_D, i \in V_L, j \neq i \quad (4.20)$$

$$t'_k \geq r'_j + \tau'_{jk} - M(1 - \sum_{\substack{i \in V_L \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall j \in V_D, k \in V_R, j \neq k \quad (4.21)$$

$$t'_k \leq r'_j + \tau'_{jk} + M(1 - \sum_{\substack{i \in V_L \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall j \in V_D, k \in V_R, j \neq k \quad (4.22)$$

$$t'_j \geq r'_j - M(1 - \sum_{\substack{i \in V_L \\ i \neq j}} \sum_{\substack{k \in V_R \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall j \in N \quad (4.23)$$

$$t'_j \leq r'_j + M(1 - \sum_{\substack{i \in V_L \\ i \neq j}} \sum_{\substack{k \in V_R \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall j \in N \quad (4.24)$$

$$r_k \geq t_k + s_L(\sum_{\substack{l \in N \\ l \neq k}} \sum_{\substack{m \in V_R \\ m \neq l \\ m \neq k \\ \langle k,l,m \rangle \in \mathbb{P}}} y_{klm}) + s_R(\sum_{\substack{i \in V_L \\ i \neq k}} \sum_{\substack{j \in N \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) - M(1 - \sum_{\substack{i \in V_L \\ i \neq k}} \sum_{\substack{j \in N \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall k \in V_R \quad (4.25)$$

$$r'_k \geq t'_k + s_L(\sum_{\substack{l \in N \\ l \neq k}} \sum_{\substack{m \in V_R \\ m \neq l \\ m \neq k \\ \langle k,l,m \rangle \in \mathbb{P}}} y_{klm}) + s_R(\sum_{\substack{i \in V_L \\ i \neq k}} \sum_{\substack{j \in N \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) - M(1 - \sum_{\substack{i \in V_L \\ i \neq k}} \sum_{\substack{j \in N \\ \langle i,j,k \rangle \in \mathbb{P}}} y_{ijk}) \quad \forall k \in V_R \quad (4.26)$$

$$r'_k - (r'_j - \tau'_{ij}) - s_L(\sum_{\substack{l \in N \\ l \neq i \\ l \neq j \\ l \neq k}} \sum_{\substack{m \in V_R \\ m \neq k \\ m \neq i \\ m \neq l \\ \langle k,l,m \rangle \in \mathbb{P}}} y_{klm}) \leq \epsilon + M(1 - y_{ijk}) \quad (4.27)$$

$$\forall k \in V_R, j \in C, j \neq k, i \in V_R, \langle i, j, k \rangle \in \mathbb{P}$$

$$w_k \geq 0 \quad \forall k \in V_R \quad (4.28)$$

$$w'_k \geq 0 \quad \forall k \in V_R \quad (4.29)$$

$$w_k \geq t'_k - t_k \quad \forall k \in V_R \quad (4.30)$$

$$w'_k \geq t_k - t'_k \quad \forall k \in V_R \quad (4.31)$$

$$w_0 = 0, w'_0 = 0, t_0 = 0, t'_0 = 0, r_0 = 0, r'_0 = 0 \quad (4.32)$$

$$r_i = r'_i \quad \forall i \in V \quad (4.33)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in V_L, j \in V_R, j \neq i \quad (4.34)$$

$$y_{ijk} \in \{0, 1\} \quad \forall i \in V_L, j \in N, k \in V_R, i \neq j, j \neq k, i \neq k, \langle i, j, k \rangle \in \mathbb{P} \quad (4.35)$$

$$p_{ij} \in \{0, 1\} \quad \forall i, j \in N, i \neq j \quad (4.36)$$

$$p_{0j} = 1 \quad \forall j \in N \quad (4.37)$$

$$0 \leq u_i \leq n + 1 \quad \forall i \in V \quad (4.38)$$

$$t_i \geq 0 \quad \forall i \in V \quad (4.39)$$

$$t'_i \geq 0 \quad \forall i \in V \quad (4.40)$$

$$r_i \geq 0 \quad \forall i \in V \quad (4.41)$$

$$r'_i \geq 0 \quad \forall i \in V \quad (4.42)$$

The objective is to minimize the operational costs. We now explain the constraints. The letter in parenthesis at the end of each bullet item, if any, denotes the association between a MILP constraint and a constraint described in the model:

- Constraint 4.2 guarantees that each node is visited once by either a truck or a drone. **(A)**
- Constraints 4.3 and 4.4 state that the truck must start from and return to the depot. **(Modelling TD)**
- Constraint 4.5 is a subtour elimination constraint. **(Modelling TD)**
- Constraint 4.6 indicates that if the truck visits j then it must depart from j . **(Modelling TD)**
- Constraint 4.7 associates a drone delivery with the truck route. In detail, if we have a drone delivery $\langle i, j, k \rangle$, then there must be a truck route between i and k . **(C)**
- Constraint 4.8 indicates that if the drone is launched from the depot, then the truck must visit k to collect it. **(C)**
- Constraint 4.9 ensures that if there is a drone delivery for $\langle i, j, k \rangle$, then the truck must visit i before k . **(C)**
- Constraints 4.10 and 4.11 state that each node in V_L or V_R can either launch the drone or retrieve it at most once, respectively. **(B)**
- Constraints 4.12, 4.13, 4.14 and 4.15 ensure that if i is visited before j in the truck route, then its ordering constraint must be maintained. **(D)**
- Constraint 4.16, if we have two drone deliveries $\langle i, j, k \rangle$ and $\langle l, m, n \rangle$ and i is visited before l , then l must be visited after k . This constraint avoids the problem of launching a drone between i and k . **(D)**
- Finally, constraints 4.17 to 4.33 ensure that waiting time and endurance is correctly handled. **(E)**

4.2 A GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE (GRASP) FOR TSP-D

This section presents a Greedy Randomized Adaptive Search Procedure (GRASP) [GP10] to solve the min-cost TSP-D. We also adapt our split procedure to solve the min-time TSP-D. In the construction step, we propose a split algorithm that builds a min-cost TSP-D solution from a TSP solution. In the local search step, new operators adapted from the traditional ones are introduced for the min-cost TSP-D. The general outline of our GRASP is shown in Algorithm 5. More specifically, in each iteration, it first generates a TSP tour using a TSP construction heuristic (line 7). In this chapter, we use three heuristics to generate giant tours as follows:

- *k-nearest neighbour*: This heuristic is inspired from the well-known nearest neighbour algorithm for solving the TSP. It starts from the depot, repeatedly visits the node v which is randomly chosen among k closest unvisited nodes.
- *k-cheapest insertion*: The approach is to start with a subtour, i.e., a small tour with a subset of nodes, and then extend this tour by repeatedly inserting the remaining nodes until no more node can be added. The unvisited node v to be inserted and its insertion location between two consecutive nodes (i, j) of the tour are selected so that this combination gives the least Insertion Costs (IC). This cost is calculated by:

$$IC = d_{iv} + d_{vj} - d_{ij} \quad (4.43)$$

To create the randomness for the heuristic, at each insertion step we randomly choose a pair of an unvisited node and its insertion location among k pairs which provides the best insertion costs. The starting subtour includes only the depot.

- *random insertion*: This heuristic works similarly to the *k-nearest neighbour* but it iteratively chooses a random node v among all unvisited nodes.

Algorithm 5: Greedy Randomized Adaptive Search Procedure (GRASP) for min-cost TSP-D

Result: *bestSolution*

```

1 bestSolution = null ;
2 bestObjectiveValue =  $\top$  ;
3 randomGenerator = initialize TSP tour random generator ;
4 iteration = 0 ;
5 while iteration <  $n_{TSP}$  do
6   iteration = iteration + 1 ;
7   tour = generate a random TSP tour using randomGenerator ;
8   (P,V,T) = Split_Algorithm_Step1(tour) ;
9   tspdSolution = Split_Algorithm_Step2(P, V, T) ;
10  tspdSolution = Local_Search(tspdSolution) ;
11  if  $f(\textit{tspdSolution}) < \textit{bestObjectiveValue}$  then
12    bestSolution = tspdSolution ;
13    bestObjectiveValue =  $f(\textit{tspdSolution})$  ;
14 return bestSolution ;
```

In the next step, we construct a min-cost TSP-D solution using the split algorithm (line 8 and 9) and then improve it by local search (line 10). The best solution found is also recorded during the processing of the tours (lines 11 to 13). The algorithm stops after n_{TSP} iterations. The detailed implementation of the split algorithm is described in Algorithms 6 and 7.

4.2.1 A Split Algorithm for min-cost TSP-D

Given a TSP tour, the split procedure algorithm selects nodes to be visited by the drone to obtain a solution for the min-cost TSP-D, assuming that the relative order of the nodes is fixed. Other split procedures are now used widely in state-of-the-art metaheuristics such as [PLR09], [Vid+13a], [Vid+14], [Cat+14] to solve many variants of VRPs. We start from a given TSP tour $s = (s_0, s_1, \dots, s_{n+1})$ and must convert this tour into a feasible min-cost TSP-D solution. This is accomplished by removing nodes from the truck tour and substituting drone deliveries for those nodes. There are two main steps in the split algorithm: auxiliary graph construction and solution extraction. The pseudo code for these is listed in Algorithms 6 and 7, respectively. Although these two steps could be possible merged into one large Algorithm, we decide to split it into two separate ones for the sake of clarity. The most important step of the split algorithm is the construction of the auxiliary graph, in which each subsequence of nodes $(s_i, \dots, s_k)$ can be turned into a drone delivery such that s_i is the launch node, s_k is the rendezvous node and s_j , where $\text{pos}(s_i, s) < \text{pos}(s_j, s) < \text{pos}(s_k, s)$, is the drone node. We now

describe the split algorithm in detail.

Algorithm 6: Split_Algorithm_Step1(s): Building the auxiliary graph and finding shortest path

Data: TSP tour s

Result: P stores the shortest path from the auxiliary graph, V is the cost of that shortest path, and T is a list of the possible drone deliveries and costs

```

1  arcs =  $\emptyset$  ;
2  T =  $\emptyset$  ;
3  /* Auxiliary graph construction - Arcs */
4  foreach i in s \ last(s) do
5      k = pos(i, s) + 1 ;
6      arcs = arcs  $\cup$  (i, k, cost(i, k, s))
7  foreach i in s \ { last(s), s[pos(last(s), s) - 1] } do
8      foreach k in s : pos(k, s)  $\geq$  pos(i, s) + 2 do
9          minValue =  $\infty$  ;
10         minIndex =  $\infty$  ;
11         foreach j in s : pos(i, s) < pos(j, s) < pos(k, s) do
12             if  $\langle i, j, k \rangle \in \mathbb{P}$  then
13                 cost = cost(sub(i, k, s)) +  $C_1(d_{prev_s(j)next_s(j)} -$ 
14                      $d_{prev_s(j),j} - d_{j,next_s(j)}) + cost(i, j, k) +$ 
15                      $cost_W^T(i, j, k) + cost_W^D(i, j, k)$  ;
16                 if cost < minValue then
17                     minValue = cost ;
18                     minIndex = pos(j, s) ;
19             arcs = arcs  $\cup$  {(i, k, minValue)} ;
20             if minIndex  $\neq \top$  then
21                 T = T  $\cup$  {(i, s[minIndex], k, minValue)} ;
22
23  /* Finding the shortest path */
24  V[o] = 0 ;
25  P[o] = 0 ;
26  foreach k in s \ { o } do
27      foreach (i, k, cost)  $\in$  arcs do
28          if V[k] > V[i] + cost then
29              V[k] = V[i] + cost ;
30              P[k] = i ;
31
32  return (P, V, T) ;

```

BUILDING THE AUXILIARY GRAPH AND FINDING SHORTEST PATH

In Algorithm 6, we construct an auxiliary weighted graph $H = (V', A')$ based on the TSP tour s of the graph $G = (V, A)$. We have $V' = V$ and an arc $(i, j) \in A'$ that represents a subroute from i to j , where $pos(i, s) < pos(j, s)$.

If i and j are adjacent nodes in s , then the cost c_{ij} of arc $(i, j) \in A'$ is calculated directly as follows:

$$c_{ij} = C_1 d_{ij}. \quad (4.44)$$

However, when i and k are not adjacent and a node j exists between i and k such that $\langle i, j, k \rangle \in \mathbb{P}$, then

$$c_{ik} = \min_{\langle i, j, k \rangle \in \mathbb{P}} \text{cost}(\text{sub}(i, k, s)) + C_1 \left(d_{\text{prev}_s(j), \text{next}_s(j)} - d_{\text{prev}_s(j), j} - d_{j, \text{next}_s(j)} \right) \\ + \text{cost}(i, j, k) + \text{cost}_W^T(i, j, k) + \text{cost}_W^D(i, j, k). \quad (4.45)$$

If i and k are not adjacent and no node j exists between i and k such that $\langle i, j, k \rangle$ could be a **drone delivery**, then

$$c_{ik} = +\infty. \quad (4.46)$$

The arc's cost calculation is shown in lines 1 to 19 in Algorithm 6. Moreover, in lines 18 and 19, we store the list of possible **drone deliveries** T . This list will be used in the extraction step.

The auxiliary graph is used to compute the cost v_k of the shortest path from the depot to node k . Because the graph H is a directed acyclic graph, these values can be computed easily using a dynamic programming approach. Moreover, an arc (i, k) in the shortest path that does not belong to the initial TSP tour means that a drone delivery can be made where i is the launch node, k is the rendezvous node, and the delivery node is a node between i and k in the TSP tour. This computation ensures that no interference occurs between the chosen drone deliveries. We therefore obtain the best solution from the TSP tour while respecting the relative order of the nodes.

In detail, given $v_0 = 0$, the value v_k of each node $k \in V' \setminus \{0\}$ is then calculated by

$$v_k = \min\{v_i + c_{ik} : (i, k) \in A'\} \quad \forall k = 1, 2, \dots, n+1. \quad (4.47)$$

We also store the shortest path from 0 to $n+1$ in $P(j)$, where $j = 1 \dots n+1$; j is the node, and the value $P(j)$ is the previous node of j . These steps are described in lines 21 to 27 in Algorithm 6. An auxiliary graph for Figure 3.1 is shown in Figure 4.1

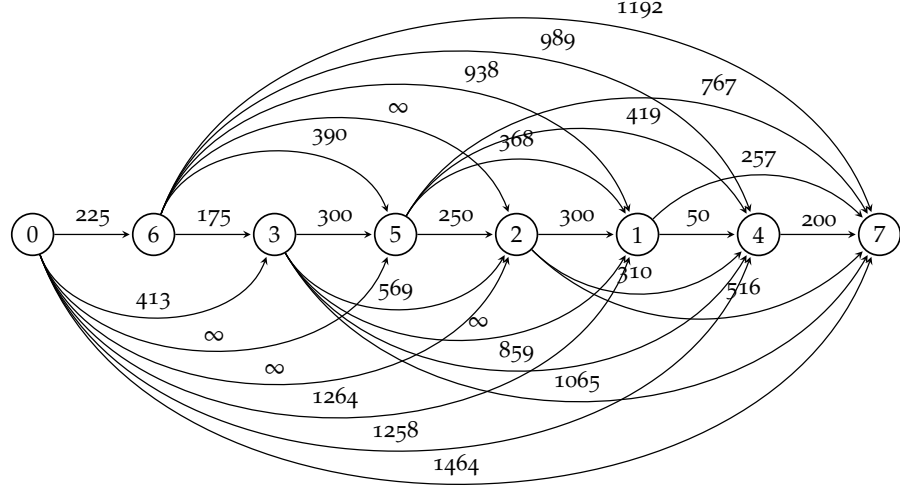


Figure 4.1: Auxiliary graph for TSP tour in Figure 3.1

The cost computation for each arc in the graph H can be done in $\mathcal{O}(n^2)$. Since H is acyclic by construction, to search for a shortest path, a Bellman-Ford algorithm for directed acyclic graphs can be used, with an $\mathcal{O}(|A|)$ complexity where $|A|$ is the number of arcs in the graph. Because the number of arcs in H is proportional to n^2 , the search for a shortest path in graph H can be done in $\mathcal{O}(n^2)$. Several Split procedures in the literature work in a similar manner (see [Pri04], [Pri09a] for example). Therefore, we get the complexity of the Algorithm 6 in $\mathcal{O}(n^4)$.

EXTRACTING MIN-COST TSP-D SOLUTION Given $P(j), j = 1 \dots n+1$ defined as above and a list of possible **drone deliveries** T , we now extract the min-cost TSP-D solution in Algorithm 7. In the first step, given P , we construct a sequence of nodes $S_a = 0, n_1, \dots, n+1$ representing the path from 0 to $n+1$ in the auxiliary graph (lines 2 to 9). Each two consecutive nodes in S_a are a subroute of the complete solution. However, they might include a **drone delivery**; consequently, we need to determine which node might be the drone node in the subroute, which is computed in T .

The second step is to construct a min-cost TSP-D solution. To do that, we first initialize two empty sets: a set of drone deliveries S_d and a set representing the truck's tour sequence S_t (lines 11 and 12). We now build these sets one at a time.

For drone delivery extractions, we consider each pair of adjacent positions i and $i+1$ in P_{new} and determine the number of in-between nodes. If there is at least one j node between the i and $i+1$ positions in the TSP tour, we will choose the drone delivery in T with the minimum value, taking its drone node j as the result (lines 14 to 17).

To extract the truck's tour (line 19 to 26), we start from the depot 0 in S_a . Each pair $i, i + 1 \in S_a$ is considered as a subroute in the min-cost TSP-D solution by taking the nodes from i to $i + 1$ in the TSP solution. However, in cases where i and $i + 1$ are launch and rendezvous nodes of a **drone delivery**, respectively, $\langle i, j, i + 1 \rangle$, j must not be considered in the truck's tour.

Algorithm 7: Split_Algorithm_Step2(P,V,T): Ex-tract_TSPD_Solution

Data: P stores the path in the auxiliary graph, V is the cost of the path in P, T is the list of drone deliveries + costs, and $tspTour$ is the truck-only TSP tour

Result: $tspdSolution$

```

1 /* Construct the sequence of nodes representing the path stored in P
   */
2 j = n + 1 ;
3 i = ∞ ;
4  $S_a = \langle j \rangle$  ;
5 while  $i \neq 0$  do
6   i = P[j] ;
7    $S_a = S_a :: \langle i \rangle$  ;
8   j = i ;
9  $S_a = S_a.reverse()$  ;
10 /* Create a min-cost TSP-D solution from  $S_a$  */
11  $S_d = \langle \rangle$  ;
12  $S_t = \langle \rangle$  ;
13 /* Drone deliveries */
14 for  $i = 0; i < S_a.size - 1; i++$  do
15   if between  $S_a[i]$  and  $S_a[i + 1]$  in  $tspTour$ , there is at least one
       node then
16      $n_{drone} = \text{obtain the associated drone node in tuples T}$  ;
17      $S_d = S_d \cup \langle S_a[i], n_{drone}, S_a[i + 1] \rangle$  ;
18 /* Truck tour */
19 currentNode = 0 ;
20 while currentNode  $\neq n + 1$  do
21   if currentNode is a launch node of a tuple t in  $S_d$  then
22      $S_t = S_t :: \langle \text{all the nodes from the currentNode to the} \rangle$ 
           return node of t in  $tspTour$  except the drone node  $\rangle$  ;
23     currentNode = the return node of t ;
24   else
25      $S_t = S_t :: \langle \text{currentNode} \rangle$  ;
26     currentNode =  $tspTour[indexOf(currentPosition) + 1]$  ;
27  $tspdSolution = (S_t, S_d)$  ;
28 return  $tspdSolution$  ;

```

SPLIT PROCEDURE ADAPTATION FOR MIN-TIME TSP-D To deal with the min-time problem, we change the way the arc's costs are computed in the auxiliary graph as follows: If i and j are adjacent nodes in s , then the cost c_{ij} is calculated by:

$$c_{ij} = \tau_{ij} \quad (4.48)$$

When i and k are not adjacent and a node j exists between i and k such that $\langle i, j, k \rangle \in \mathbb{P}$, then

$$c_{ik} = \min_{\langle i, j, k \rangle \in \mathbb{P}} (\max(\text{time}_T(i \rightarrow k), \text{time}_D(i, j) + \text{time}_D(j, k)) + s_R + s_L). \quad (4.49)$$

where $\text{time}_T(i \rightarrow k)$ is the travel time of truck from launch point i to rendezvous point j and $\text{time}_D(i, j)$ is the travel time of drone from i to j . Eventually, this modification results in a change of Algorithm 6, specifically, line 13 to 15 as follows:

Algorithm 8: Split_Algorithm_Step1(s): Min-time adaptation

```

1 ...
2 timeDrone = timeD(i, j) + timeD(j, k);
3 timeTruck = timeT(i → k);
4 if max(timeTruck, timeDrone) + sR + sL < minVal then
5   minVal = max(timeTruck, timeDrone) + sR + sL ;
6   ...

```

4.2.2 Local search operators

Two of our local search operators are inspired from the traditional move operators Two-exchange and Relocation [KS97]. In addition, given the characteristics of the problem, we also develop two new move operators, namely, "drone relocation", which is a modified version of the classical relocation operator, and "drone removal", which relates to the removal of a drone node. In detail, from a min-cost TSP-D solution (TD, DD) , we denote the following:

- $N_T(TD, DD) = \{e : e \in TD, \langle e, \cdot, \cdot \rangle \notin DD, \langle \cdot, \cdot, e \rangle \notin DD\}$ is the set of **truck-only nodes** in the solution (TD, DD) that are not associated with any drone delivery
- $N_D(TD, DD) = \{e : \langle \cdot, e, \cdot \rangle \in DD\}$ is the set of **drone nodes** in the solution (TD, DD)

We now describe each operator.

Relocation: This is the traditional relocation operator with two differences: (1) We consider only truck-only nodes; (2) we only relocate into a new position in the truck's tour. An example is shown in Figure 4.2. In detail, we denote

$$\text{relocate}_T((TD, DD), a, b), a \in N_T((TD, DD)), b \in TD, b \neq a, b \neq 0 \quad (4.50)$$

as the operator that—in effect—relocates node a before node b in the truck tour.

Drone relocation: The original idea of this operator is that it can change a truck node to a drone node or relocate an existing drone node so that it has different launch and rendezvous locations. The details are as follows: (1) We consider both truck-only and drone nodes; (2) each of these nodes is then relocated as a drone node in a different position in the truck's tour. This move operator results in a neighbourhood that might contain more **drone deliveries**; hence, it has more possibilities to reduce the cost. An example is shown in Figure 4.3. More precisely, we denote

$$\text{relocate}_D((TD, DD), a, i, k) \quad (4.51)$$

$$a \in N_T((TD, DD)) \cup N_D((TD, DD)), i, k \in TD \setminus \{a\}, i \neq k,$$

$$\text{pos}(i, TD) < \text{pos}(k, TD), \langle i, a, k \rangle \in \mathbb{P}$$

as the operator procedure, where a is the node to be relocated and i and k are two nodes in TD . There are two possibilities for effects: (1) If a is a truck-only node, this move creates a new drone delivery $\langle i, a, k \rangle$ in DD and removes a from TD ; (2) if a is a drone node, the move changes the drone delivery $\langle \cdot, a, \cdot \rangle \in DD$ to $\langle i, a, k \rangle$.

Drone removal: In this move operator, we choose a drone node $j \in N_D$ and replace the drone delivery by a truck delivery. An example is shown in Figure 4.4. In detail, we denote

$$\text{remove}_D((TD, DD), j, k), j \notin TD, \langle \cdot, j, \cdot \rangle \in DD, k \in TD, k \neq \{0\} \quad (4.52)$$

as the operator procedure, where j is the drone node to be removed and k is a node in TD such that j will be inserted before k . As a result, we have a new solution in which the number of nodes in TD has been increased by one and DD 's cardinality has been decreased by one.

Two-exchange: We exchange the position of two nodes. There are three possibilities: (1) When the exchanged nodes are both drone nodes, we make the change in the drone delivery list; (2) when the exchanged nodes are both truck nodes, we first exchange their positions in the truck sequence and then apply changes in the drone delivery list; and (3) when the exchanged nodes are a truck node and a drone node, we remove the old tuple and create a new one with the exchanged node. Next, we update the truck sequence and apply the changes to the tuples if the truck node is associated with any drone delivery. An example is shown in Figure 4.5. In detail, we denote

$$\text{two_exchange}((TD, DD), a, b), a, b \in V \setminus \{0, n+1\}, a \neq b \quad (4.53)$$

as the operator procedure, where a and b are the two nodes to be exchanged. We then swap their positions. The three swap possibilities

are as follows: (1) a drone node with a node in TD ; (2) two drone nodes; and (3) two nodes in TD .

To ensure the feasibility of resulting solutions, we only accept the moves which satisfy the constraints of the problem. And finally, our local search operators are easily adapted to deal with the min-time objective. They work on the travel time instead of the travel cost of each arc. Whenever there is a need to update a travel time of a drone delivery, we need to take the greater value between travel times of drone and truck instead of the summation. The mechanism of the rest of the local search then stays untouched.

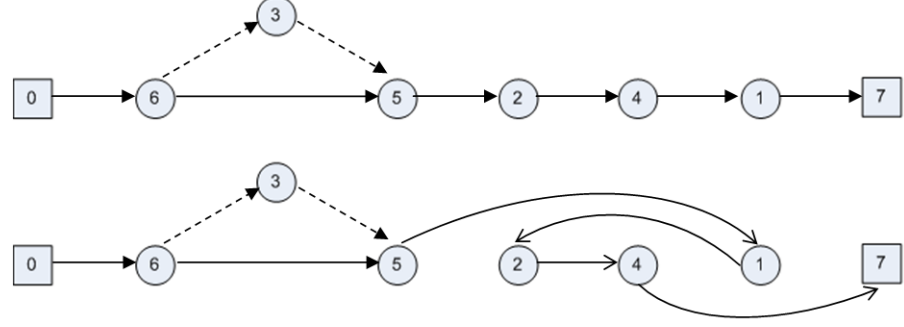


Figure 4.2: A truck relocation move operator : $relocate_T((TD, DD), 1, 2)$

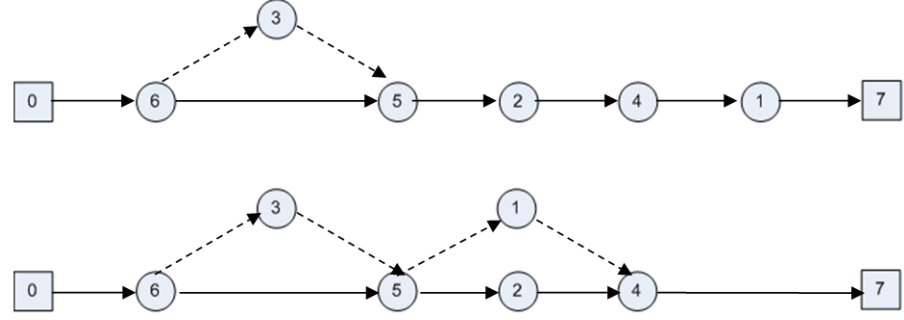


Figure 4.3: A drone relocation move operator : $relocate_D((TD, DD), 1, 5, 4)$

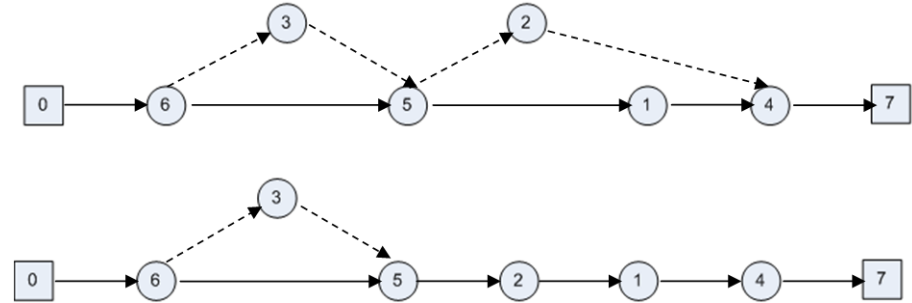


Figure 4.4: A drone removal move operator : $remove_D((TD, DD), 2)$

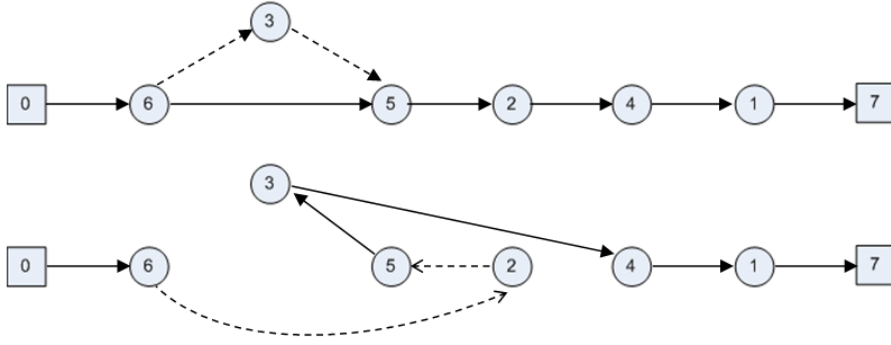


Figure 4.5: A two-exchange move operator in which a drone node is exchanged with a truck node : $two_exchange((TD, DD), 3, 2)$

4.3 TSP-LS HEURISTIC

The TSP-LS algorithm is adapted from the work of [MC15] to solve the min-cost TSP-D. The differences between min-time FSTSP and the adapted min-cost TSP-LS are at the calculation of cost savings (Algorithm 10), the cost of relocating a truck node to another position (Algorithm 11) and the cost of inserting a node as a drone node between two nodes in the truck tour (Algorithm 12). These changes are not only about the unit of measurement (time vs. cost) but also the waiting cost of two vehicles. We now describe the algorithm in details.

The algorithm starts by calculating a TSP tour and then repeatedly relocates customers until no more improvement can be reached. The outline is shown in Algorithm 9. Lines 1–8 define the global variables, which are $Customers = [1, 2, \dots, n]$, the sequence of truck nodes— $truckRoute$, and an indexed list $truckSubRoutes$ of smaller sequences that represent the subroutes in $truckRoute$. The distinct combination of elements in $truckSubRoutes$ must be equal to $truckRoute$. We define i^*, j^*, k^* , where j^* is the best candidate for relocation and i^* and k^* denote the positions between which j^* will be inserted. We also store $maxSavings$ which is the cost improvement value of this relocation. The two Boolean variables $isDroneNode$ and $Stop$ respectively determine whether a node in a subroute is a drone node and whether TSP-LS should terminate. These global variables are updated during the iterations, and the heuristic terminates when no more positive $maxSavings$ can be achieved ($maxSavings = 0$).

Algorithm 9: TSP-LS heuristic

Data: truck-only sequence $truckRoute$
Result: TSP-D solution sol

```

1  $Customers = N$  ;
2  $truckRoute = solveTSP(N)$ ;
3  $truckSubRoutes = \{truckRoute\}$ ;
4  $sol = (truckRoute, \emptyset)$ ;
5  $i^* = -1$ ;
6  $j^* = -1$ ;
7  $k^* = -1$ ;
8  $maxSavings = 0$ ;
9  $isDroneNode = null$ ;
10  $Stop = false$ ;
11 repeat
12   foreach  $j \in Customers$  do
13      $savings = calcSavings(j)$  ;
14     foreach  $subroute$  in  $truckSubRoutes$  do
15       if  $drone(subroute, sol)$  then
16          $(isDroneNode, maxSavings, i^*, j^*, k^*) =$ 
17            $relocateAsTruck(j, subroute, savings)$ ;
18       else
19          $(isDroneNode, maxSavings, i^*, j^*, k^*) =$ 
20            $relocateAsDrone(j, subroute, savings)$ ;
21   if  $maxSavings > 0$  then
22      $(sol, truckRoute, truckSubRoutes, Customers) =$ 
23        $applyChanges(isDroneNode, i^*, j^*, k^*,$ 
24          $sol, truckRoute, truckSubRoutes, Customers)$ ;
25      $maxSavings = 0$  ;
26   else
27      $Stop = true$ ;
28 until  $Stop$ ;
29 return  $truckSubRoutes$ ;

```

Algorithm 10: calcSavings(j)

Data: j : a customer currently assigned to the truck
Result: Solution

```

1  $i = prev_{truckRoute}(j)$  ;
2  $k = next_{truckRoute}(j)$  ;
3  $savings = (d_{i,j} + d_{j,k} - d_{i,k})C_1$  ;
4 if  $j$  is associated with a drone delivery in subroute  $s$  then
5    $i = first(s)$ ;
6    $k = last(s)$ ;
7    $w = \alpha \times \max(0, t_{i \rightarrow k} - \tau_{ij} - \tau_{jk} + \tau_{ik} - t'_{ijk})$  ;
8    $w' = \beta \times \max(0, t'_{ijk} - (t_{i \rightarrow k} - \tau_{ij} - \tau_{jk} + \tau_{ik}))$  ;
9    $savings = savings + w + w'$ 
10 return  $savings$ ;

```

Algorithm 11: `relocateAsTruck(j, subroute, savings)`—
Calculates the cost of relocating the customer j into a
different position in the truck's route

Data:

j : current customer under consideration

s : current subroute under consideration

$savings$: savings that occur if j is removed from its current
position

Result: Updated $i^*, j^*, k^*, isDroneNode$

```

1  $a = first(s)$  ;
2  $b = last(s)$  ;
3 foreach  $(i, k) \in A(s)$  do
4    $\Delta = (d_{i,j} + d_{j,k} - d_{i,k})C_1$  ;
5   if  $\Delta < savings$  then
6     if the drone is still feasible to fly then
7       if  $savings - \Delta > maxSavings$  then
8          $isDroneNode = False$ ;
9          $j^* = j; i^* = i; k^* = k$ ;
10         $maxSavings = savings - \Delta$ ;
11 return  $(isDroneNode, maxSavings, i^*, j^*, k^*)$ ;

```

For an additional notation used in Algorithm 9, line 15, given a solution (TD, DD) , we denote $drone(s, (TD, DD)) \in \{True, False\}$ as *True* if the subsequence s in TD is associated with a drone:

$$drone(s, (TD, DD)) = \begin{cases} True & \text{if } \exists j \in V(s), j \neq first(s), \\ & j \neq last(s) : \langle first(s), j, last(s) \rangle \in DD; \\ False & \text{if } \forall j \in V(s), j \neq first(s), \\ & j \neq last(s) : \langle first(s), j, last(s) \rangle \notin DD. \end{cases}$$

In detail, each iteration has two steps: (1) Consider each customer in *Customers* to determine the best candidate for relocation along with its new position and the cost savings. (2) If the candidate relocation can improve the current solution, then relocate the customer by updating *truckRoute* and *truckSubRoutes* and remove it from *Customers* so that it will not be considered in future iterations; otherwise (when the candidate relocation cannot improve the current solution), the relocation terminates. We now explain each step and its implementations in Algorithms 9, 10, 11, 12, 13.

Step 1 of the iteration is presented from lines 12 to 18 in Algorithm 9. It first considers each customer j (line 12) and then calculates the cost savings by removing j from its current position (line 13). The calculation is shown in Algorithm 10. Next, line 14 considers each subroute in *truckSubRoute* as a possible target for the relocation of j . When the current considered subroute is a **drone delivery** (line 15), we then try to relocate j into this subroute as a truck node (line 16); otherwise, we try to relocate j as a drone node to create a new **drone delivery** (line 18). The relocation analyses of j as a truck node or a drone node are presented in Algorithms 11 and 12, respectively.

In Algorithm 11, we aim to find the best position in subroute s to insert the current customer under consideration j by checking each pair of adjacent nodes i and k in s (line 3). After that, if the cost of inserting j in this position is less than the current *savings*, then relocating j here results in some savings (line 5). Furthermore, because this subroute has a **drone delivery**, we need to check whether inserting j into it still lies within the drone's power limit so that the truck can still pick up the drone (line 6). Finally, if the cost saved is below the best known *maxSavings*, we apply the changes to this location by updating the values of *isDroneNode*, i^* , j^* , k^* and *maxSavings* (lines 7 to 10).

Algorithm 12: relocateAsDrone(j , subroute, savings) - Calculates the cost of relocating customer j as a drone node

Data:

j : current considered customer

s : current considered subroute

savings : current savings if j is removed from its position

Result: Updated i^* , j^* , k^* , *isDroneNode*

```

1 for  $i = 0$  to  $\text{size}(s) - 2$  do
2   for  $k = i + 1$  to  $\text{size}(s) - 1$  do
3     if  $\langle s[i], j, s[k] \rangle \in \mathbb{P}$  then
4        $w_k =$  waiting cost of truck at  $k$  if  $j$  is drone node ;
5        $w'_k =$  waiting cost of drone at  $k$  if  $j$  is drone node ;
6        $\Delta = (d'_{s[i],j} + d'_{j,s[k]})C_2 + w_k + w'_k$  ;
7       if  $\text{savings} - \Delta > \text{maxSavings}$  then
8          $\text{isDroneNode} = \text{True}$ ;
9          $j^* = j; i^* = s[i]; k^* = s[k]$ ;
10         $\text{maxSavings} = \text{savings} - \Delta$ ;
11 return ( $\text{isDroneNode}, \text{maxSavings}, i^*, j^*, k^*$ );
```

In Algorithm 12, we consider the relocation of a customer j in a subroute s that does not have **drone delivery**. The objective is simple: try to make j become the drone node of this subroute to reduce the cost. Hence, we consider each pair of i and k in s , where i precedes k (line 1, 2), and check whether $\langle i, j, k \rangle$ could be a viable **drone delivery** (line 3). We then calculate the cost of this change in lines 4–6. Next, we check whether the relocation is better than the best known *maxSavings* in line 7. Finally, we update the relocation information in lines 8–10 as in Algorithm 11

In step 2 of the iteration in Algorithm 13, when any cost reduction exists (*maxSavings* $\neq 0$), we apply the changes based on the current values of i^* , j^* , k^* , and *isDroneNode*. If *isDroneNode* = *True*, we relocate j^* between i^* and k^* as a drone node, forming a **drone delivery** (line 1 to 5). Otherwise, j^* is inserted as a normal truck node (line 6 to 8). More specifically, these changes take place on the *truckRoute* and *truckSubRoutes*.

Returning to Algorithm 9, after the changes have been applied in line 18, we reset the value of *maxSavings* to 0 to prepare for the next

iteration. Moreover, the algorithm terminates when $maxSavings = 0$ (line 21).

Algorithm 13: applyChanges function

Data: $isDroneNode, i^*, j^*, k^*, sol, truckRoute, truckSubRoutes, Customers$

Result: Updated $truckRoute, truckSubRoutes, t$

```

1 if  $isDroneNode == True$  then
2   The Drone is now assigned to  $i^* \rightarrow j^* \rightarrow k^*$ ;
3   Remove  $j^*$  from  $truckRoute$  and  $truckSubRoutes$ ;
4   Append a new truck subroute that starts at  $i^*$  and ends at
    $k^*$ ;
5   Remove  $i^*, j^*, k^*$  from  $Customers$ ;
6 else
7   Remove  $j^*$  from its current truck subroute;
8   Insert  $j^*$  between  $i^*$  and  $k^*$  in the new truck subroute ;
9 Update  $sol$  using  $truckRoute$  and  $truckSubRoutes$  ;
10 return ( $sol, truckRoute, truckSubRoutes, Customers$ );

```

4.4 EXPERIMENT SETUP

For the experiments, we generate customer locations randomly on a plane. We consider graphs with 10, 50 and 100 customers. These customers are created in squares with three different areas: $100 km^2$, $500 km^2$ and $1000 km^2$. An instance of the TSP-D is characterised by: customer locations, total area of the plane, drone endurance, depot location as well as speed, distance types, travelling cost and time of each vehicle, drone launch time and retrieve time. In total, 65 instances are generated; their characteristics are partially shown in Table 4.1:

Instances	# of Customers	Area (km^2)	Density	Distance (km)	P
A1 to A5	10	100	1	7.43	595
B1 to B10	50	100	0.5	7.13	73053
C1 to C10	50	500	0.1	15.45	10005
D1 to D10	50	1000	0.05	22.19	2932
E1 to E10	100	100	1	7.14	590144
F1 to F10	100	500	0.2	15.21	81263
G1 to G10	100	1000	0.1	21.59	24666

Table 4.1: Instances of min-cost TSP-D

The numbers in this table represent the average values over each class of instances. Three first columns "Instances", "# of Customers", and "Area" are self-explained. Column "Density" represents the number of customers generated in an area unit while column "Distance"

indicates the average Euclidean distance among customers. And finally, column " $|\mathbb{P}|$ " implies the number of possible drone deliveries.

For all instances, the speeds of drone and truck are both set to 40 km/h. Moreover, d_{ij} is calculated using Manhattan distance, while d'_{ij} is in Euclidean distance. The objective here is to partially simulate the fact that the truck has to travel through a road network (which is longer) and the drone can fly directly from an origin to a destination. The drone's endurance ϵ is set to 20 minutes of flight time. The truck's cost C_1 is by default set to 25 times the drone's cost C_2 . Depot location is at the bottom left of the square. To simulate the real situation where not all packages can be delivered by drone, in all instances, only 80 % of customers can be served by drone. Waiting penalty coefficients α and β are set to 10. And finally, the launch time s_L and retrieve time s_R are all set to 1 minute, as in [MC15].

For the results, we denote γ, T , and ρ as the objective value, running time in seconds and performance ratio, respectively, defined as follows:

$$\rho = \frac{value}{referenceValue} \times 100, \quad (4.54)$$

where *value* is the objective value obtained by the considering algorithm and *referenceValue* is the objective value obtained by a reference algorithm. We will specify these algorithms for each experiment. Because we are dealing with a minimization problem, a ratio ρ less than 100 % means that the considered algorithm provides a better solution than the reference algorithm. Furthermore, we denote σ the relative standard deviation percentage in multiple runs. The objective value, running time and performance ratio on average are denoted as γ_{avg} , T_{avg} , and ρ_{avg} . In addition, the geometric mean, which is more appropriate than the arithmetic mean when analysing normalized performance numbers, is used to calculate the values of ρ_{avg} , T_{avg} (see [FW86] for more information).

CPLEX 12.6.2 is used whenever the MILP formulation needs to be solved, and optimal TSP tours are obtained with the state-of-the-art Concorde solver [App+06]. The values of k in k -nearest neighbour and k -cheapest insertion heuristics are chosen randomly between $\{2, 3\}$ to give the best results. Also by experiment, the value of parameter n_{TSP} of GRASP is set to 2000 in all tests. And finally, all instances and detailed results are available at <http://research.ha Quang Minh.com/tspd/index>.

4.5 RESULTS

In this section, we present and analyse the computational results obtained by the proposed methods. The algorithms are implemented in C++ and run on an Intel Core i7-6700 @ 3.4 GHz processor. Different experiments have been carried out to evaluate the performance

of the proposed methods and analyse the impact of parameters: explore the performance of GRASP on different TSP-tour construction heuristics in min-cost TSP-D, compare min-cost TSP-D solutions provided by the proposed heuristics and optimal solutions computed from the MILP formulation (if possible), compare min-cost TSP-D solutions with TSP solutions (i.e., no drone delivery), compare GRASP with TSP-LS on min-cost TSP-D instances, analyse the impact of the drone/truck cost ratio in min-cost TSP-D, and verify heuristics' performance under min-time objective as well as the trade-off between two objectives.

4.5.1 Performance of GRASP on different TSP-tour construction heuristics in the min-cost TSP-D

In this subsection, we evaluate the performance of GRASP under three proposed TSP construction heuristics in the min-cost TSP-D. We also analyse the impact of the local search operators on the behaviour of GRASP. For each instance set labeled from B to G, we select 3 instances. Then each combination of instance and TSP construction heuristic will be run 10 times. With 18 instances, 3 heuristics, 2 local search settings (enable/disable), we have in total: $18 \times 3 \times 10 \times 2 = 1080$ tests. We use TSP optimal solutions (obtained by Concorde) as reference *referenceValue* to calculate the performance ratios ρ .

The columns ρ_{avg}^{tsp} represent the performance ratio on average of TSP solutions obtained by TSP tour generation heuristics. The columns ρ_{avg}^{withLS} , ρ_{avg}^{noLS} respectively report the performance ratio on average of GRASP with and without local search. The results are presented in Table 4.2.

In overall, GRASP with k -nearest neighbour provided the best performance in terms of solution quality, followed by k -cheapest insertion and then random insertion. It is well-known that greedy algorithms such as nearest neighbour and cheapest insertion give better solutions for the TSP than totally random insertion algorithm does (as confirmed again by the columns " ρ_{avg}^{tsp} "); the use of good TSP tours is an important factor to improve the quality of our GRASP. However, although k -cheapest insertion in general gives better TSP tours than k -nearest neighbour, TSP-D solutions obtained from k -nearest neighbour are better than ones obtained from k -cheapest insertion. The reason could be due to our local search operators which seems to work better with k -nearest neighbour. In GRASP with k -cheapest insertion, the local search operators in general converge more prematurely. And as a result, GRASP with k -cheapest insertion stops earlier than GRASP with k -nearest neighbour. In addition, we carried out some additional tests and found that, in general, using optimal TSP tours does not provide best solutions for the min-cost TSP-D.

Instance	k -nearest neighbour					k -cheapest insertion					random insertion				
	ρ_{avg}^{withLS}	σ	T_{avg}	ρ_{avg}^{isp}	ρ_{avg}^{noLS}	ρ_{avg}^{withLS}	σ	T_{avg}	ρ_{avg}^{isp}	ρ_{avg}^{noLS}	ρ_{avg}^{withLS}	σ	T_{avg}	ρ_{avg}^{isp}	ρ_{avg}^{noLS}
B1	66.33	1.26	8.57	142.12	82.55	66.80	0.27	6.94	117.31	77.20	69.92	1.44	66.70	409.23	187.15
B2	74.33	0.66	8.66	146.05	82.25	75.72	1.45	4.56	115.86	82.79	75.80	1.19	58.83	420.47	191.41
B3	71.62	1.22	9.90	137.05	85.11	75.17	0.10	6.09	117.38	86.03	73.80	1.49	57.38	438.95	209.62
C1	71.11	1.01	7.09	143.08	82.05	76.66	1.42	6.23	120.74	85.74	75.44	1.37	43.12	462.91	345.88
C2	72.66	0.91	8.69	150.82	81.78	78.72	0.83	6.54	122.71	84.65	75.80	0.74	52.65	498.46	363.96
C3	81.09	1.58	5.44	147.67	89.66	83.52	0.41	4.67	115.72	87.64	82.46	1.30	43.32	521.19	389.58
D1	77.63	0.79	6.97	146.24	92.73	81.16	0.30	4.85	118.39	89.14	79.36	1.30	58.33	469.68	364.12
D2	72.69	0.81	6.32	140.80	91.17	72.73	0.84	6.08	115.21	81.73	75.11	1.06	51.61	459.79	355.26
D3	74.75	0.74	5.85	144.54	88.93	85.51	0.47	4.27	123.19	90.78	78.04	1.47	52.83	519.60	388.39
E1	70.40	1.21	85.40	137.31	84.81	69.69	0.65	37.35	107.67	76.71	78.07	2.12	605.80	554.25	286.11
E2	70.64	1.13	82.52	135.61	85.71	67.16	0.46	41.68	112.61	74.89	79.31	1.79	605.82	560.64	278.69
E3	71.29	0.64	85.59	135.23	85.79	70.70	0.60	37.46	108.42	76.58	78.93	1.91	605.99	561.45	283.99
F1	75.12	1.27	67.42	144.97	94.16	78.12	0.56	51.06	115.20	84.60	86.37	1.27	604.38	614.22	512.49
F2	74.62	1.40	81.67	148.48	93.44	77.08	1.19	73.21	120.45	86.02	82.57	2.44	604.33	647.65	529.60
F3	76.44	1.27	80.90	137.19	93.56	79.26	0.91	63.74	120.15	87.27	85.02	2.41	604.39	662.81	527.52
G1	78.18	1.81	69.73	149.72	95.99	81.37	0.87	64.65	125.09	89.75	87.29	1.51	604.16	682.44	570.93
G2	78.28	1.36	90.76	148.37	97.76	81.96	0.73	84.38	118.48	89.63	88.66	0.83	604.18	749.13	623.78
G3	74.19	1.04	89.86	146.62	95.86	72.22	0.99	84.66	120.03	82.26	81.75	2.18	604.20	662.59	556.53
Mean	73.88		24.44	143.35	88.92	76.12		17.72	117.39	83.94	79.50		179.69	541.23	362.85

Table 4.2: Performance of GRASP on TSP-tour construction heuristics in min-cost TSP-D

Furthermore, all three heuristics provided stable results with most of standard deviations σ less than 2 %. More precisely, GRASPs with k -nearest neighbour and k -cheapest insertion are more stable than GRASP using random insertion heuristic. From these analyses, we decide to use k -nearest neighbour heuristic to generate TSP tours for GRASP in the next experiments.

4.5.2 Comparison with min-cost TSP-D optimal solutions

In this section, to validate the MILP formulation, we report the results obtained by CPLEX. We also wish to observe the possibility that finds optimal solutions of two approximate approaches GRASP and TSP-LS. The preliminary experiments show that the MILP formulation cannot solve to optimality instances with more than 10 nodes under a time limit of 1 hour. Therefore, in this subsection, we use only the 10-customer instances to compare the solutions obtained by GRASP, TSP-LS and ordinary TSP with the optimal solutions of the min-cost TSP-D computed through the MILP formulation. The *referenceValue* used to compute the ratio ρ is the optimal min-cost TSP-D solution. For each instance, GRASP is repeatedly run 10 times and we record the number of times (in Column *opt*) it can reach the optimality. The comparison results reported in Table 6.2 show that GRASP can find all optimal solutions consuming much less computation time than the MILP formulation. On the other hand, although TSP-LS is faster, it can only find one optimal solution. It is clear that GRASP outperforms TSP-LS in terms of solution quality. In details, GRASP shows a stable performance with standard deviation of 0 (which reported in

Column σ) and can reach to optimality in all cases. From the column "TSP", we observe that using the drone allows to save more than 20 % and up to 53 % of operational costs. Next, we focus on analysing the performance of GRASP and TSP-LS on the larger instances.

Instance	TSP		MILP formulation		GRASP					TSP-LS		
	γ	ρ	γ	T	γ_{avg}	T_{avg}	ρ_{avg}	σ	opt	γ	T	ρ
A1	1007.33	153.01	658.322	46.64	658.322	0.84	100	0	10	810.244	0.013	123.07
A2	955.876	140.58	679.932	144.51	679.932	1.06	100	0	10	777.119	0.006	114.29
A3	985.679	120.31	819.251	133.30	819.251	0.78	100	0	10	819.251	0.005	100
A4	944.645	126.22	748.405	41.31	748.405	1.70	100	0	10	834.89	0.007	111.55
A5	985.679	121.60	810.567	57.18	810.567	1.63	100	0	10	853.728	0.005	105.32

Table 4.3: Comparison with the min-cost TSP-D optimal solution.

4.5.3 Performance of heuristics on the larger instances in the min-cost TSP-D

In this subsection, we aim to analyse the performance of GRASP and TSP-LS on the larger min-cost TSP-D instances—those with 50 and 100 customers. Two methods TSP-LS and GRASP are tested and obtained solutions are compared with ones of the ordinary TSP. The *referenceValue* used to compute the ratio ρ is the objective value of the TSP optimal solution. For each instance, we also report the average waiting times of truck and drone as well as the average latest time at which either the truck or the drone return to the depot (Column w_{avg} , w'_{avg} and t_{avg}). These values are measured in minutes. Again, for each instance, GRASP is repeatedly run 10 times. Tables 4.4 and 4.5 show the results for the instances with 50 and 100 customers, respectively.

As can be observed, GRASP outperforms TSP-LS in terms of solution quality. In terms of running time, GRASP runs slower. However, considering that it never runs in more than 4 minutes, while performs up to 7 % better than TSP-LS in terms of ρ_{avg} , this trade-off is worthy.

$N = 50$	GRASP								TSP-LS	
	γ_{best}	γ_{avg}	ρ_{avg}	T_{avg}	σ	w_{avg}	w'_{avg}	t_{avg}	ρ_{avg}	T_{avg}
B1	1372.82	1413.24	66.59	16.30	1.34	70.48	0.56	192.52	78.62	0.40
B2	1491.30	1513.98	73.42	15.67	0.48	36.13	1.00	162.78	77.42	0.34
B3	1503.78	1521.67	72.06	16.70	0.68	44.11	1.57	165.46	81.44	0.28
B4	1396.17	1426.20	65.33	15.92	0.98	66.97	0.13	190.45	79.38	1.01
B5	1457.91	1500.90	71.52	18.73	1.51	53.38	1.94	178.12	81.28	0.39
B6	1316.08	1353.76	63.87	15.94	1.04	81.87	0.57	198.83	75.51	0.30
B7	1370.05	1399.71	65.90	14.27	0.83	63.16	1.46	183.53	78.69	0.30
B8	1484.93	1517.23	73.24	15.23	0.95	60.04	0.41	184.35	83.03	0.28
B9	1442.09	1468.86	70.27	17.05	0.94	43.65	3.87	168.32	79.19	0.31
B10	1392.54	1429.57	67.94	15.19	1.04	54.40	0.08	174.44	75.62	0.33
C1	2870.41	2935.87	71.70	12.62	0.88	112.76	2.47	318.15	79.52	0.12
C2	2804.47	2868.67	72.97	15.74	0.75	88.56	2.43	293.69	78.67	0.12
C3	3087.55	3185.09	81.87	9.73	1.07	56.35	4.03	272.77	83.06	0.14
C4	2844.10	2916.86	70.97	11.78	0.70	91.54	1.37	297.05	82.39	0.16
C5	3323.92	3367.34	80.54	11.40	0.57	58.89	4.72	286.26	89.21	0.09
C6	3433.99	3472.39	79.79	11.24	0.65	68.47	3.43	301.24	86.71	0.12
C7	3001.13	3047.71	71.92	12.86	0.64	105.33	0.58	317.15	80.75	0.11
C8	3481.17	3557.99	82.21	13.07	1.02	71.66	3.02	312.14	86.84	0.10
C9	3267.23	3306.38	75.35	11.56	0.40	85.48	0.46	311.73	80.09	0.27
C10	3291.20	3356.29	78.34	13.77	0.84	74.47	1.29	304.23	82.22	0.14
D1	4159.39	4389.24	76.86	12.87	1.61	93.40	1.61	382.41	89.35	0.10
D2	4275.46	4334.40	72.32	11.67	0.52	106.81	2.42	392.96	76.75	0.06
D3	4085.71	4191.08	75.25	11.06	1.01	92.96	1.59	368.91	82.21	0.07
D4	4612.46	4714.62	77.14	12.74	0.80	91.33	1.96	399.14	80.52	0.11
D5	4717.67	4793.39	80.26	11.70	0.79	77.54	0.97	390.97	82.89	0.06
D6	4405.02	4485.87	78.64	11.73	0.79	78.61	2.87	373.53	85.93	0.06
D7	4749.57	4796.23	82.77	15.06	0.46	68.48	7.74	384.65	86.69	0.07
D8	4143.03	4287.87	77.71	11.99	1.56	90.75	1.05	374.24	87.62	0.06
D9	4653.73	4688.16	76.11	13.39	0.50	86.84	4.82	392.72	86.34	0.17
D10	4260.60	4301.83	75.33	11.96	0.41	91.95	5.98	375.94	78.89	0.08
Mean			74.09	13.46					81.80	0.15

Table 4.4: Performance of heuristics on 50-customer instances in the min-cost TSP-D

$N = 100$	GRASP								TSP-LS	
	γ_{best}	γ_{avg}	ρ_{avg}	T_{avg}	σ	w_{avg}	w'_{avg}	t_{avg}	ρ_{avg}	T_{avg}
E1	2206.53	2255.99	70.64	137.02	0.96	81.22	1.75	293.35	76.22	5.52
E2	2210.61	2273.09	70.53	136.68	1.08	77.09	2.21	288.01	76.26	5.90
E3	2248.16	2312.76	71.43	148.62	1.09	77.57	1.71	287.39	72.41	6.09
E4	2179.06	2223.97	70.35	178.37	0.90	75.97	1.69	282.29	78.14	6.02
E5	2286.16	2360.30	73.58	172.10	1.31	60.15	2.26	269.52	77.85	6.17
E6	2244.62	2313.86	71.89	195.51	1.05	74.50	1.87	286.99	78.14	5.89
E7	2249.09	2313.67	71.94	190.84	0.90	67.23	2.28	279.10	82.33	6.32
E8	2220.88	2272.55	70.66	189.24	0.79	71.45	1.40	280.26	72.37	6.64
E9	2279.91	2326.29	72.33	172.03	0.83	67.60	1.72	277.53	74.74	5.72
E10	2324.74	2384.52	74.30	204.74	0.96	64.16	1.94	277.90	77.23	4.98
F1	4569.83	4648.20	76.20	111.07	0.85	109.53	6.36	443.43	83.13	1.23
F2	4186.76	4318.78	74.74	143.07	1.47	138.73	2.39	459.57	80.43	1.19
F3	4414.38	4563.64	76.57	146.75	1.31	119.39	6.88	454.68	81.77	1.46
F4	4499.09	4600.27	79.53	128.53	1.25	123.85	3.15	456.31	80.99	1.38
F5	4381.37	4597.32	76.34	159.76	1.66	129.97	1.88	464.85	80.65	1.13
F6	4032.90	4171.80	74.54	157.70	1.53	130.63	3.55	442.99	79.74	1.06
F7	4076.31	4213.52	72.64	170.14	1.62	159.33	1.44	478.18	74.39	1.29
F8	4491.20	4597.90	75.37	165.96	0.98	126.90	4.52	464.09	82.89	1.31
F9	4388.91	4463.39	75.24	153.04	0.90	124.91	3.43	455.09	83.62	1.52
F10	4173.64	4567.84	76.48	153.99	2.31	118.52	3.07	451.57	80.48	1.51
G1	5947.97	6148.50	77.05	116.24	1.61	163.73	3.72	589.97	79.66	0.65
G2	5882.97	5987.64	79.70	158.00	0.76	118.74	5.14	532.92	81.70	0.52
G3	6074.57	6138.94	74.64	169.26	0.80	163.40	2.78	585.57	78.02	1.03
G4	6458.96	6632.14	82.34	143.47	1.16	135.84	5.02	588.44	85.79	0.90
G5	6198.95	6329.25	80.46	155.52	0.73	127.68	4.00	563.97	82.04	0.58
G6	6049.34	6343.26	77.02	177.07	1.69	149.52	6.42	589.69	81.67	0.64
G7	5889.08	6023.11	75.66	171.24	0.88	141.96	4.50	557.86	75.98	0.81
G8	5599.55	5871.96	71.99	156.90	1.87	159.24	6.95	570.88	80.03	0.86
G9	6050.80	6254.50	74.29	184.69	1.48	174.62	2.73	609.20	80.87	0.89
G10	6249.69	6534.13	79.63	162.11	1.97	124.85	6.79	572.85	83.47	0.78
Mean			74.87	158.77					79.36	1.79

Table 4.5: Performance of heuristics on 100-customer instances in the min-cost TSP-D

In all cases, GRASP finds the best solutions. Regardless of slower speed, its average computational time is acceptable on even 100-customer instances (about 2.5 minutes averagely). Furthermore, its relative standard deviation percentage – reported in Column σ – is less than 3% in all instances, proving the stability of the algorithm. The results obtained once again prove the effectiveness of using the drone for delivery. GRASP gives solutions with a cost saving of more

than 25 % compared with the TSP optimal solutions, which do not use any drone delivery.

Regarding the waiting times, one can observe that in min-cost TSP-D solutions, truck has to wait for drone most of the time among all instances ($w_{avg} > w'_{avg}$). In details, while drone's waiting times are only a couple of minutes, truck's waiting times make up approximately 25.95% and 26.20% of the delivery completion time (t_{avg}) regarding to geometric mean in 50-customer and 100-customer instances, respectively. This could be due to the fact that truck's transportation cost is much larger than drone's transportation cost (25 times larger), the min-cost TSP-D solutions tend to select drone deliveries in which flying distance of drone is quite longer than traveling distance of truck.

4.5.4 Impact of cost ratio in the min-cost TSP-D

In this experiment, we explore the impact of the drone/truck cost ratio on the objective values of the min-cost TSP-D solutions provided by the GRASP and TSP-LS algorithms. By default, this parameter is set to 1:25; therefore, we added two more ratios, 1:10 and 1:50. Table 4.6 shows the geometric mean values of ρ_{avg} for the two heuristics. The *referenceValue* used to compute the ratio ρ is the objective value of the TSP optimal solution. Again, for each instance, GRASP is repeatedly run 10 times.

Cost ratio	N = 50		N=100	
	GRASP	TSP-LS	GRASP	TSP-LS
1:10	79.41	84.94	80.63	82.92
1:25	74.09	81.80	74.86	79.36
1:50	71.93	80.70	72.53	78.05

Table 4.6: Performance of heuristics with different cost-ratio settings in min-cost TSP-D.

Logically, the value of ρ_{avg} should decrease as the ratio increases. However, it does not reduce proportionally. More specifically, for GRASP, when the ratio changes from 1:10 to 1:25, the mean of ρ_{avg} decreases by approximately 5% for the 50-customer instances and approximately 6% for the 100-customer instances. In contrast, as the ratio changes from 1:25 to 1:50, the mean of ρ_{avg} decreases by only approximately 3% in both cases. The same phenomenon is observed for TSP-LS. Consequently, when constructing distribution networks for drone/truck combinations, overestimating the transportation cost of the drone does not always improve significantly the results. This means that the efficiency of investment in improving the cost ratio should be carefully considered because such an investment may prove more expensive than the savings in operational costs.

Varying the cost ratio does not significantly impact the relative performance between the heuristics. The GRASP still outperforms the TSP-LS in all cases in terms of solution quality but is slower in terms of running time.

4.5.5 Performance of heuristics with min-time objective

In this section, we analyse the performance of proposed algorithms under min-time objective. We first compare the solutions provided by GRASP with the best ones found by [MC15] on 10-customer instances. We then evaluate the performance of these two heuristics on the larger instances proposed in Section 4.4.

Instance	ϵ	TSP	Murray et al.	GRASP	Instance	ϵ	TSP	Murray et al.	GRASP
37v1	20	57.446	56.468	56.468	40v7	20	60.455	49.996	49.470
37v1	40	57.446	50.573	50.573	40v7	40	60.455	49.204	49.233
37v2	20	54.184	53.207	53.207	40v8	20	73.255	62.796	62.270
37v2	40	54.184	47.311	47.311	40v8	40	73.255	62.270	62.033
37v3	20	54.664	53.687	53.687	40v9	20	54.517	42.799	42.533
37v3	40	54.664	53.687	53.687	40v9	40	54.517	42.799	42.533
37v4	20	67.464	67.464	67.464	40v10	20	54.055	43.076	43.076
37v4	40	67.464	66.487	66.487	40v10	40	54.055	43.076	43.076
37v5	20	58.022	50.551	47.457	40v11	20	60.455	49.204	49.204
37v5	40	58.022	45.835	45.835	40v11	40	60.455	49.204	49.204
37v6	20	54.184	45.176	45.145	40v12	20	73.255	62.004	62.004
37v6	40	54.184	45.863	44.602	40v12	40	73.255	62.004	62.004
37v7	20	54.664	49.581	49.581	43v1	20	69.586	69.586	69.586
37v7	40	54.664	46.621	46.754	43v1	40	69.586	55.493	55.493
37v8	20	67.464	62.381	62.381	43v2	20	72.146	72.146	72.146
37v8	40	67.464	59.776	59.614	43v2	40	72.146	58.053	58.053
37v9	20	58.022	45.985	42.585	43v3	20	77.344	77.344	77.344
37v9	40	58.022	42.416	42.416	43v3	40	77.344	69.175	69.175
37v10	20	54.184	42.416	41.908	43v4	20	90.144	90.144	90.144
37v10	40	54.184	41.729	40.908	43v4	40	90.144	82.700	82.700
37v11	20	54.664	42.896	42.896	43v5	20	69.586	55.493	53.053
37v11	40	54.664	42.896	42.896	43v5	40	69.586	53.447	52.093
37v12	20	67.464	56.696	56.696	43v6	20	72.146	58.053	55.209
37v12	40	67.464	55.696	55.696	43v6	40	72.146	52.329	52.329
40v1	20	54.517	49.430	49.430	43v7	20	77.344	64.409	64.409
40v1	40	54.517	46.886	46.886	43v7	40	77.344	60.743	60.886
40v2	20	54.055	50.708	50.708	43v8	20	90.144	77.209	77.209
40v2	40	54.055	46.423	46.423	43v8	40	90.144	73.967	73.727
40v3	20	60.455	56.102	56.102	43v9	20	69.586	49.049	46.931
40v3	40	60.455	53.933	53.933	43v9	40	69.586	47.250	46.931
40v4	20	73.255	69.902	69.902	43v10	20	72.146	47.935	47.935
40v4	40	73.255	68.397	67.917	43v10	40	72.146	47.935	47.935
40v5	20	54.517	43.533	43.533	43v11	20	77.344	57.382	56.395
40v5	40	54.517	43.533	43.533	43v11	40	77.344	56.395	56.395
40v6	20	54.055	44.076	44.076	43v12	20	90.144	69.195	69.195
40v6	40	54.055	44.076	44.076	43v12	40	90.144	69.195	69.195

Table 4.7: Comparison between GRASP and the best solutions found by [MC15]

Performance on small instances

We now compare the performance of GRASP with Murray et al. - the best recorded results found in [MC15] on small size instances of 10 customers spreading in a region of 8-mile square. These results have

been selected from different approaches proposed in [MC15] including MILP formulation with Gurobi solver and FSTSP heuristic with different TSP tour constructions (IP, Savings, Nearest Neighbor, and Sweep). The detailed results are presented in Table 4.7. The best found solutions are appeared in bold. Column ϵ represents the drone's endurance in minutes, while column "TSP" contains the optimal TSP solution values.

In overall, GRASP performs better than the methods presented in [MC15]. Among 72 instances, GRASP provides solutions worse than those of Murray et al. in only three instances while improves the results of Murray et al. in 20 instances. These results demonstrate the performance of our algorithm to solve not only min-cost TSP-D but also min-time TSP-D.

Performance on larger instances

Since larger instances of the min-time TSP-D are not available in the literature, we use the instances proposed in this work to analyse further the performance of GRASP and TSP-LS under min-time objective. Moreover, we vary the drone's speed among three values 25 km/h, 40 km/h and 55 km/h as in [MC15]. Again, GRASP is repeatedly run 10 times for each combination of instance and drone's speed. Some preliminary experiments show that, on many min-time TSP-D instances, solutions provided by GRASP were worse than ones provided by TSP-LS and even ordinary TSP. Upon further investigation, we found out that very few drone deliveries were used and min-time TSP solutions were not far from TSP solutions. As a result, TSP-LS which constructs optimal TSP tours and then improves them with local search operators could perform better than GRASP. In this experiment, we use an additional setting of GRASP that we call GRASP+: only one iteration with an optimal TSP tour is performed.

The results are shown in Table 4.8. The columns and their descriptions are similar to Tables 4.4 and 4.5 in Section 4.5.3 except that the *referenceValue* used to compute the ratio ρ is the objective value of the TSP optimal solution regarding to traveling time (instead of transportation cost in the previous section). In addition, the average objective value of GRASP+ is reported in minutes in Column γ_{avg} .

As can be observed, GRASP has better performance than TSP-LS on 50-customer instances, but performs slightly worse than TSP-LS on 100-customer instances regarding to solution quality. Its solutions are even worse than those of optimal TSP on the instances of class "E". This phenomenon could be due to high launch and retrieve times of drone (totally 2 minutes while the average traveling time of truck between two customers is only about 30 seconds - See Table 4.1) leading to the low frequency of using drone in min-time TSP-D solutions. This also leads to an observation that, on the instances generated in wider regions, more drone deliveries are used and more savings in

objective values of the min-time TSP-D are created. For example, the instances of type "E" have averagely 590,144 possibilities of drone deliveries and 100 customers, but only less than 6 drone deliveries are used (as showed in Table 4.10 of the next section). But on the instances of types "F" and "G", the number of average used drone deliveries increases to 16 and 19, respectively. GRASP is still much slower than TSP-LS due to its higher-level characteristic, but similarly to Section 4.5.3, its running time is acceptable, slightly more than a minute in terms of geometric mean.

In overall, GRASP+ performs better than TSP-LS on all instance classes in terms of solution quality. Regarding the running time, GRASP+ also runs much faster with the average running time less than 0.1 second. These demonstrate the performance of our Split and local search operators.

In terms of waiting times (Columns w_{avg} and w'_{avg}), we can observe an opposite phenomenon with respect to Section 4.5.3. Unlike min-cost TSP-D solutions where truck has to wait for drone most of the time ($w_{avg} > w'_{avg}$), min-time solutions tend to choose drone deliveries in which drone has to wait for truck ($w_{avg} < w'_{avg}$).

Instances	GRASP+					GRASP		TSP-LS	
	γ_{avg}	ρ_{avg}	T_{avg}	w_{avg}	w'_{avg}	ρ_{avg}	T_{avg}	ρ_{avg}	T_{avg}
B	121.2	95.88	0.01	2.51	21.64	96.16	25.64	97.56	0.22
C	232.8	92.80	0.01	4.93	28.08	93.40	21.09	94.28	0.10
D	323.4	92.49	0.01	9.14	29.14	92.84	22.33	94.08	0.08
E	190.2	98.86	0.04	0.14	30.77	101.79	275.08	98.88	1.62
F	334.8	94.49	0.03	2.97	52.84	97.86	151.60	96.88	0.44
G	449.4	92.90	0.02	6.34	51.15	96.91	142.16	95.13	0.33
Mean		94.54	0.01			96.45	64.44	96.12	0.27

Table 4.8: Performance of heuristics of min-time TSP-D

4.5.6 Comparison between min-cost and min-time objectives

In this experiment, we compare the TSP-D under two objectives in terms of performance ratios and the frequency of using drones given the results from Sections 4.5.3 and 4.5.5. The experimental results are presented in Tables 4.9 and 4.10.

In Table 4.9, Columns $\rho_{min-time}^{time}$ and $\rho_{min-time}^{cost}$ represent the performance ratios of two objective values, i.e. delivery completion time and operational cost, of the best solutions found in min-time TSP-D problems. Similarly, Columns $\rho_{min-cost}^{time}$ and $\rho_{min-cost}^{cost}$ are delivery completion time and operational cost of the best solution found in min-cost TSP-D problems. Again, the *value* in performance ratio is the optimal TSP solution calculated in time and cost, depending on the objective type. In overall, min-time TSP-D solutions not only reduce

the delivery completion time but also the operational cost compared to corresponding optimal TSP solutions. In contrast, min-cost TSP-D solutions increase the completion time compared to optimal TSP solutions by up to 56.25 % and 43.55 % for 50 and 100-customer instances. However, min-cost TSP-D solutions can save averagely 30 % of the operational cost compared to 20 % of min-time TSP-D solutions. This proves the importance of the new min-cost objective function in the class of transportation problems with drone.

Instances	$\rho_{min-time}^{time}$	$\rho_{min-time}^{cost}$	$\rho_{min-cost}^{time}$	$\rho_{min-cost}^{cost}$
B	91.83	80.60	156.25	62.10
C	86.37	76.89	115.35	69.20
D	81.82	77.29	108.31	71.34
E	97.31	90.27	143.55	68.59
F	90.88	81.31	128.44	69.88
G	85.63	77.07	111.15	68.65
Mean	88.83	80.44	125.99	68.23

Table 4.9: Trade-off between min-cost and min-time in terms of performance ratio

In Table 4.10, we observe the number of times the drone is used in both min-cost TSP-D solutions (Column "Min-Cost") and min-time TSP-D solutions (Columns "Min-Time 25", "Min-Time 40" and "Min-Time 55" representing the cases where the drone speed is set respectively to 25 km/h, 40 km/h and 55 km/h). The results show that min-cost solutions tend to use the drone to service up to 40% of the customers whereas min-time solutions only use the drone to service up to 22% of the customers. This is because, as analysed above, using drone likely helps to reduce the transportation cost but this is not the case for the delivery completion time. Additionally, increasing drone speed leads to higher frequency of using drones.

Instances	Number of drone uses			
	Min-Cost	Min-Time 25	Min-Time 40	Min-Time 55
B	20.27	4.70	6.96	8.04
C	18.19	4.43	9.46	11.27
D	17.89	3.10	10.20	12.47
E	39.95	5.30	5.77	5.83
F	36.77	9.57	15.97	18.58
G	36.44	8.73	18.66	21.90

Table 4.10: The frequency of using drone in two objective functions

4.6 CONCLUSION

This chapter presents a new variant of the Traveling Salesman Problem with Drone (TSP-D) whose objective is to minimize the total operational costs including the transportation cost and the waiting penalties. We propose a model, a mixed integer linear programming formulation and two heuristic methods—GRASP and TSP-LS—to solve the problem. The MILP formulation is an extension of the mathematical model propose in [MC15]; TSP-LS is adapted from an existing heuristic, while GRASP is based on our new split algorithm and local search operators. Numerous experiments conducted on a variety of instances of both objective functions show the performance of our GRASP algorithm. Overall, it outperforms TSP-LS in terms of solution quality in an acceptable running time. The results also demonstrate the important role of the new objective function in the new class of vehicle routing problems with drone. Future researches could aim for proposing more efficient metaheuristics which based on our split procedure. The extension of the proposed methods to solve problems with multiple vehicles and multiple drones could be also an interesting research direction.

A HYBRID GENETIC SEARCH ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM WITH DRONE

5.1 A HYBRID GENETIC ALGORITHM FOR TSP-D (HGA)

In this section, we describe a hybrid genetic algorithm with adaptive diversity control method for solving TSP-D. The framework, as proposed in [Vid+12], is a hybrid metaheuristic that combines the exploration capability of genetic algorithms with efficient local search and diversity control. We adapt this general framework with modifications to match the characteristics of the TSP-D. They include new local search operators, crossovers, a penalized mechanism and a restoration method to convert from a TSP-D solution to a giant-tour chromosome. We describe the approach in Algorithm 14. In detail, starting from an initial population (Line 1), for each iteration, two parents are selected to generate an offspring individual using a crossover operator (Line 4). This offspring then goes through a split procedure (proposed in [Ha+18b]) to obtain the drone delivery and truck delivery chromosome (Line 5). Subsequently, the offspring is “educated” by a local search method – which contains multiple operators – to improve its quality. The educated offspring then employs a restore method to update its giant tour chromosome (Line 7). The offspring is then checked for feasibility and is added to the appropriate subpopulation (feasible or infeasible). It also has a probability of being repaired of $P_{rep} = 50\%$ and is added to a feasible subpopulation if the repair succeeds (Lines 8 to 14). In the next step, if a subpopulation reaches its maximum size, a survivor selection method is called to eliminate a number of individuals in that subpopulation, keeping only the best ones (Lines 15 to 17). The method then adjusts the penalty parameters (Line 18) and calls the *diversification* procedure if the search is not improved after a certain number of iterations (Line 19). Finally, we return the best feasible solution found (Line 22).

The rest of this section is arranged as follows. We first define the search space in Section 5.1.1. Section 5.1.2 describes the solution representation. Section 5.1.3 presents the evaluation of individuals. Parent selection and crossover are described in Section 5.1.4. Section 5.1.5 discusses the local search procedure, and various operators are presented. The restore method is introduced in Section 5.1.6. Finally, Section 5.1.8 regards the population management with the population initialization, adjustment of penalty coefficients, survivor selection and diversity control.

Algorithm 14: HGA for TSP-D

```

1 Initialize population ;
2 while number of iterations without improvement <  $Iter_{NI}$  do
3   Select parents  $P1$  and  $P2$  ;
4   Generate offspring individual  $C$  from  $P1$  and  $P2$  ;
5   Apply split on  $C$  ;
6   Educate  $C$  using local search ;
7   Call restore method to update the giant-tour chromosome
   in  $C$  ;
8   if  $C$  is infeasible then
9     insert  $C$  into infeasible subpopulation ;
10    with probability  $P_{rep}$ , repair  $C$  ;
11   if  $C$  is feasible then
12     insert  $C$  into feasible subpopulation ;
13   if maximum subpopulation size reached then
14     Select survivors ;
15   Adjust the penalty parameters for violating the drone
   endurance constraint ;
16   if best solution is not improved for  $Iter_{DIV}$  iterations then
17     Diversify population
18 Return the best feasible solution ;

```

5.1.1 Search space

It has been well studied that by exploiting infeasible solutions, we can significantly improve the performance of a heuristic [GH11]. In this section, we define the search space $\mathcal{S}$, which includes the feasible and infeasible solutions $s \in \mathcal{S}$. Infeasible solutions comprise drone deliveries that violate the drone endurance constraint. More specifically, a drone delivery in a TSP-D solution is not valid in the following two scenarios.

- The truck travel time constraint is violated (except for the case where the launch node is the depot, as described in the problem description above):

$$\tau_{i \rightarrow k} + \Gamma_{truck} > \epsilon,$$

where $\tau_{i \rightarrow k}$ is the truck travel time from i to k .

- The drone travel time constraint is violated:

$$\tau'_{ij} + \tau'_{jk} + \Gamma_{drone} > \epsilon.$$

In these two cases, the drone cannot feasibly be flown because its battery will be depleted before the retrieval operation undertaken by the truck driver is completed.

Let $sol(s)$ represent a TSP-D solution in the search space. We have $sol(s) = (TD, DD)$, where $TD = \langle e_0, \dots, e_k \rangle$, $e_i \in V$ is the truck tour, and $DD \subseteq \mathcal{P}$ is the set of drone deliveries in solution s .

We now define the fitness evaluation function for min-time and min-cost TSP-D separately.

Min-cost TSP-D

For min-cost TSP-D, the operational cost of solution s , denoted $cost(TD, DD)$, is calculated as

$$cost(TD, DD) = cost(TD) + cost(DD) + cost_W(DD), \quad (5.1)$$

where

- $cost(TD) = \sum_{i=0}^{k-1} C_1 d_{i,i+1}$ is the cost of the truck tour;
- $cost(DD) = \sum_{\langle i,j,k \rangle \in DD} C_2 (d'_{ij} + d'_{jk})$ is the cost of drone deliveries;
- and
- $cost_W(DD) = \sum_{\langle i,j,k \rangle \in DD} cost_W^T(\langle i,j,k \rangle) + cost_W^D(\langle i,j,k \rangle)$ is the wait cost of the truck and drone. We have $cost_W^T = \alpha \times \max(0, \tau_{i \rightarrow k} - \tau'_{ijk})$, where $\tau_{i \rightarrow k}$ is the truck travel time from i to k (in the truck tour), and τ'_{ijk} is the drone travel time from i to j to k . In addition, $cost_W^D = \beta \times \max(0, \tau'_{ijk} - \tau_{i \rightarrow k})$.

Let ω represent the penalty for violating the drone endurance constraint. We define the **penalized cost** of a solution s as the sum of the operational cost and the weighted sum of the truck's or drone's excess travel time during drone deliveries. This **penalized cost** is computed as

$$\begin{aligned} \phi(s) = cost(TD, DD) + \omega \sum_{\langle i,j,k \rangle \in DD} & \max(0, \tau_{i \rightarrow k} + \Gamma_{truck} - \epsilon) \times Y_T \times C_1 + \\ & \max(0, \tau'_{ij} + \tau'_{jk} + \Gamma_{drone} - \epsilon) \times Y_D \times C_2, \end{aligned} \quad (5.2)$$

where ω is the penalty for violating the constraint, and Y_T and Y_D are the speeds of the truck and drone, respectively. This penalized cost is then used as the fitness function to compute the fitness of the individuals.

Min-time TSP-D

In min-time TSP-D, the completion time of a solution s , denoted $time(s)$, is calculated as

$$time(s) = \max(t_{n+1}, t'_{n+1}). \quad (5.3)$$

Similar to min-cost TSP-D, we also have the penalized cost of a solution s in the min-time objective as the sum of the completion time of two vehicles and the penalties for violating the constraint. It is computed as follows:

$$\phi(s) = \text{time}(s) + \omega \sum_{\langle i,j,k \rangle \in DD} \max(0, \max(\tau_{i \rightarrow k} + \Gamma_{truck}, \tau'_{ij} + \tau'_{jk} + \Gamma_{drone}) - \epsilon). \quad (5.4)$$

Again, this is used to compute the fitness of individuals.

5.1.2 Solution representation

A solution in HGA is represented as a giant TSP tour (giant tour) with the two depots removed. We also denote this as a (giant-tour) chromosome. When a TSP-D solution is needed for a local search method, it can be obtained using the *split* procedure, which runs in polynomial time [Ha+18b]. Conversely, we can retrieve a giant tour from a TSP-D solution by using a *restore method*, which will be discussed in the coming section. To conclude, by having a transformation between a giant tour and TSP-D solution using the split and restore method, we can use the fast and efficient operators in both the crossover and local search step. A demonstration of this transformation is described in Figure 5.1.

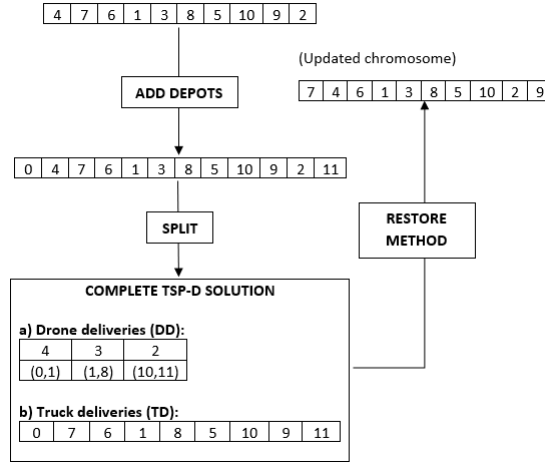


Figure 5.1: Transformations between giant-tour chromosome and TSP-D solution

5.1.3 Individual evaluation

This section discusses the individual assessment method which has been proposed in [Vid+12]. To evaluate any individual P_1 in the population, we consider two factors: its penalized cost $\phi(P_1)$ (described in Section 5.1.1) and its contribution to the diversity of the population,

denoted $\Delta(P_1)$ and calculated as the average “distance” from P_1 to its closest neighbours in the population. By taking into account these two factors, we can obtain a balance between intensification and diversification. Otherwise, the heuristic might either converge too soon and too quickly (focusing only on improving the penalized cost) or will always explore completely different giant tours, leading to a random search. In detail, the diversity contribution described above is presented in Equation 5.5:

$$\Delta(P_1) = \frac{1}{n_{close}} \sum_{P_2 \in \mathcal{N}_{close}} \delta(P_1, P_2), \quad (5.5)$$

where n_{close} is the number of considered closest neighbours, and $\mathcal{N}_{close}$ is the set of closest neighbours of P_1 (i.e., the set of elements sorted using Equation 5.6). The distance between two individuals P_1 and P_2 , denoted $\delta(P_1, P_2)$, is a normalized Hamming distance based on the differences between the nodes in the same positions of the giant-tour chromosome. This distance is shown in Equation 5.6, where $\mathbf{1}(\text{condition})$ is a valuation function that returns 1 if the condition is true and 0 otherwise.

$$\delta(P_1, P_2) = \frac{1}{n} \sum_{i=1, \dots, n} [\mathbf{1}(P_1^{gt}(i) \neq P_2^{gt}(i))], \quad (5.6)$$

where $\mathbf{1}(P_1^{gt}(i) \neq P_2^{gt}(i))$ returns 1 if the node in position i of the giant-tour chromosome in P_1 is different than the node in position i of the giant-tour chromosome of P_2 , and 0 otherwise.

The evaluation of an individual P , or as we call it, the **biased fitness**, denoted $BF(P)$, is then computed as in Equation 5.7, where $fit(P)$ is the rank of P in the subpopulation of size $nbIndiv$ with respect to its penalized cost $\phi(P)$, and $dc(P)$ is the rank of P in the subpopulation in terms of diversity contribution. The parameter $nbElite$ ensures that a certain number of elite individuals will survive to the next generation during the survival selection process (proven in [Vid+12]).

$$BF(P) = fit(P) + (1 - \frac{nbElite}{nbIndiv})dc(P) \quad (5.7)$$

5.1.4 Parent selection and crossover

Each iteration in HGA includes a generation of a new child chromosome. This is done by first merging two subpopulations into one population and randomly selecting two parents, P_1 and P_2 , in that population using the tournament selection method. In detail, to choose a parent, we pick two individuals from the complete population above and select the one with the best biased fitness. Two parents have then gone through a crossover step.

For crossover operators, one can use the classical TSP crossovers – OX (order crossover), PMX (partially mapped crossover), OBX (order-based crossover), and PBX (position-based crossover) [Pot96]. In this paper, we propose a problem-dependent crossover called DX that can solve the TSP-D more effectively. The most important feature of DX is that it exploits the characteristics of a TSP-D solution – the drone deliveries and truck deliveries – and try to transmit that information from the parents to the offspring. A detailed description of this crossover is presented in Figure 5.2, and the crossover is described in Algorithm 15.

Algorithm 15: Crossover DX for TSP-D

- 1 **Input:** Parents P_1, P_2 and the corresponding TSP-D solution of P_1 which is (TD_1, DD_1) ;
 - 2 Let $TSP_1 = P_1$ with 2 depots added, $TSP_2 = P_2$ with 2 depots added ;
 - 3 Let C = An empty chromosome with 2 depots added ;
 - 4 Let r = A random number in range of $[0, 1]$;
 - 5 **if** $r \leq 0.5$ **then**
 - 6 Choose 2 cut points $a, b, a < b$ in TD_1 and copy the nodes between these cut points to C while respecting its position in TSP_1 ;
 - 7 **else**
 - 8 Choose 2 cut points $a, b, a < b$ in DD_1 and copy the nodes between these cut points to C while respecting its position in TSP_1 ;
 - 9 Fill the other positions of C , starting at position 1, by taking the remaining nodes of TSP_1 while keeping their relative orders in TSP_2 . Return C with 2 depots removed.
-

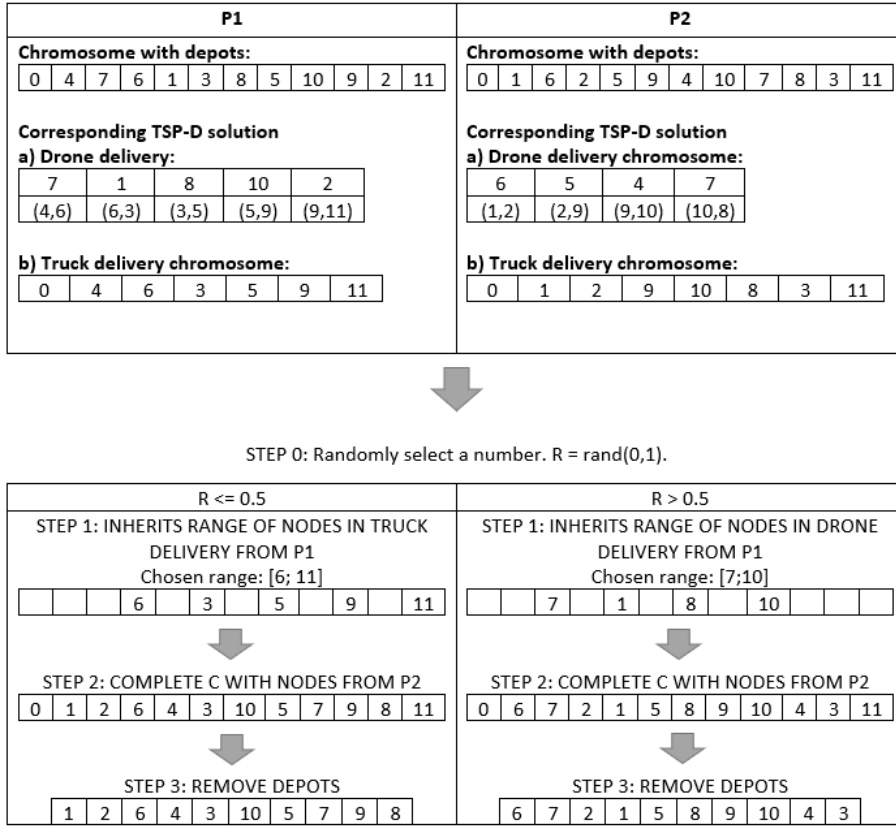


Figure 5.2: DX Crossover for TSP-D.

In detail, Algorithm 15 first takes the two parents P_1, P_2 as one of its inputs. Moreover, in line 1, it also takes into account the corresponding TSP-D solution (TD_1, DD_1) of P_1 , which was obtained during the “education” process (Line 6 of Algorithm 14). Subsequently, it defines two TSP tours, TSP_1, TSP_2 , in Line 2 by taking two parents and adding two depots to them. An empty offspring with two depots is also initialized in Line 3. In Line 4, a random number is generated to decide from which component – TD_1 or DD_1 – the algorithm will inherit. In either case, it will choose a random segment of the chosen component by generating two random cut points, a, b , with $a < b$, and copy the nodes between those cut points to C while keeping their original positions in TSP_1 (Lines 5 to 9). Finally, the remaining nodes of C are filled one by one, starting from position 1, by taking the remaining nodes of TSP_1 and copying to C while keeping their relative orders in TSP_2 (Line 10). The offspring is returned by removing two depots of C (Line 11).

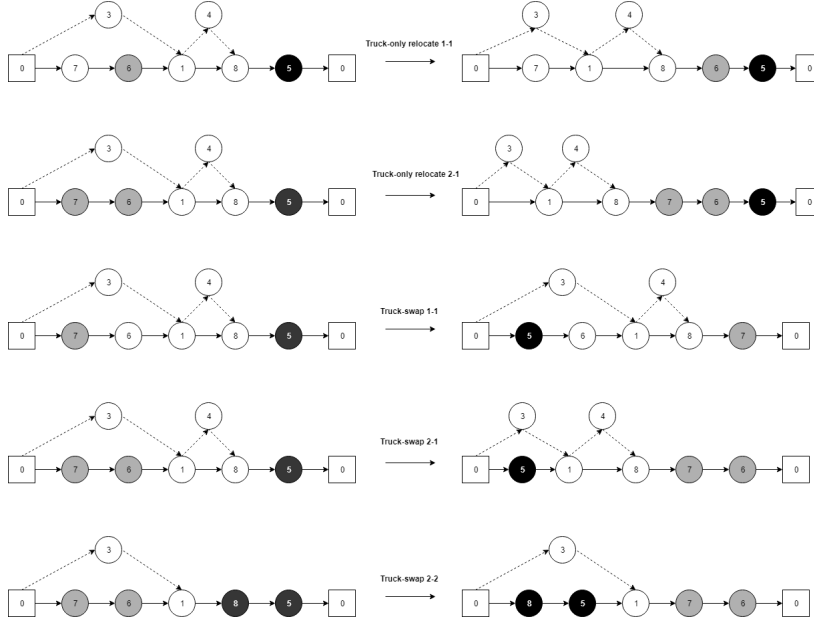
5.1.5 Education using local search

The main role of the *education* step is to improve the quality of solutions by means of the local search procedure. We design a hill-

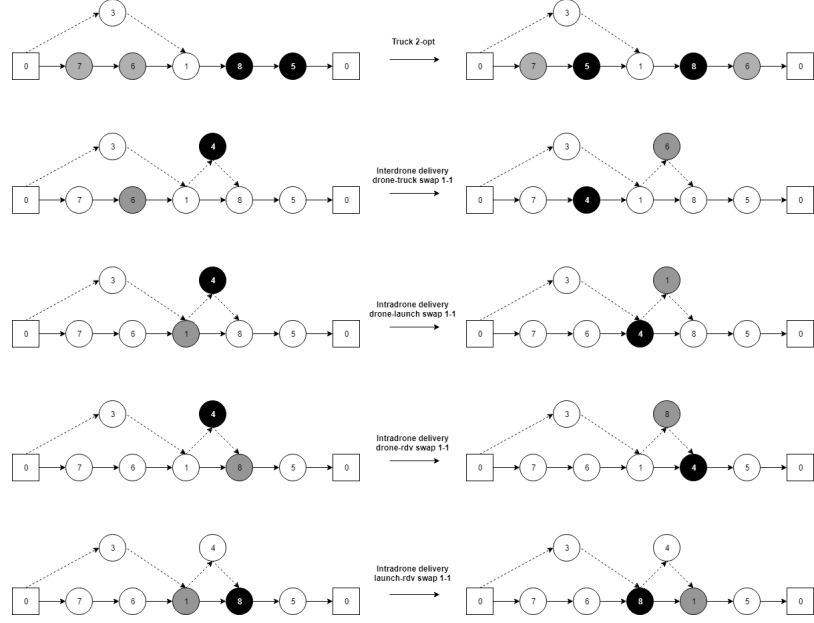
climbing and first-improvement local search for both min-cost and min-time objectives. Similar to [Vid+12], we also apply the technique proposed in [TV03] to restrict the search to the $h \times n$ closest vertices, where $h = 0.1$ is the *granular threshold*. This technique significantly reduces the computation time consumed by the education process. We define neighborhoods for the TSP-D based on a set of 16 move operators in which 3 operators – $\mathcal{N}_1$, $\mathcal{N}_{13}$ and $\mathcal{N}_{14}$ – are inherited from the work of [Ha+18b]. In each operator, the evaluation separately evaluates the move costs for the min-cost and min-time objectives. For min-cost, it is the total truck and drone costs of the affected arcs, while the total truck and drone travel times of the affected arcs are calculated in the min-time problem. Moreover, the truck and drone cumulative time and cost as well as the cost and time of all drone tuples in set $\mathcal{P}$ are pre-computed at the beginning of the HGA to effectively accelerate the algorithm.

We now describe in detail the neighbourhoods to be explored. The illustrations of these move operators are also presented in Figures 5.3, 5.4 and 5.5.

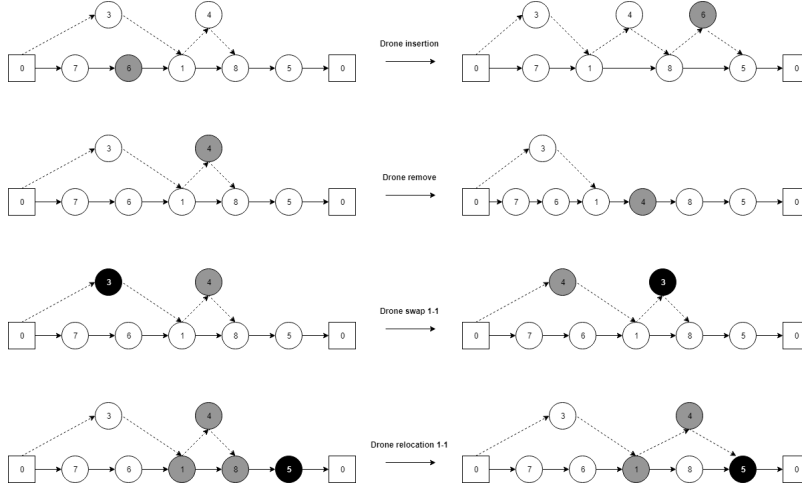
- $\mathcal{N}_1$ (Truck-only relocation 1-1): Choose random truck-only node u (the node where the drone is carried by truck), and relocate it after a node v in the truck tour.
- $\mathcal{N}_2, \mathcal{N}_3$ (Truck-only relocation 2-1): Choose two random consecutive truck-only nodes u_1, u_2 , and relocate them after a node v in the truck tour as u_1, u_2 or u_2, u_1 .
- $\mathcal{N}_4$ (Truck swap 1-1): Choose a random node u in the truck tour, and swap with another node v in the truck tour.
- $\mathcal{N}_5$ (Truck swap 2-1): Choose two random consecutive nodes u_1, u_2 in the truck tour such that u_2 does not have a drone launch or retrieval activity, and swap with another node v in the truck tour. Again, we update the corresponding drone deliveries.
- $\mathcal{N}_6$ (Truck swap 2-2): Select two random consecutive nodes u_1, u_2 in the truck tour, and swap with two other nodes v_1, v_2 in the truck tour. Drone deliveries associated with those nodes are updated.

Figure 5.3: Illustrations of $\mathcal{N}_1$ to $\mathcal{N}_6$

- $\mathcal{N}_7, \mathcal{N}_8$ (Truck 2-opt): Select two random pairs of consecutive nodes (u, x) and (v, y) in the truck tour, and relocate them as $(u, v), (x, y)$ or $(u, y), (x, v)$.
- $\mathcal{N}_9$ (Interdrone delivery drone-truck swap 1-1): Select a random drone node d , and swap it with another node u in the truck tour such that u is neither d 's launch node, rendezvous node, or the node between its launch and rendezvous.
- $\mathcal{N}_{10}$ (Intradrone delivery drone launch swap 1-1): Select a random drone 3-tuple $\langle i, j, k \rangle$ in the drone delivery list, and swap i and j .
- $\mathcal{N}_{11}$ (Intradrone delivery drone rdv swap 1-1): This is similar to the above move operator, except that we swap j and k .
- $\mathcal{N}_{12}$ (Intradrone delivery launch rdv swap 1-1): Again, it is similar to the above move operator, but instead, we swap i and k .

Figure 5.4: Illustrations of $\mathcal{N}_7$ to $\mathcal{N}_{12}$

- $\mathcal{N}_{13}$ (Drone insertion): Select a random node j such that j is either a truck-only node or the node in between a drone delivery, choose two other nodes i and k in the truck tour – i is before k – and create a new drone delivery $\langle i, j, k \rangle$. This move is only valid when no drone delivery interference exists between i and k or when we can say that there is no drone launch or retrieval between i and k .
- $\mathcal{N}_{14}$ (Drone remove): We select a random drone node j , remove the associated drone delivery, and reinsert j between two consecutive nodes i and k in the truck tour.
- $\mathcal{N}_{15}$ (Drone swap 1-1): Select two random drone deliveries $\langle i_1, j_1, k_1 \rangle$ and $\langle i_2, j_2, k_2 \rangle$, and swap j_1 and j_2 . We will therefore have two new drone deliveries: $\langle i_1, j_2, k_1 \rangle$ and $\langle i_2, j_1, k_2 \rangle$.
- $\mathcal{N}_{16}$ (Drone relocation 1-1): Select a random drone delivery $\langle i, j, k \rangle$, and choose a new launch i' and rendezvous node k' for j to have a new drone delivery $\langle i', j, k' \rangle$.

Figure 5.5: Illustrations of $\mathcal{N}_{13}$ to $\mathcal{N}_{16}$

5.1.6 Restore method

To more efficiently guide the search for good solutions, a restoration method is developed in which we use the *educated* TSP-D solution (the one that has been improved using local search) to update the existing giant tour individual. In detail, the new giant tour is constructed by reinserting drone nodes in the drone deliveries of the educated TSP-D solution to a *random position* between their launch node and rendezvous node on the truck delivery tour of that solution. After the insertion operation is finished, two depots are removed to obtain a valid giant tour individual. As a result, we have a new giant tour individual that is formed by an “educated” truck tour with drone nodes being reinserted. An illustration of this process is shown in Figure 5.6.

It is possible that our randomized restoring can generate a random solution based on the provided solution but the new solution is at least as good as the old one due to the optimality of our split procedure. This ensures the convergence of our algorithm. In the restore method, with each drone delivery $\langle i, j, k \rangle$, we have tried three ways to choose the position for reinserting drone node j : (1) the position right after node i ; (2) the position right before node k ; and (3) a random position between i and k . The experimental results show that the third option performs the best. Our guess is that random restore method could help to increase the diversification of the algorithm, thus avoid the algorithm converging too fast.

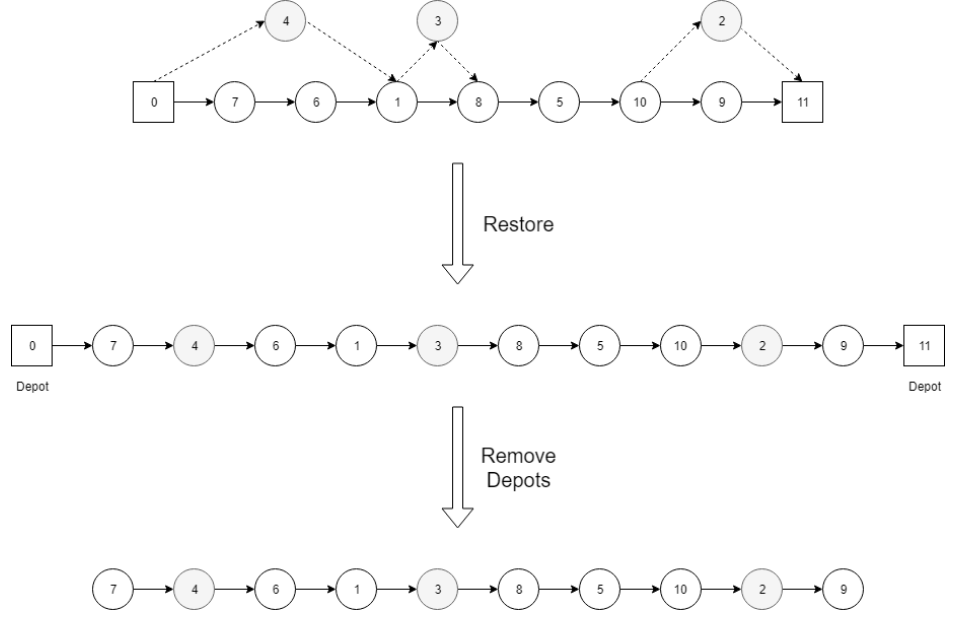


Figure 5.6: Reinsertion in restore method. Truck travels the solid lines and drone travels the dashed lines.

5.1.7 Repair mechanism

As described in Algorithm 14, when a solution becomes infeasible after the local search step, a repair mechanism is invoked to try to make it feasible with the hope that this repaired (and feasible) solution might be a high quality one. This repair method can be described as follows.

Given an TSP-D infeasible solution sol , the method firstly initializes an exponential value $exp = 1$ and record the current penalty coefficient ω as ω_0 . After that, a number of iterations is started. In each iteration, we first increase the penalty coefficient ω as follows: $\omega = \omega * 10^{exp}$. In the next step, local search method is called taking sol and the new ω value as the input. If this results in a new sol' that is feasible, the iterations are stopped. Otherwise, at the end of each iteration, we increase the value of exp by one, update sol with sol' and continue. After all iterations have been run, restore method is called to update the chromosome. Finally, the original penalty coefficient is restored $\omega = \omega_0$.

In conclusion, by gradually increase the penalty coefficient, we expect the local search method to choose moves that can fix the violated drone deliveries that currently violate the endurance constraint (since the penalty is high and therefore results in a bad objective value).

5.1.8 Population management

As an adaptation of the framework in [Vid+12], the population management mechanism in HGA remains untouched. In detail, two subpopulations are created and managed separately. They are the feasible and infeasible subpopulations. Each contains between μ to $\mu + \lambda$ individuals.

In the initialization step, $n_{initPop}$ of individuals are created by generating a set of TSP tours using a k-cheapest insertion heuristic with $k = 3$ [Ha+18b]. The choice of a heuristic-based population comes from the analysis of [MC15] and the tested result in [Ha+18b], which suggests the use of high-quality TSP tours instead of completely random ones. We obtain the giant tour chromosomes after these generation steps. These tours then pass through the *split* method to obtain the corresponding TSP-D solutions of each individual. In the next step, individuals' TSP-D solutions are processed using the *education* process to improve their qualities, and when an infeasibility occurs, they are probabilistically repaired. After that, the *restore method* is called to update individuals' giant tour chromosomes. The individuals are then added to the appropriate subpopulations based on their feasibilities.

Any subpopulation that exceeds the size of $\mu + \lambda$ is passed through a *select survivors* method in which λ individuals are discarded. The discarded ones are ones defined as "clones" or the worst individuals with respect to their biased fitnesses. Solutions are defined as clones if and only if they have the same giant tour (possibly in reversed order).

Furthermore, the penalty coefficient ω is dynamically adjusted during the search for each 100 iterations. This mechanism is necessary to guide the algorithm in two search spaces. More specifically, the penalty coefficient is increased when the search produces too many infeasible solutions (meaning that it falls too deeply into the infeasible search space) and is decreased in the opposite case. In detail, let $\mathcal{E}^{REF}$ be the targeted proportion of the feasible solution, and we then adjust the parameter ω as follows: if the naturally feasible proportion is below $\mathcal{E}^{REF} - \zeta\%$ (is higher than $\mathcal{E}^{REF} + \zeta\%$), then the penalty coefficient is increased by η_I (decreased by η_D). This means that when the feasible proportion is in the range $\mathcal{E}^{REF} \pm \zeta\%$, the coefficient remains unchanged to avoid the search jumping too quickly between regions in the search space.

When the search is not improved after $Iter_{DIV}$ iterations, the *diversification* method is called, in which we retain the best n_{best} individuals with respect to their biased fitness and generate $n_{initPop}$ new individuals as in the initialization phase. This technique is important because it creates new genetic materials for the search when the population has lost its diversification characteristic.

5.2 COMPUTATIONAL RESULTS

This section presents the computational results of the HGA, which has been implemented in C++ and compiled with the “-O3” flag. The experiments are run on a desktop computer with an Intel Core i7-6700, 3.4 GHz processor.

Because the parameters proposed in [Vid+12] have been proven to work well on many variants of VRP, we retained most of them. In detail, the default parameters of HGA are $\mu = 15$, $\lambda = 25$, $n_{initPop} = 4\mu$, $\zeta = 5\%$, $\eta_I = 1.2$, $\eta_D = 0.85$, $n_{best} = \mu/3$, $nbElite = 6$, $\mathcal{E}^{REF} = 0.3$, $n_{close} = 0.2$, $\omega = 1.0$, $Iter_{NI} = 2500$, and $Iter_{DIV} = 0.3 \times Iter_{NI}$.

For the TSP-D parameters, we used the parameters proposed in [MC15]: the truck speed and drone speed were set to 40 km/h, and the drone endurance ϵ was 20 minutes. The time required to launch and retrieve the drone (s_L and s_R) were both set to 1 minute.

As described in Section 5.1.1, there are two types of infeasibilities in a TSP-D: truck travel time and drone travel time constraint violations. From those constraint violations, we define three levels of relaxations.

- **RelaxAll:** We accept both types of infeasibilities.
- **RelaxTruck:** We only accept the truck travel time constraint violation in infeasible solutions.
- **RelaxDrone:** We only accept the drone travel time constraint violation in infeasible solutions.

The impacts of these different types of relaxations are investigated in Section 5.2.3. By default, RelaxAll is used. The default selection for the crossover is DX, which is the best performing crossover as tested in Section 5.2.2.

The following sections are organized as follows. We first evaluate the performance of HGA with different instance sets and compare with the existing methods. Next, an analysis of the impacts of different crossovers is presented. Finally, we evaluate the sensitivity of each component in HGA.

5.2.1 Performance on different instance sets

In this section, we test HGA on three sets of instances: (1) 72 min-time instances of 10 customers from [MC15]; (2) 60 instances of 50 and 100 customers from [Ha+18b] under both min-time and min-cost objective functions; and (3) 24 instances with various size introduced in the recent work of [FP18]. For the HGA, we collected its best found solutions and computed the objective function’s value of solutions on average over 10 runs. Current best methods - GRASP in [Ha+18b], HGVNS in [FP18] and different approaches proposed in [MC15] -

were selected to compare with HGA. As mentioned before, the standard version of HGA with DX and RelaxAll was used in this experiment. The results for Instance Sets (1), (2) and (3) are presented in Tables 5.1, 5.2, 5.3 and 5.4.

Results for instances from Murray and Chu [MC15]

In Table 5.1, we compare HGA with the best results found by [MC15] and GRASP [Ha+18b] among 36 instances of Set 1 with two settings of drone endurance (20 and 40 minutes). The ϵ column shows the drone endurance in minutes. Column *HGA* represents the best found solutions while column $\overline{HGA}$ reports the average values among 10 runs of our new algorithm. The values in bold text imply the best result found among the three approaches. Overall, HGA was able to improve the existing best found solutions in 9 tests and obtained results as good as the best ones in 60 tests. Column $\overline{HGA}$ shows the stability of HGA in this context when the solutions over 10 runs generally reach the best ones in all instances but two. The results also demonstrate a dominance of our HGA over GRASP in terms of solutions' quality. However, HGA is in general slower than GRASP.

Results for instances from Ha et al. [Ha+18b]

Tables 5.2 and 5.3 report the comparisons of objective value and average run time (in minutes) between HGA and GRASP in [Ha+18b] on Instance Set 2. We collected the average value (Column " $\overline{HGA}$ ") and best solution of HGA found among repeated runs (Column "*HGA*") and its average run time in minutes (Column " T_{HGA} "). The corresponding values of GRASP are reported in Column " $\overline{GRASP}$ ", "*GRASP*", and " T_{GRASP} ". Column "Change(%)" calculates the percentage change between best found objective values of HGA and GRASP. A negative value indicates an improvement of our new method in comparison to GRASP. With respect to this comparison, HGA shows improvements in terms of solutions' quality in both min-cost and min-time objectives.

In detail, for min-cost TSP-D (Table 5.2), the average objective values of solutions of HGA are even better than those of the best found solutions of GRASP on most instances (see Columns " $\overline{HGA}$ " and " $\overline{GRASP}$ "). The proposed algorithm can significantly improve existing best known solutions by 6.40% and 15.10% on average (up to nearly 15% and 20%) for 50- and 100-customer instances, respectively. We can observe that the algorithm performs better in large instances (i.e., 100-customer instances). However, it is worth mentioning that GRASP performs better on two instances D5 and D6. Regarding run time, HGA is 1.5 to 2 times slower than GRASP due to its more complex design. This result is acceptable since it still can deliver signifi-

cantly better results in less than 1 minute for 50-customer instances and less than 5 minutes for 100-customer instances.

For min-time TSP-D (Table 5.3), HGA can also improve the existing best known solutions found by GRASP on all instances but not as significantly as in min-cost TSP-D. In detail, the improvements are 2.39% and 4.05% on average (and up to nearly 6% and 8%) for 50- and 100-customer instances, respectively. Again, HGA performs approximately 1.5 times slower than GRASP but can still deliver better solutions in less than 1 minute and 5 minutes for 50- and 100-customer instances, respectively.

Results for instances from Freitas and Penna [FP18]

We report the comparison between HGA and HGVNS proposed by [FP18] on Instance Set 3 under Min-time objective in Table 5.4. In this table, Column “TSP*” is the optimal TSP value obtained by Concorde [App+06], Columns “HGA” and “ $\overline{HGA}$ ”, respectively, are the best and average results among 10 repeated runs. Columns “ Gap_{HGA} ” and “ $\overline{Gap_{HGA}}$ ” represent the best and average gaps between HGA and optimal TSP while Column “ T_{HGA} ” reports the average running time of HGA. Similarly, Columns “HGVNS”, “ $\overline{HGVNS}$ ”, “ Gap_{HGVNS} ”, “ $\overline{Gap_{HGVNS}}$ ” and T_{HGVNS} represent the best and average value and gap as well as the running time of HGVNS.

In overall, among 24 instances, HGA can improve existing best known solution found by HGVNS in 16 instances, performs worse than HGVNS in 7 instances and a draw in one instance (eil51). In average, HGA performs approximately 5% better than HGVNS and up to 17.17% in kroC100. Regarding the computational cost, HGA is about 2 times slower than HGVNS but the running time between two algorithms is not fairly compared since they are run on two different machine configurations (HGVNS runs on a faster machine Intel Core i7 Processor 3.6 GHz). In detail, HGA mostly performs better than HGVNS on instances where solutions contain many drone deliveries (such as the kro instances) while HGVNS, on the other hand, can find better solutions among instances where solutions contain very few drone deliveries. This can be explained by the fact that HGVNS explores the search space from the optimal TSP solutions, which are very closed to the final TSP-D solutions.

Instance	ϵ	Murray et al.	GRASP	HGA	$\overline{HGA}$	Instance	ϵ	Murray et al.	GRASP	HGA	$\overline{HGA}$
437v1	20	56.468	57.446	56.468	56.468	440v7	20	49.996	49.776	49.422	49.422
437v1	40	50.573	50.573	50.573	50.573	440v7	40	49.204	49.204	49.204	49.204
437v2	20	53.207	53.207	53.207	53.207	440v8	20	62.796	62.700	62.576	62.576
437v2	40	47.311	47.311	47.311	47.311	440v8	40	62.270	62.004	62.004	62.004
437v3	20	53.687	54.664	53.687	53.687	440v9	20	42.799	42.566	42.533	42.533
437v3	40	53.687	53.687	53.687	53.687	440v9	40	42.799	42.566	42.533	42.533
437v4	20	67.464	67.464	67.464	67.464	440v10	20	43.076	43.076	43.076	43.076
437v4	40	66.487	66.487	66.487	66.487	440v10	40	43.076	43.076	43.076	43.076
437v5	20	50.551	50.551	50.551	50.551	440v11	20	49.204	49.204	49.204	49.204
437v5	40	45.835	44.835	44.835	44.835	440v11	40	49.204	49.204	49.204	49.204
437v6	20	45.176	47.601	47.311	47.311	440v12	20	62.004	62.004	62.004	62.004
437v6	40	45.863	43.602	43.602	43.602	440v12	40	62.004	62.004	62.004	62.004
437v7	20	49.581	49.581	49.581	49.581	443v1	20	69.586	69.586	69.586	69.586
437v7	40	46.621	46.621	46.621	46.621	443v1	40	55.493	55.493	55.493	55.493
437v8	20	62.381	62.381	62.381	62.381	443v2	20	72.146	72.146	72.146	72.146
437v8	40	59.776	59.416	59.416	59.416	443v2	40	58.053	58.053	58.053	58.053
437v9	20	45.985	42.945	42.416	42.416	443v3	20	77.344	77.344	77.344	77.344
437v9	40	42.416	42.416	42.416	42.416	443v3	40	69.175	68.431	68.431	68.431
437v10	20	42.416	41.729	41.729	41.729	443v4	20	90.144	90.144	90.144	90.144
437v10	40	41.729	41.729	41.729	41.729	443v4	40	82.700	83.700	82.700	82.700
437v11	20	42.896	42.896	42.896	42.896	443v5	20	55.493	58.210	54.973	55.077
437v11	40	42.896	42.896	42.896	42.896	443v5	40	53.447	51.929	51.929	51.929
437v12	20	56.696	56.425	56.273	56.273	443v6	20	58.053	58.053	55.209	55.209
437v12	40	55.696	55.696	55.696	55.696	443v6	40	52.329	52.329	52.329	52.329
440v1	20	49.430	50.164	49.430	49.430	443v7	20	64.409	65.523	65.523	65.523
440v1	40	46.886	46.886	46.886	46.886	443v7	40	60.743	60.743	60.743	60.743
440v2	20	50.708	51.828	50.708	50.708	443v8	20	77.209	78.323	78.323	78.323
440v2	40	46.423	46.423	46.423	46.423	443v8	40	73.967	72.967	72.967	72.967
440v3	20	56.102	58.502	56.102	56.102	443v9	20	49.049	45.931	45.931	45.931
440v3	40	53.933	53.933	53.933	53.933	443v9	40	47.250	45.931	45.931	45.931
440v4	20	69.902	73.091	69.902	69.902	443v10	20	47.935	46.935	46.935	46.935
440v4	40	68.397	68.397	68.397	68.397	443v10	40	47.935	46.935	46.935	46.935
440v5	20	43.533	44.624	43.533	43.533	443v11	20	57.382	56.395	56.395	56.395
440v5	40	43.533	43.533	43.533	43.533	443v11	40	56.395	56.395	56.395	56.395
440v6	20	44.076	44.122	43.949	43.949	443v12	20	69.195	69.195	69.195	69.195
440v6	40	44.076	43.944	43.810	43.853	443v12	40	69.195	69.195	69.195	69.195

Table 5.1: Comparison of Instance Set 1 under Min-time objective

5.2.2 Performance under different crossovers

We evaluate the performance of HGA when using our proposed crossover versus four classical crossovers [Pot96] (OX, PMX, OBX, and PBX) in Tables 5.5 under two objectives with Instance Set 2 mentioned in the above section. Again, HGA was repeatedly run 10 times for each choice of crossover, and we have conducted 6000 tests in total. For each crossover, we report the average percentage gap with the best found solution (regardless of crossover), the run time in minutes (Column “ T ”), the standard deviation (Column “sd”) and the geometric mean value (row “Mean”). Furthermore, a comparison of the

Instance	$GRASP$	$\overline{GRASP}$	T_{GRASP} (min)	HGA	$\overline{HGA}$	Change(%)	T_{HGA} (min)	Instance	$GRASP$	$\overline{GRASP}$	T_{GRASP} (min)	HGA	$\overline{HGA}$	Change(%)	T_{HGA} (min)
B1	1372.82	1413.24	0.27	1225.78	1239.85	-10.71	0.5	E1	2206.53	2255.99	2.28	1775.1	1802.47	-19.55	3.47
B2	1491.3	1513.98	0.26	1381.89	1402.98	-7.34	0.38	E2	2210.61	2273.09	2.28	1795.03	1830.95	-18.80	3.66
B3	1503.78	1521.67	0.28	1357.17	1370.82	-9.75	0.45	E3	2248.16	2312.76	2.48	1818.16	1861.59	-19.13	3.5
B4	1396.17	1426.2	0.27	1282.16	1292.87	-8.17	0.49	E4	2179.06	2223.97	2.97	1776.58	1822.36	-18.47	3.68
B5	1457.91	1500.9	0.31	1351.37	1357.61	-7.31	0.4	E5	2286.16	2360.3	2.87	1866.22	1899.25	-18.37	3.55
B6	1316.08	1353.76	0.27	1159.79	1174.30	-11.88	0.44	E6	2244.62	2313.86	3.26	1795.17	1831.23	-20.02	4.12
B7	1370.05	1399.71	0.24	1308.25	1322.70	-4.51	0.42	E7	2249.09	2313.67	3.18	1892.41	1923.18	-15.86	3.46
B8	1484.93	1517.23	0.25	1255.61	1275.61	-15.44	0.62	E8	2220.88	2272.55	3.15	1813.73	1831.46	-18.33	4.3
B9	1442.09	1468.86	0.28	1355.32	1365.53	-6.02	0.52	E9	2279.91	2326.29	2.87	1882.71	1901.76	-17.42	5.01
B10	1392.54	1429.57	0.25	1252.9	1257.48	-10.03	0.47	E10	2324.74	2384.52	3.41	1870.55	1932.75	-19.54	4.07
C1	2870.41	2935.87	0.21	2679.1	2703.14	-6.66	0.29	F1	4569.83	4648.2	1.85	3766.63	3854.56	-17.58	3.24
C2	2804.47	2868.67	0.26	2750.74	2755.04	-1.92	0.36	F2	4186.76	4318.78	2.38	3469.54	3575.90	-17.13	3.95
C3	3087.55	3185.09	0.16	2932.78	2952.48	-5.01	0.32	F3	4414.38	4563.64	2.45	3751.07	3891.27	-15.03	3.02
C4	2844.1	2916.86	0.20	2655.25	2676.82	-6.64	0.67	F4	4499.09	4600.27	2.14	3818.62	3862.57	-15.12	3.27
C5	3323.92	3367.34	0.19	3133.69	3156.88	-5.72	0.36	F5	4381.37	4597.32	2.66	3756.78	3807.86	-14.26	3.28
C6	3433.99	3472.39	0.19	3238.92	3268.49	-5.68	0.4	F6	4032.9	4171.8	2.63	3465.56	3560.66	-14.07	2.54
C7	3001.13	3047.71	0.21	2681.01	2738.71	-10.67	0.91	F7	4076.31	4213.52	2.84	3601.78	3660.90	-11.64	4.83
C8	3481.17	3557.99	0.22	3250.19	3259.22	-6.64	0.91	F8	4491.2	4597.9	2.77	3803.14	3930.37	-15.32	3.91
C9	3267.23	3306.38	0.19	3032.73	3056.08	-7.18	0.58	F9	4388.91	4463.39	2.55	3873.51	3904.18	-11.74	4.07
C10	3291.2	3356.29	0.23	3082.07	3117.05	-6.35	0.61	F10	4173.64	4567.84	2.57	3837.47	3895.06	-8.05	3.1
D1	4159.39	4389.24	0.21	3927.97	3928.12	-5.56	0.43	G1	5947.97	6148.5	1.94	5084.56	5312.36	-14.52	3.08
D2	4275.46	4334.4	0.19	4097	4113.47	-4.17	0.58	G2	5882.97	5987.64	2.63	5198.89	5234.67	-11.63	3.16
D3	4085.71	4191.08	0.18	3846.84	3861.61	-5.85	0.39	G3	6074.57	6138.94	2.82	5063.87	5116.38	-16.64	4.08
D4	4612.46	4714.62	0.21	4334.32	4334.32	-6.03	0.31	G4	6458.96	6632.14	2.39	5542.19	5703.23	-14.19	4.1
D5	4717.67	4793.39	0.20	4569.16	4584.22	-3.15	0.65	G5	6198.95	6329.25	2.59	5496.77	5566.09	-11.33	2.94
D6	4405.02	4485.87	0.20	4384.00	4385.85	-0.48	0.77	G6	6049.34	6343.26	2.95	5377.51	5463.62	-11.11	2.67
D7	4749.57	4796.23	0.25	4634.53	4657.77	-2.42	0.73	G7	5889.08	6023.11	2.85	5318.17	5396.12	-9.69	4.98
D8	4143.03	4287.87	0.20	3911.94	3963.02	-5.58	0.64	G8	5599.55	5871.96	2.62	5112.58	5246.50	-8.70	5.03
D9	4653.73	4688.16	0.22	4469.78	4490.67	-3.95	0.66	G9	6050.8	6254.5	3.08	4996.42	5187.45	-17.43	4.3
D10	4260.6	4301.83	0.20	4208.93	4232.38	-1.21	0.95	G10	6249.69	6534.13	2.70	5473.91	5598.25	-12.41	3.77
Mean			0.22			-6.40	0.51				2.65			-15.10	3.68

Table 5.2: Comparison with GRASP under Min-cost objective - Instance set 2

Instance	GRASP	$\overline{GRASP}$	T_{GRASP} (min)	HGA	$\overline{HGA}$	Change(%)	T_{HGA} (min)	Instance	GRASP	$\overline{GRASP}$	T_{GRASP} (min)	HGA	$\overline{HGA}$	Change(%)	T_{HGA} (min)
B1	120.68	121.69	0.45	115.65	116.43	-4.17	0.76	E1	188.58	192.08	5.45	187.67	188.32	-0.48	3.6
B2	118.53	119.46	0.48	118.39	118.39	-0.12	0.33	E2	190.55	192.88	5.71	187.21	188.01	-1.75	5.6
B3	119.7	120.25	0.52	116.21	116.39	-2.92	0.57	E3	189.05	192.83	5.65	188.09	188.89	-0.51	4.58
B4	123.02	124.7	0.36	118.71	119.26	-3.50	0.47	E4	188.61	191.27	4.54	186.23	186.99	-1.26	4.69
B5	119.46	120.77	0.48	115.78	115.91	-3.08	0.58	E5	190.47	193.61	4.34	187.71	188.26	-1.45	4.06
B6	119.54	121.46	0.39	114.31	115.46	-4.38	0.88	E6	190.32	193.86	4.10	189.16	189.44	-0.61	4.84
B7	118.54	121.02	0.33	115.52	115.63	-2.55	0.62	E7	191.51	194.41	4.33	190.39	190.89	-0.58	3.84
B8	119.36	119.99	0.35	117.9	118.04	-1.22	0.78	E8	190.47	193.74	3.86	189.02	189.54	-0.76	4.22
B9	118.26	119.86	0.42	117.64	117.72	-0.52	0.39	E9	191.12	193.7	4.31	189.76	189.94	-0.71	4
B10	119.8	121.27	0.37	117.38	117.70	-2.02	0.6	E10	189.71	193.28	4.17	189.45	189.91	-0.14	3.4
C1	220.63	222.6	0.27	215.07	215.37	-2.52	0.6	F1	341.68	344.98	2.65	322.94	326.10	-5.48	5.73
C2	210.39	211.14	0.41	209.23	210.11	-0.55	0.53	F2	325.7	330.63	2.87	308.74	310.89	-5.21	5.24
C3	214.61	215.31	0.28	212.02	212.22	-1.21	0.38	F3	336.35	340.92	3.88	309.67	313.55	-7.93	5.61
C4	225.15	225.47	0.25	212.08	213.27	-5.81	0.6	F4	326.79	334.69	1.98	311.37	314.96	-4.72	6.06
C5	226.36	233.97	0.32	223.06	224.57	-1.46	0.48	F5	335.88	344.61	2.04	314.82	317.83	-6.27	6.57
C6	240.37	242.22	0.27	234.01	235.56	-2.05	0.31	F6	309.71	319.22	2.23	294.38	297.47	-4.95	4.7
C7	227.73	229.56	0.20	222.27	223.40	-2.40	0.51	F7	317.86	330.83	1.67	311.41	316.15	-2.03	4.92
C8	242.19	245.12	0.37	234.26	237.53	-3.27	0.46	F8	345.44	350.75	1.96	323.74	326.40	-6.28	5.21
C9	237.98	241.07	0.26	226.01	227.43	-5.03	0.68	F9	339.53	342.41	1.83	315.56	318.47	-7.06	4.66
C10	230.03	235.47	0.33	226.17	226.17	-1.68	0.48	F10	332.05	340.35	1.87	312.7	315.13	-5.83	3.94
D1	315.8	318.9	0.31	306.39	307.09	-2.98	0.61	G1	437.48	450.23	1.87	417.92	425.19	-4.47	4.45
D2	317.15	322.85	0.29	313.93	315.64	-1.02	0.57	G2	415.32	424.88	2.67	389.64	390.14	-6.18	2.4
D3	300.4	303.26	0.29	295.86	297.54	-1.51	0.6	G3	446.6	454.98	2.45	411.47	415.14	-7.87	4.9
D4	333.47	336.7	0.31	323.72	324.60	-2.92	0.56	G4	449.68	465.83	1.83	433.09	435.56	-3.69	4.67
D5	324.68	326.46	0.25	321.46	321.83	-0.99	0.4	G5	434.6	446.73	1.67	421.05	422.49	-3.12	4.48
D6	315.16	317.73	0.28	313.21	313.65	-0.62	0.49	G6	450.28	462	1.73	415.46	420.84	-7.73	5.51
D7	329.31	330.24	0.33	316.65	317.83	-3.84	0.32	G7	420	439.62	1.42	409.31	412.14	-2.55	5.21
D8	306.28	312.12	0.31	293.76	296.51	-4.09	0.58	G8	442.67	453.19	1.71	406.51	407.89	-8.17	5.08
D9	326.09	331.31	0.27	317.85	318.31	-2.53	0.41	G9	456.78	469.49	1.32	428.16	435.75	-6.27	5.91
D10	306.1	309.54	0.29	305.51	305.54	-0.19	0.41	G10	460.89	470.44	2.15	426.82	430.94	-7.39	5.4
Mean			0.33			-2.39	0.52				2.66			-4.05	4.51

Table 5.3: Comparison with GRASP under Min-time objective - Instance set 2

Instance	TSP*	HGA	$Gap_{HGA}(\%)$	$\overline{HGA}$	$\overline{Gap_{HGA}}(\%)$	$Th_{GA}(s)$	HGVNS	$Gap_{HGVNS}(\%)$	$\overline{HGVNS}$	$\overline{Gap_{HGVNS}}(\%)$	$Th_{GVNS}(s)$
berlin52	239.75	198.00	-17.41	199.80	-16.66	14.15	210.03	-12.39	220.23	-8.14	6.50
bier127	3665.60	3499.11	-4.54	3506.42	-4.34	64.06	3456.80	-5.70	3587.88	-2.12	53.69
ch130	187.83	182.86	-2.64	182.86	-2.64	76.12	178.16	-5.15	180.40	-3.95	44.13
dl98	463.45	460.16	-0.71	461.16	-0.50	114.00	461.83	-0.35	461.83	-0.35	67.69
eil51	13.45	13.45	0.00	13.45	0.00	10.51	13.45	0.00	13.68	1.71	11.57
eil76	16.90	16.90	0.00	16.90	0.00	26.71	16.35	-3.25	16.68	-1.30	27.14
kroA100	661.30	539.91	-18.36	541.36	-18.14	97.87	587.80	-11.11	609.71	-7.80	30.95
kroA150	822.60	688.35	-16.32	693.612	-15.68	145.16	764.42	-7.07	780.93	-5.07	40.95
kroA200	922.05	806.87	-12.49	820.86	-10.97	169.53	870.65	-5.57	873.99	-5.21	46.53
kroB150	823.65	656.36	-20.31	676.11	-17.91	146.29	763.15	-7.35	773.72	-6.06	50.20
kroB200	917.95	799.64	-12.89	801.36	-12.70	152.21	835.43	-8.99	838.40	-8.67	31.94
kroC100	662.30	544.68	-17.76	547.38	-17.35	79.22	658.38	-0.59	660.93	-0.21	36.63
kroD100	661.00	544.88	-17.57	547.22	-17.21	65.43	666.45	-8.25	652.34	-1.31	40.15
kroE100	690.35	576.97	-16.42	581.86	-15.71	69.41	651.31	-5.65	659.48	-4.47	48.57
lin105	420.60	377.95	-10.14	381.69	-9.25	90.89	378.25	-10.07	380.43	-9.55	40.27
pr107	1222.50	1032.64	-15.53	1038.11	-15.08	79.05	1204.42	-1.48	1224.35	0.15	32.54
pr124	1687.25	1615.88	-4.23	1618.10	-4.10	46.67	1653.80	-1.98	1996.62	18.34	25.45
pr136	2762.00	2397.25	-13.21	2474.30	-10.42	141.84	2642.00	-4.34	2789.00	0.98	44.50
pr144	1688.75	1675.75	-0.77	1675.75	-0.77	175.92	1666.25	-1.33	1675.75	-0.77	43.33
pr152	2123.95	1969.80	-7.26	1973.67	-7.08	119.13	2114.04	-0.47	2128.53	0.22	61.29
rat99	37.45	37.45	0.00	37.45	0.00	54.71	37.15	-0.80	37.33	-0.32	35.37
rat195	71.50	71.50	0.00	71.50	0.00	168.83	71.40	-0.14	71.93	0.60	44.89
rd100	246.70	217.00	-12.04	219.42	-11.06	85.14	240.46	-2.53	243.84	-1.16	33.87
sf70	21.00	21.00	0.00	21.00	0.00	21.91	20.50	-2.38	21.00	0.00	3.85
Average			-9.19		-8.65	92.28		-4.46		-3.69	37.58

Table 5.4: Comparison with HGVNS in [FP18] under min-time objective.

convergence of these crossovers in both objectives is presented in Figures 5.7 and 5.8, where the Y-axis shows the averaged percentage gap with the best found solutions, and the X-axis contains the maximum number of iterations over which an improvement could be made.

Overall, DX delivers the best value among other crossovers in terms of percentage gap. For min-cost, DX is approximately 18%, 5.7%, 283%, and 16.5% better than OX, PMX, OBX, and PBX, respectively. For min-time, that superiority is approximately 26.5%, 10.2%, 283%, and 46.9%. As can be seen, OBX performs worst among the crossovers, possibly due to its design, for which only a random number of separated nodes is copied from the parent. This causes the OBX to have a smaller chance of transmitting “good” materials from its parent such as good drone deliveries or good, complete truck deliveries. The performances of OX and PMX, on the other hand, were much closer to those of DX, especially for PMX in the min-cost problem, being only 5.7% inferior. This result is because OX and PMX are both designed to copy a random subsequence of the parent to the children, thus having a high chance of transmitting “good” materials such as complete drone or truck deliveries from parent to offspring.

With respect to run time, OBX performs nearly 1.5 to 2 times faster than other crossovers. However, due to its poor performance, this fast run time is not valuable. Other crossovers deliver similar run times – less than 2 minutes on average – which is an acceptable value.

When considering standard deviation, DX, OX, PMX and PBX perform stably, the values of which are mostly less than 0.5% and no more than 1%, while OBX shows its instability in delivering values that are more than 0.5% and up to nearly 1.3%.

Finally, from Figures 5.7 and 5.8, we can see a similar pattern in the convergences of all the crossovers. They all converge quickly in the first 5000 iterations.

	DX			OX			PMX			OBX			PBX		
	Gap	T (min)	sd	Gap	T (min)	sd	Gap	T (min)	sd	Gap	T (min)	sd	Gap	T (min)	sd
Min-Cost	1.39	1.37	0.86	1.64	1.44	0.87	1.47	1.31	0.92	5.33	0.87	1.28	1.62	1.53	0.95
Min-Time	0.49	1.55	0.33	0.62	1.50	0.41	0.54	1.48	0.30	1.88	0.96	0.51	0.72	1.73	0.40

Table 5.5: Crossover performance comparison - Min-cost and Min-time objective

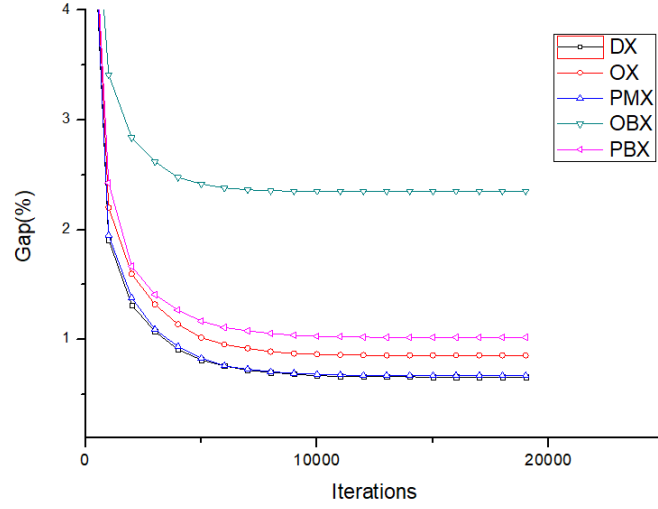


Figure 5.7: Crossovers' performance - Min-Time objective.

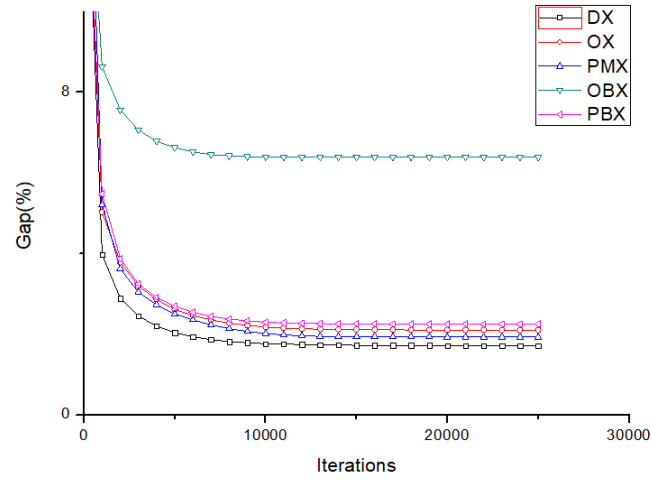


Figure 5.8: Crossovers' performance - Min-Cost objective.

5.2.3 Sensitivity analyses

This section provides analyses, as shown in Table 5.6, of the impact of the key components of HGA as based on the measurement of percentage gap on average of solutions over 10 runs to the best known solutions (BKS). The investigated components are the restore method, repair mechanism, relaxation choice, infeasibility of solutions, and diversity contribution. We adapted the standard setting (crossover DX is used with parameters mentioned at the beginning of Section 6.5) and modified each of the key components to test their impact. In detail, we have the following.

- **No INF:** Instead of relaxing the endurance constraint on truck and drone travel times, we insist that it hold. Therefore, no infeasible solution is allowed.
- **No DIV:** We do not count the diversity contribution (setting it to 0) during the calculation of biased fitness.
- **No REPAIR:** We do not use a repair method in HGA.
- **No RESTORE:** We do not use a restore method in HGA.
- **RelaxTruck:** We only allow for infeasible solutions in which the endurance constraint is violated by truck travel times but not the drone's time.
- **RelaxDrone:** We only allow for infeasible solutions in which the endurance constraint is violated by the drone's travel times but not the truck's time.

The experiment results show that HGA is indeed sensitive to its parameters (infeasibility, diversity contribution, repair, and restore method) in such a way that any change to those values negatively impact the algorithm's performance. However, those negative changes do not share the same impact. In detail, eliminating the role of the restore method (**No RESTORE**) strongly reduces the performance of HGA, which proves the necessity of this problem-specific component to the general framework in order to efficiently solve the TSP-D problem.

The infeasible solutions management, diversity contribution and repair mechanism (**No INF**, **No DIV** and **No REPAIR**) also contribute to the performance of HGA, notably the **No INF** and **No DIV**, where the increment compared to the standard gap exceeds 50%. This result proves the effectiveness of using both feasible and infeasible solutions as well as the importance of a diversity control mechanism to avoid the search becoming stuck too quickly in the local minima.

Regarding the relaxation selection (**RelaxTruck**, **RelaxDrone**), we can observe the negative impact of these choices on the performance of HGA for both objectives. However, this impact is not the same for each of the objective types. In detail, while the min-cost objective performs well when the drone travel time constraint is relaxed (**RelaxDrone**), the min-cost objective delivers a gap close to the standard gap when the truck travel time constraint is relaxed (**RelaxTruck**). This phenomenon could be explained as follows.

In the min-cost problem, the longer the distance (or time) the truck travels between launch and rendezvous nodes during a drone delivery is, the greater the impact on the travel cost it would receive, as the transportation cost of the truck is many times larger than that of the drone. Hence, with the **RelaxTruck** option for which the truck travel time constraint is relaxed and the drone travel time constraint

is imposed, the truck would be less likely to receive this relaxation advantage because of its high transportation cost per unit distance. On the other hand, when the drone travel time constraint is not enforced (**RelaxDrone**), the algorithm could have infeasible solutions in which the drone will take the longer arcs (because of its small transportation cost). These solutions then have more opportunities to be repaired to become a high quality solution.

In the min-time problem, as analysed in [Ha+18b], the frequency at which the drone is used is much less than that in the min-cost problem. Therefore, min-time solution quality depends more on truck tour quality. Hence, when the truck travel time constraint is relaxed (**RelaxTruck**), we could have infeasible solutions in which the drone arrives at the rendezvous node before the truck. This is the ideal situation for the truck as it could immediately proceed to the next customer location or prepare a parcel for the next launch without waiting for the drone to arrive [MC15]. This could shorten the truck's wait time and possibly lead to a good truck tour. Thus, along with the repair method, these kinds of infeasible solutions would have more chances to be repaired to become a high quality solution. On the other hand, the opposite fact occurs when the drone travel time constraint is relaxed (**RelaxDrone**), meaning that the truck is more likely to wait for the drone at the rendezvous node, therefore having less chance of obtaining good solutions.

	No INF	No DIV	No REPAIR	No RESTORE	RelaxTruck	RelaxDrone	Standard
Min-cost	2.39	2.19	1.34	5.42	2.19	1.30	1.29
Min-time	0.84	0.94	0.58	1.39	0.64	0.79	0.53

Table 5.6: Sensivity analysis of key components

5.3 CONCLUSION

In this paper, we presented a new hybrid genetic algorithm – HGA – to effectively solve the TSP-D under both min-cost and min-time objectives. Our algorithm includes new problem-tailored components such as local searches, crossover, restore method and penalized mechanism to effectively guide the search for good solutions. Computational experiments show that HGA outperforms two existing methods in terms of solution quality. Our method can also improve a number of the best known solutions found in the literature. An extensive analysis was carried out to demonstrate the importance of the new components to the overall performance of HGA. In future work, we intend to develop an efficient exact method to better investigate the performance of the algorithm. Also, we would like to test HGA on other variants of the TSP-D such as the version with multiple trucks and multiple drones under both objectives.

THE MULTI-VISITS TRAVELING SALESMAN PROBLEM WITH DRONE

6.1 PROBLEM DESCRIPTION

This section presents the description of the TSP-mVD. The problem is an extension of the TSP-D proposed in [MC15] where instead of restricting the number of customers that drone can visit to one, we allow the drone to deliver parcels to multiple customers before returning to the truck. More formally, given a graph $G = (V, A)$, $V = \{0, 1, 2, \dots, n+1\}$ is the set of depot and customer locations, A is the set of arcs in the graph G . Nodes 0 and $n+1$ are the depot and its duplication. The main task is to deliver parcels to a set $N = \{1, \dots, n\}$ of customers by using a truck and a drone under the following conditions:

- Each customer is only visited once by either a truck or a drone. However, not all customers could be visited by the drone due to its limitation in lifting power, endurance, and ability to handle large-size parcels. Therefore, we also denote $N_D \subseteq N$ the set of nodes that can be serviced by drone.
- Two vehicles (truck and drone) have to start from and return to the depot. The drone could either be launched from/returned to the depot or from/to a truck. When not in service, the drone is carried (and recharged) by the truck.
- When a customer is visited by a truck, we have a *truck delivery*, otherwise we have a *drone delivery*.
- A drone delivery is represented as a 3-tuple $\langle i, J, k \rangle$ where:
 - i is the launch node where the truck releases the drone for delivering. We denote s_L the time required for the preparation of launching the drone.
 - $J = 1, \dots, m, m \leq q, J \subseteq N_D$ is a sequence of m nodes representing the customer locations that the drone will service before returning to the truck. Moreover, the cardinality of J cannot exceed the maximum number of drone visits allowed per sortie denoted by q . The value of q is predefined based on technical characteristics of drone such as: design, practical payload, and flight endurance.
 - k is the rendezvous node where the drone will return to the truck at a customer location. Let s_R be the time required for the truck driver to retrieve the drone.

- At the rendezvous node, two vehicles are required to wait for each other. Furthermore, while waiting for the truck, the drone is assumed to be in a constant flight.
- A drone has an endurance – its maximum operating time without recharging – which is denoted as ϵ . With respect to this value, a drone delivery must satisfy the following travel time conditions:
 - The travel time of truck from the launch node to the rendezvous node ($\tau_{i \rightarrow k}$) and its time required for retrieving and/or relaunching the drone at this location (*recoverTime*) cannot exceed the drone endurance:

$$\tau_{i \rightarrow k} + \text{recoverTime}_{truck} \leq \epsilon$$

- The travel time of drone from the launch node to its serviced customers and then to the rendezvous node ($\tau'_{i,j} + \tau'_{j,k} + \tau'_j$ where τ'_j is the time the drone travel among its customers) and its time required for being retrieved by the truck cannot exceed the drone endurance:

$$\tau'_{i,j} + \tau'_{j,k} + \tau'_j + \text{recoverTime}_{drone} \leq \epsilon$$

The objective is to minimize the time that both vehicles arrive to vertex $n + 1$ (e.g. return to the depot). We will now discuss the mathematical model of the TSP-mVD in the next section.

6.2 MATHEMATICAL MODEL

Our formulation uses a notation that we call "component" to represent the solution. A component starts and ends at nodes on which one of two synchronization operations (launch or retrieve) occurs. A component is represented by its starting node and all components constitute a complex solution of the problem. Figure 6.1 shows a solution with four components: C_1 includes vertices (0, 1, 2), C_2 includes (2, 3, 4, 5, 6), C_3 includes (6, 7), and C_4 includes (7, 8, 9). The formulation for the TSP-mVD uses following additional mathematical notations:

- $V_0 = 0, 1, \dots, n$: set of all customers and the depot
- $V_+ = 1, \dots, n, n+1$: set of all customers and the depot's copy.
- s_L : launching time of drone
- s_R : retrieving time of drone
- q : maximum number of customers that drone can service after being launched from truck (should be defined earlier)

- ϵ : is the endurance of drone

Variables of the model are as follows:

- z_k : binary variable equal to 1 if node k is the starting node of a component.
- x_{ki} : binary variable equal to 1 if node i belongs to component k and truck is present at node i
- x'_{ki} : binary variable equal to 1 if node i belongs to component k and drone is present at node i
- y_{kij} : binary variable equal to 1 if two nodes i and j belong to component k and truck travels from i to j .
- y'_{kij} : binary variable equal to 1 if two nodes i and j belong to component k and drone flies from i to j .
- u_i : non-negative continuous variable representing sequence in which node i is visited. This variable is used to eliminate sub-tours.
- t_k : non-negative continuous variable representing total time to complete service at component k .
- l_k : binary variable equal to 1 if at the last node of component k , the drone is launched again.

For the solution showed in Figure 6.1, non-zero values of binary variables are: $z_0, z_2, z_6, z_7, z_9, x_{00}, x_{01}, x_{02}, x_{22}, x_{23}, x_{62}, x_{66}, x_{67}, x_{77}, x_{79}, x'_{00}, x'_{01}, x'_{02}, x'_{22}, x'_{24}, x'_{25}, x'_{26}, x'_{66}, x'_{67}, x'_{77}, x'_{78}, x'_{79}, y_{001}, y_{012}, y_{223}, y_{236}, y_{667}, y_{779}, y'_{224}, y'_{245}, y'_{256}, y'_{778}, y'_{789}, l_0, l_2$, and l_6 .

The formulation for the problem can be stated as follows:

$$\text{Minimize } \sum_{k \in V_0} t_k \quad (6.1)$$

$$\text{s.t. } z_0 = 1 \quad (6.2)$$

$$z_{n+1} = 1 \quad (6.3)$$

$$u_0 = 0 \quad (6.4)$$

$$u_{n+1} = n + 1 \quad (6.5)$$

$$x_{kk} = z_k \quad \forall k \in V_0 \quad (6.6)$$

$$x'_{kk} = z_k \quad \forall k \in V_0 \quad (6.7)$$

$$x_{k0} = 0 \quad \forall k \in V_+ \quad (6.8)$$

$$x'_{k0} = 0 \quad \forall k \in V_+ \quad (6.9)$$

$$\sum_{k \in V_0} x_{ki} + \sum_{k \in V_0} x'_{ki} - 3z_i = 1 \quad \forall i \in N \quad (6.10)$$

$$\sum_{k \in V_0} x_{k(n+1)} = 1 \quad (6.11)$$

$$\sum_{k \in V_0} x'_{k(n+1)} = 1 \quad (6.12)$$

$$\sum_{k \in V_0} x_{ki} \geq 1 \quad \forall i \notin N_D \quad (6.13)$$

$$\sum_{i \in V} x_{ki} \leq (n+2)z_k \quad \forall k \in V_0 \quad (6.14)$$

$$\sum_{i \in V} x'_{ki} \leq (q+2)z_k \quad \forall k \in V_0 \quad (6.15)$$

$$z_i + x_{ki} - x'_{ki} \leq 1 \quad \forall k \in V_0, \forall i \in V_+, i \neq k \quad (6.16)$$

$$z_i + x'_{ki} - x_{ki} \leq 1 \quad \forall k \in V_0, \forall i \in V_+, i \neq k \quad (6.17)$$

$$\sum_{\substack{i \in V_0 \\ i \neq j}} y_{kij} = x_{kj} \quad \forall k \in V_0, \forall j \in V_+, k \neq j \quad (6.18)$$

$$\sum_{\substack{j \in V_+ \\ j \neq i, k}} y_{kij} \geq x_{ki} - z_i \quad \forall k \in V_0, \forall i \in N, i \neq k \quad (6.19)$$

$$\sum_{\substack{j \in V_+ \\ j \neq i, k}} 2y_{kij} \leq x_{ki} + 1 - z_i \quad \forall k \in V_0, \forall i \in N, i \neq k \quad (6.20)$$

$$\sum_{i \in V_+} y_{kki} \geq z_k \quad \forall k \in V_0 \quad (6.21)$$

$$\sum_{\substack{i \in V_0 \\ i \neq j}} y'_{kij} \leq x'_{kj} \quad \forall k \in V_0, \forall j \in V_+, k \neq j \quad (6.22)$$

$$\sum_{\substack{i \in V_0 \\ i \neq j}} y'_{kij} \geq x'_{kj} - z_j \quad \forall k \in V_0, \forall j \in V_+, k \neq j \quad (6.23)$$

$$\sum_{\substack{j \in V_+ \\ j \neq i, k}} y'_{kij} \geq x'_{ki} - z_i \quad \forall k \in V_0, \forall i \in N, i \neq k \quad (6.24)$$

$$\sum_{\substack{j \in V_+ \\ j \neq i, k}} 2y'_{kij} \leq x'_{ki} + 1 - z_i \quad \forall k \in V_0, \forall i \in C, i \neq k \quad (6.25)$$

$$u_i - u_j + (n+1)y_{kij} \leq n \quad \forall k \in V_0, \forall i \in V_0, \forall j \in V_+ \quad (6.26)$$

$$u_i - u_j + (n+1)y'_{kij} \leq n \quad \forall k \in V_0, \forall i \in V_0, \forall j \in V_+ \quad (6.27)$$

$$l_k \geq \sum_{j \in V_+} y'_{ijj} + x_{ki} - 1 \quad \forall k \in V_0, \forall i \in V_+, k \neq i \quad (6.28)$$

$$t_k \geq \sum_{\substack{i \in V_0 \\ i \neq j}} \sum_{\substack{j \in V_+ \\ j \neq k}} \tau'_{ij} y'_{kij} + s_R \sum_{\substack{j \in V_+ \\ j \neq k}} y'_{kkj} \quad \forall k \in V_0 \quad (6.29)$$

$$t_k \geq \sum_{\substack{i \in V_0 \\ i \neq j}} \sum_{\substack{j \in V_+ \\ j \neq k}} \tau_{ij} y_{kij} + s_R \sum_{\substack{j \in V_+ \\ j \neq k}} y'_{kkj} + s_L l_k \quad \forall k \in V_0 \quad (6.30)$$

$$\sum_{\substack{i \in V_0 \\ i \neq j}} \sum_{\substack{j \in V_+ \\ j \neq 0}} \tau'_{ij} y'_{0ij} + s_R \sum_{j \in V_+} y'_{00j} \leq \epsilon \quad (6.31)$$

$$t_k \leq \epsilon + M(1 - \sum_{\substack{j \in V_+ \\ j \neq k}} y'_{kkj}) \quad \forall k \in N \quad (6.32)$$

$$u_i \in \mathbb{R}^+, 0 \leq u_i \leq n+1 \quad \forall i \in V \quad (6.33)$$

$$x_{ki}, x'_{ki} \in \{0, 1\} \quad \forall k \in V, \forall i \in V \quad (6.34)$$

$$y_{ijk}, y'_{ijk} \in \{0, 1\} \quad \forall k \in V, \forall i \in V, \forall j \in V \quad (6.35)$$

$$l_k \in \{0, 1\} \quad \forall k \in V_0 \quad (6.36)$$

$$t_K \in \mathbb{R}^+ \quad \forall k \in V_0 \quad (6.37)$$

The objective function (6.1) is to minimize the total service time of all components t_k which is also the latest time the truck and the drone return to the depot. Constraints (6.2)-(6.5) fix the value of variables z and u at the depot and its copy. Constraints (6.6) and (6.7) mean that if a component starts at k , both vehicles must be present at k and this node also belongs to the component. Constraints (6.8) and (6.9) say that the depot cannot belong to any component other than o . Constraint (6.10) imposes each customer must be visited. More precisely, if i is not a node at which a synchronization operation occurs (e.g. $z_i = 0$), it must be serviced either by truck or drone. On the other hand, if $z_i = 1$, the constraint indicates that node i must belong to two components. For example, in Figure 6.1, node 2 belongs to two components C_0 and C_2 leading to four binary variables that are set to 1: x_{02} , x'_{02} , x_{22} , and x'_{22} . Constraints (6.11) and (6.12) written for node $n+1$ indicate that the last node on which both vehicles are present must belong to only one component. Constraint (6.13) says that, if i is a customer that cannot be serviced by drone, it must be serviced by truck.

Constraints (6.14) and (6.15) state that if node k is not the representation of a component, there is no node belonging to a component that starts from node k . Constraint (6.15) ensures also the capacity of the drone, e.g. in each component, drone cannot bring more than q parcels. Constraints (6.16) and (6.17) partly ensure the synchronization. If component k ends at node i (e.g. z_i and x_{ki} are both equal to 1), both vehicles must be present on node i (e.g. x'_{ki} must also be set to 1).

The flow conservation for the truck in each component is satisfied by Constraints (6.18)-(6.20). More precisely, consider component k , Constraint (6.18) indicates that there is maximally one arc traveled by truck going in node j . Constraints (6.19) and (6.20) mean that if node i is not a representation of another component ($z_i = 0$) and truck is present at i belonging to component k ($x_{ki} = 1$), there is only one arc traveled by truck going out i . Constraint (6.21) implies that, if node k other than the duplication of depot $n+1$ is a representation of a component ($z_k = 1$), there must be an arc of component k traveled by truck going out node k . Similarly, Constraints (6.22)-(6.25) assure the flow conservation for the drone. However the difference

between truck and drone flows lies in the fact that at representative nodes (nodes 2, 7, 0 and 6 in Figure 6.1), drone can be placed on the truck and does not fly in or out these nodes. This leads to constraints (6.22) and (6.23) for drone different from constraint (6.18) for truck. In addition, there is no constraint similar to (6.21) for the drone.

The sub-tour elimination is ensured by Constraints (6.26) and (6.27). Constraint (6.28) computes the value of variable l_k for component k . There is a launch operation from the last node i of the component k , if and only if drone flies from i to another node j . Constraints (6.29) and (6.30) calculate t_k , the completion time of component k . Constraints (6.31) and (6.32) make sure the endurance requirement of drone. Note that because the drone can be launched from the depot any time, it is not necessary to wait for the truck in the first rendez-vous location. Therefore, for the component starting at the depot, the endurance constraint is restricted by (6.31). And finally, remaining constraints (6.33)-(6.37) define the domain of variables.

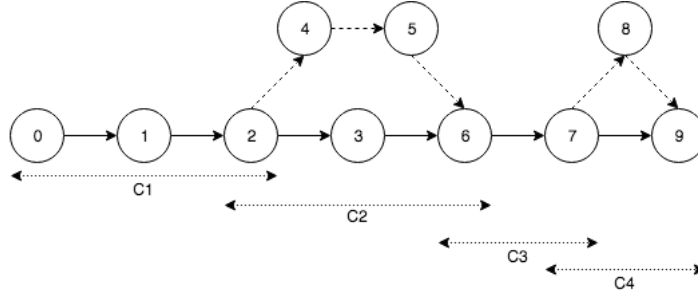


Figure 6.1: Illustration of TSP-mVD solution

6.3 A SPLIT ALGORITHM FOR TSP-MVD

In this section, we discuss a split algorithm for the TSP-mVD and restore method to reinsert drone nodes to truck tour. We first discuss the split method. Given a TSP tour s , this algorithm will optimally select the nodes that could be drone nodes while respecting their relative orders. Originally, split algorithm was used in many variants of the well-known Vehicle Routing Problem, as in [Prio4], [PLR09], [Pri09b], [Cat+14], [Vid16], [Vid+13b]. This algorithm's version for TSP-D was firstly introduced in [Ha+18b] where it can solve the problem of single drone visit. This paper continues to adapt this fast and efficient method to solve the TSP-mVD.

In detail, we first build an auxiliary graph $H = (V', A')$ based on the TSP tour s of the graph $G = (V, A)$ where $V = V'$ and $(i, j) \in A'$ represents a subroute from i to j in the TSP tour, i is placed before j in the TSP tour. Algorithm 16 describes the steps to calculate the cost of the arcs in graph H . In detail, we first initialize the cost of all arcs to a big number (Line 1). We then calculate the cost of all arcs of nodes that are adjacent to each other in s (Line 2 and 3). After this,

we start to compute the arcs of nodes that are not adjacent (Line 4, 5). At first, we store the best arc cost in *minValue* (Line 6). For each limit m of drone visit q (Line 7), we consider all the subsequences of size m (Line 8). Each of these subsequences must contain nodes that are able to be visited by drone (Line 10). We then calculate the travel time of truck and drone given that this *droneSequence* is visited by drone which is launched at i and return to k (Line 11, 12). More importantly, these travel times must be checked to ensure that they meet the drone endurance constraint (Line 13). The effective time is then calculated by taking the maximum travel time between truck and drone (Line 14). We also add the launch time if i is not a depot node in Line 15 and 16 to incorporate the time the truck needs to prepare the drone for next launch. Next, the best value is updated if this effective time is better (Line 18). Finally, we assign the arc cost with the *minValue*.

After the calculation of arcs' costs is finished, the cost of the shortest path from depot to node k is computed using dynamic programming

approach and the corresponding TSP-D solution could be extracted, as described in [Ha+18b].

Algorithm 16: Cost calculation in TSP-mVD split

Data: Tour s , graph G
Result: Cost of all *arcs* in graph H

```

1  $arcs_{ij} =$  a big double number  $\forall i, j \in A'$  ;
2 for  $i = 0 \rightarrow s.size() - 2$  do
3    $arcs_{i,i+1} = \tau_{ij}$  ;
4 for  $i = 0 \rightarrow s.size() - 3$  do
5   for  $k = i + 2 \rightarrow s.size() - 1$  do
6      $minValue =$  a big double number ;
7     for  $m = 1 \rightarrow q$  do
8       for  $j = i + 1 \rightarrow k - m - 1$  do
9          $droneSequence =$  subsequence of  $s$  from  $j$  to  $j + m$ 
10        ;
11        if All nodes in  $droneSequence$  are allowed to be
12        visited by drone then
13           $droneTime =$  (drone travel time when being
14          launched at  $s[i]$ , services all nodes in
15           $droneSequence$  and comes back to  $s[k]$ ) +  $s_R$  ;
16           $truckTime =$  (truck travel time with nodes in
17           $droneSequence$  being removed) +  $s_R$  ;
18          if  $droneTime \leq \epsilon$  and  $truckTime \leq \epsilon$  then
19             $effectiveTime =$ 
20             $\max(droneTime, truckTime)$  ;
21            if  $s[i] \neq 0$  then
22               $effectiveTime = effectiveTime + s_L$  ;
23            if  $effectiveTime < minValue$  then
24               $minValue = effectiveTime$  ;
25           $arcs_{ik} = minValue$ ;

```

6.4 ITERATED LOCAL SEARCH FOR TSP-MVD

This section presents Iterated Local Search (ILS) [GP10], a metaheuristic that is used to solve the TSP-mVD problem. We first describe briefly the key concept of the algorithm and then specify in detail its main components.

The key idea of ILS is to focus the search on a smaller set of the search space instead of the total solution space. In order to realise this idea, the algorithm first starts with a local optimal solution found by the local search and then repeatedly applies the local search procedure on a set of solutions obtained by perturbing (shaking) the previously visited local optimum solutions. The ILS in this paper is

composed of five components which corresponds to five procedures: (1) *InitialSolution* where the initial solution is obtained; (2) *LocalSearch* where the solution obtained at the beginning of each iteration is improved; (3) *Perturb* where we "shake" the solution by using different shaking operators to get a new start point; (4) *Restore* where we update the giant tour with the information from the "educated" TSP-D solution; and (5) *AcceptanceCriterion* where we decide which solution is chosen as a new start point.

Algorithm 17 describes the overall framework of ILS used in this paper. In detail, we initialize of set H of 2-tuples of giant-tour and its TSP-D value in Line 1. The initial giant tour s and TSP-D s_{tspd} are then obtained by using *InitialSolution* method (Line 2). The probability of accepting bad moves is initially set to 0.5 (Line 7). While the stopping criteria has not been met (Line 8), for each iteration, we first "shake" the current giant-tour by using the *Perturb* method (Line 10). The perturbed tour s is then splitted using split algorithm to obtain the corresponding TSP-D solution (Line 11). This solution is then passing to a local search method to improve its quality (Line 12). After being educated, the giant-tour s is updated with restore method (Line 13). We then update the best objective and best tour in Lines 14-17. In the next step, if the new tour s can improve the previous tour s' , it is added to the history H . This list of past tours is then sorted in ascending order and truncate to the size of 10 elements (Lines 18-21). After that, *AcceptanceCriterion* method is called to decide which tour will be chosen for the next solution. If the considering tour s is accepted, the current tour s' is replaced by s (Lines 22-23). Otherwise, we choose a random tour in the history (Lines 24-28). Finally, we obtain the best objective value v^* , its tour s^* and TSP-D solution s^*_{tspd} .

The next sections discuss the 5 steps of ILS described above which are getting initial solution, local search, restore method and acceptance criterion.

Algorithm 17: Iterated Local Search for TSP-mVD**Result:** Best objective v^* , its giant tour s^* , TSP-D solution s^*_{tspd}

```

1  $H = \{\}$ ;
2  $s, s_{tspd} = \text{InitialSolution}()$ ;
3  $s' = s$ ;
4  $s^* = s, s^*_{tspd} = s_{tspd}$ ;
5  $v^* = f(s)$ ;
6  $\text{consecutiveIter} = 0$ ;
7  $\text{acceptProb} = 0.5$ ;
8 while  $\text{consecutiveIter} < n_{NI}$  do
9    $\text{consecutiveIter} = \text{consecutiveIter} + 1$ ;
10   $s = \text{Perturb}(s', s_{tspd})$ ;
11   $s_{tspd} = \text{Split}(s)$ ;
12   $s_{tspd} = \text{LocalSearch}(s)$ ;
13   $s = \text{Restore}(s_{tspd})$ ;
14  if  $f(s) < f(s^*)$  then
15     $v^* = f(s)$ ;
16     $s^* = s$ ;
17     $s^*_{tspd} = s_{tspd}$ ;
18  if  $f(s) < f(s')$  then
19     $H = H + \langle s, s_{tspd} \rangle$ ;
20    Sort  $H$  in ascending order of  $v$  value;
21    Remove worst element from  $H$  until  $|H| = 10$ ;
22  if  $\text{AcceptanceCriterion}(s, s', H, \text{acceptProb})$  then
23     $s' = s$ ;
24  else
25     $r = \text{random number between } (0, 1]$ ;
26     $y = 6$  is the degree of randomness;
27     $s' = \text{giant tour of } H[|H| * r^y]$ ;
28     $s_{tspd} = \text{TSP-D solution of } H[|H| * r^y]$ ;

```

6.4.1 Initial solution

Algorithm 18 describes the step to obtain the initial solution for ILS. In detail, we first construct a TSP tour based on a constructive heuristic. There exists multiple heuristics for TSP tour construction such as k-nearest neighbor and k-cheapest insertion. In this paper, k-cheapest insertion is used with $k = 3$ (Line 1). This TSP tour is then used as input for the split method to get the TSP-D solution. Again, this solution is "educated" using local search method and is restored to have

the updated giant tour (Lines 2-4). Finally, we push this giant tour into history H and return it.

Algorithm 18: InitialSolution

Result: s_0, s_{tspd}

```

1  $s_0$  = TSP tour based on k-cheapest insertion ;
2  $s_{tspd} = Split(s_0)$  ;
3  $s_{tspd} = LocalSearch(s_{tspd})$ ;
4  $s_0 = Restore(s_{tspd})$  ;
5  $H = H + \langle s_0, s_{tspd} \rangle$  ;
6 return  $s_0, s_{tspd}$  ;
```

6.4.2 Local search operators

After the giant tour is split to obtain a TSP-D solution, this one will be "educated" using local search. In this paper, we use a set move operators which combines the ones proposed in [Ha+18b], [Ha+18a] and 4 new moves – the Drone sequence moves – as described below:

- Truck Swap 1-1: the two random positions of truck tour are exchanged.
- Truck Swap 2-1: two consecutive nodes u, v are swapped with another random nodes x in the truck tour.
- Truck Swap 2-2: two pair of consecutive node u, v and x, y are swapped.
- Truck-only Relocate 1-1: relocate a random truck-only node x after random node y .
- Truck-only Relocate 2-1: relocate a random pair of truck-only nodes u, v after random node x . We also consider move where u, v are reversed to v, u when being relocated to x .
- 2-opt: two pairs of consecutive nodes in the truck tour u, v and x, y are swapped to have u, x and v, y .
- Drone delivery creation: we choose a truck-only node j , two nodes i and k and make a drone delivery where i and k are launch and rendezvous nodes and j is the drone node.
- Drone delivery relocation: given a drone sequence J of a drone delivery $\langle i, J, k \rangle$, we relocate J to have another launch and rendezvous nodes i' and k' in the truck tour
- Drone delivery removal: given a drone delivery $\langle i, J, k \rangle$, we remove this drone delivery by reinserting nodes in J after a random node x in the truck tour.

- Inter drone-truck swap 1-1: we swap the launch or rendezvous node of a drone delivery with a drone node of another drone delivery.
- Inter truck-rdv swap 1-1: we swap the position of launch and rendezvous node of a drone delivery.
- Drone sequence insertion: we insert a truck-only node x into a drone sequence J of an existing drone delivery.
- Drone sequence removal: we consider a drone sequence J of a drone delivery and remove a node $x \in J$ and reinsert it after a node y in the truck tour.
- Drone sequence swap: we swap two drone nodes that belongs to two different drone sequences J and J' of two drone deliveries.
- Drone sequence intra exchange: we exchange position of a drone sequence J of a drone delivery.

Finally, to speed up the performance of local search method, we adapt the technique proposed in [TV03] in which we avoid scanning all the neighborhood but the ones that are closest to the considering node(s). The granular threshold h is therefore set to 0.1 (10% of the node).

6.4.3 Perturbation technique

In this section, we discuss the perturbation technique where a giant tour is "shaken" to have a new starting point of the ILS. In Algorithm 19, the method takes the current giant tour s and TSP-D solution s_{tspd} as the input and return a perturbed tour $newTour$. Firstly, the TSP-D solution is extracted into two parts: the truck tour and the list of drone deliveries (Line 1). The new tour is also initialized with default value of -1 (Lines 2-3). After this step, we copy all the truck tour nodes to the new tour (Lines 4-6). In the third step, we iterate through each of the drone nodes in the drone deliveries and decide with probability of 50% whether to copy this node to the $newTour$ or not (Lines 7-12). Finally, the uncopy nodes are gathered into a list and shuffle to be used to complete the unassigned nodes in $newTour$ (Lines 13-14).

The intuition behind this technique is based on the observation firstly mentioned in [MC15] in which the quality of TSP tour has a positive impact on the quality of the TSP-D solution under min-time objective. From this information, the perturbation technique is designed in the way that keeps the positions of the truck tour and only "shake" a part of the drone nodes to avoid too much different (and possibly bad) new TSP tour. Furthermore, preliminary tests also

show that some very common TSP perturbation methods such as the "Double-bridge" does not work well with ILS in this paper.

Algorithm 19: Perturbation technique

Data: s, s_{tspd}
Result: $newTour$

```

1  $truckTour, droneDeliveries = s_{tspd}$ ;
2  $newTour = s$ ;
3 Replace all nodes in  $newTour$  with  $-1$ ;
4 forall  $n \in truckTour$  do
5    $in = \text{index of } n \text{ in } s$ ;
6    $newTour[in] = n$ ;
7 forall  $J$  in  $droneDeliveries$  do
8   forall  $j \in J$  do
9      $r = \text{random number between } (0, 1]$ ;
10    if  $r \leq 0.5$  then
11       $in = \text{index of } j \text{ in } s$ ;
12       $newTour[in] = j$ ;
13  $u = \text{uncopy nodes, shuffled in random order}$ ;
14 Fill the unassigned positions of  $newTour$  with nodes from  $u$ ;
15 return  $newTour$ ;

```

6.4.4 Restore method

Different from split method where truck nodes are selected to become drone nodes and to obtain a TSP-D solution, the restore method reinserts the drone nodes to the truck nodes given a TSP-D solution. The description of this method is described in Algorithm 20. More specifically, given a TSP-D solution with a truck tour and set of drone deliveries, a giant tour s is firstly initialized with nodes from $truckTour$. Then, for each drone delivery, we insert the drone nodes J to a random position between launch node i and rendezvous node k in s . The

purpose of restore method is to boost up the convergence speed of the search since the giant tour is updated with the "educated" sequences.

Algorithm 20: Restore method for TSP-mVD

Data: *truckTour*, *droneDeliveries* of a TSP-D solution

Result: A giant tour *s*

```

1 s = truckTour;
2 foreach drone nodes set J in droneDeliveries do
3   i = launch node ;
4   k = rendezvous node ;
5   if  $k - i = 1$  then
6     insert drone nodes in J before k in s, respecting their
       orders ;
7   else
8     r = random node between i and k (including k) ;
9     insert drone nodes in J before r in s, respecting their
       orders ;

```

6.4.5 Acceptance criterion

Acceptance criterion plays a role in deciding whether we should accept the newly obtained solution. If the new solution is better than the previous one then we immediately accept it. Otherwise we only accept this new solution with a probability *acceptProb* which is initially set to 0.5 and decrease gradually, in detail, $acceptProb = acceptProb / 1.0002$ after each time a bad solution is allowed. By doing this, we allow the search to explore various regions in the search space at the beginning iterations with the hope to find good solutions. Moreover, when approaching the final iterations of the algorithm, the probability of accepting bad moves would be very small, making the algorithm to focusing on finding high-quality solutions.

Algorithm 21: AcceptanceCriterion

Data: *s*, *s'*, *H*, *acceptProb*

```

1 if  $s' < s$  then
2   return true ;
3 else
4   rndProb = random value between (0, 1] ;
5   if  $rndProb \leq acceptProb$  then
6      $acceptProb = acceptProb / 1.0002$  ;
7     return true ;
8   else
9     return false ;

```

6.5 COMPUTATIONAL RESULTS

In this section, we present the computational results of MIP formulation and ILS framework. MIP formulation is run using CPLEX 12.6 solver under an Intel Xeon E5620 @ 2.4GHz and 16 GB of memory. Our metaheuristic is implemented in C++ language and the tests are run on a desktop PC equipped with Intel i7-6700 @ 3.4 GHz processor and 16 GB of memory.

To generate TSP-mVD instances, we use TSP-D instances proposed in [MC15; Ha+18b] adding the information related to the maximum number of visits the drone can perform during a flight. By experiment, we found the best settings for ILS as follows. n_{NI} is set to 3000 for drone visit limit $q = 1$, 1500 for $q = 2, 5$. The degree of randomness y in 17 is set to 6 to avoid too much randomness in choosing history tours.

This section is organised as follows. We first validate the exactitude of MILP formulation and evaluate the performance of ILS against optimal solutions obtained by the formulation on small instances. The next sub-section discusses the performance of ILS on TSP-D instances (where the maximum visits per flight of drone is limited to 1). Finally, we analyse the impact of drone capacity to the problem solutions.

- The truck speed and drone speed are set to 40 km/h.
- Drone endurance is set to 20 minutes.
- The maximum number of drone visits per drone delivery is set to $q = 2$ by default.

6.5.1 Results of MIP formulation and ILS on small instances

The first experiment is designed to validate the MILP formulation and access the solution quality of ILS algorithm on small instances proposed in [Ha+18b]. Again, the drone capacity q is set to 2. On each instance, ILS is repeatedly run 10 times. The results are reported in Table 6.2 where column *obj* represents the objective value, column *T* is the running time of MILP solver in seconds, T_{avg} is the averaged running time of ILS, and *opt* is the number of times ILS can reach optimal solution.

As can be seen, ILS could be able to reach the optimal values in all runs on all instances in less than 2 seconds, comparing to hours of running time from MILP formulation run by solver. This proves the effectiveness of ILS designed in this paper for solving the problem.

Instance	TSP	MILP formulation		ILS		
	<i>obj</i>	<i>obj</i>	<i>T(s)</i>	<i>obj</i>	$T_{avg}(s)$	<i>opt</i>
A1	1.00733	0.84648	26465	0.84648	0.84	10
A2	0.95588	0.87581	17411	0.87581	1.06	10
A3	0.98568	0.85092	23321	0.85092	0.78	10
A4	0.94464	0.86195	13318	0.86195	1.70	10
A5	0.98568	0.86635	15321	0.86635	1.63	10

Table 6.1: Compare the solutions of MILP formulation and ILS on small instances $q = 2$

Instance	TSP	MILP formulation		ILS		
	<i>obj</i>	<i>obj</i>	<i>T(s)</i>	<i>obj</i>	$T_{avg}(s)$	<i>opt</i>
A1	1.00733	0.85817	1688	0.84648	0.84	10
A2	0.95588	0.89960	23027	0.87581	1.06	10
A3	0.98568	0.87293	2535	0.85092	0.78	10
A4	0.94464	0.89952	3798	0.86195	1.70	10
A5	0.98568	0.90720	5821	0.86635	1.63	10

Table 6.2: Compare the solutions of MILP formulation and ILS on small instances $q = 1$

6.5.2 Performance of ILS on TSP-D instances

This section reports the performance of ILS on TSP-D where we only allow single drone visit per drone delivery, i.e. $q = 1$. In detail, ILS is tested on two sets of instances, one with 36 instances of 10-customer size from [MC15] and the other with 60 larger size instances from [Ha+18b]. Tables 6.3 and 6.4 report these results.

In Table 6.3, ILS is repeatedly run 30 times and the best result is shown in Column "ILS", the averaged result is presented in Column $\overline{ILS}$. Column ϵ is the drone endurance in two settings: 20 and 40. The bold value in Column "ILS" indicates that ILS could find better solution than the best known solution among GRASP and Murray et al in the same instance. In overall, ILS could improve 10 best known solutions from [Ha+18b] and perform as good as GRASP or Murray et al in other instances. Again, this shows the effectiveness of ILS comparing to previous methods.

In Table 6.4, we compare ILS with GRASP in [Ha+18b] in terms of averaged best value (Columns $\overline{GRASP}$, $\overline{ILS}$, running time in minutes (Columns T_{GRASP} , T_{ILS}) and the change in percentage of best known solutions (Column $\overline{Change_{BKS}}$). The negative value in this percentage indicates an improvement of ILS over GRASP and vice versa. In overall, ILS could improve the best known solution by nearly 2% in average and up to 3% when keeping the average running time less than 2 minutes. Its running time, on the other hand, is slower than GRASP by nearly 30%. However, when taking into account the fact

that ILS is more complex in terms of algorithm design, and that the algorithm's running time is acceptable, this trade-off is tolerable.

Instance	ϵ	Murray et al.	GRASP	ILS	$\overline{ILS}$	Instance	ϵ	Murray et al.	GRASP	ILS	$\overline{ILS}$
437V1	20	56.468	57.446	56.468	56.468	440V7	20	49.996	49.776	49.422	49.422
437V1	40	50.573	50.573	50.573	50.573	440V7	40	49.204	49.204	49.204	49.204
437V2	20	53.207	53.207	53.207	53.207	440V8	20	62.796	62.700	62.222	62.222
437V2	40	47.311	47.311	47.311	47.311	440V8	40	62.270	62.004	62.004	62.004
437V3	20	53.687	54.664	53.687	53.687	440V9	20	42.799	42.566	42.533	42.533
437V3	40	53.687	53.687	53.687	53.687	440V9	40	42.799	42.566	42.533	42.533
437V4	20	67.464	67.464	67.464	67.464	440V10	20	43.076	43.076	43.076	43.076
437V4	40	66.487	66.487	66.487	66.487	440V10	40	43.076	43.076	43.076	43.076
437V5	20	50.551	50.551	50.551	50.551	440V11	20	49.204	49.204	49.204	49.204
437V5	40	45.835	44.835	44.835	44.835	440V11	40	49.204	49.204	49.204	49.204
437V6	20	45.176	47.601	47.311	47.311	440V12	20	62.004	62.004	62.004	62.004
437V6	40	45.863	43.602	43.602	43.602	440V12	40	62.004	62.004	62.004	62.004
437V7	20	49.581	49.581	49.581	49.581	443V1	20	69.586	69.586	69.587	69.587
437V7	40	46.621	46.621	46.621	46.621	443V1	40	55.493	55.493	55.493	55.493
437V8	20	62.381	62.381	62.381	62.381	443V2	20	72.146	72.146	72.146	72.146
437V8	40	59.776	59.416	59.416	59.416	443V2	40	58.053	58.053	58.053	58.053
437V9	20	45.985	42.945	42.416	42.416	443V3	20	77.344	77.344	77.344	77.344
437V9	40	42.416	42.416	42.416	42.416	443V3	40	69.175	68.431	68.431	68.431
437V10	20	42.416	41.729	41.729	41.729	443V4	20	90.144	90.144	90.144	90.144
437V10	40	41.729	41.729	41.729	41.729	443V4	40	82.700	83.700	82.700	82.700
437V11	20	42.896	42.896	42.896	42.896	443V5	20	55.493	58.210	54.973	55.077
437V11	40	42.896	42.896	42.896	42.896	443V5	40	53.447	51.929	51.929	51.929
437V12	20	56.696	56.425	56.273	56.273	443V6	20	58.053	58.053	55.209	56.631
437V12	40	55.696	55.696	55.696	55.696	443V6	40	52.329	52.329	52.329	52.329
440V1	20	49.430	50.164	49.430	49.430	443V7	20	64.409	65.523	65.523	65.523
440V1	40	46.886	46.886	46.886	46.886	443V7	40	60.743	60.743	60.743	60.743
440V2	20	50.708	51.828	50.708	50.708	443V8	20	77.209	78.323	78.323	78.323
440V2	40	46.423	46.423	46.423	46.423	443V8	40	73.967	72.967	72.967	72.967
440V3	20	56.102	58.502	56.102	56.102	443V9	20	49.049	45.931	45.931	45.931
440V3	40	53.933	53.933	53.933	53.933	443V9	40	47.250	45.931	45.931	45.931
440V4	20	69.902	73.091	69.902	69.902	443V10	20	47.935	46.935	46.935	46.935
440V4	40	68.397	68.397	68.397	68.397	443V10	40	47.935	46.935	46.935	46.935
440V5	20	43.533	44.624	43.533	43.533	443V11	20	57.382	56.395	56.395	56.395
440V5	40	43.533	43.533	43.533	43.533	443V11	40	56.395	56.395	56.395	56.395
440V6	20	44.076	44.122	43.949	43.949	443V12	20	69.195	69.195	69.195	69.195
440V6	40	44.076	43.944	43.810	43.841	443V12	40	69.195	69.195	69.195	69.195

Table 6.3: Comparison with small instances from Murray and Chu's work.

6.5.3 Impact of drone capacity

In this section, we show the efficiency of using multiple visit drone by evaluating the impact of drone capacity on solution of ILS algorithm. Three values of $q = (1, 2, 5)$ are selected in the test. For each combination of instance and q , ILS is repeatedly run 30 times. In total, 5400 tests have been conducted. The results are reported in Tables 6.5 and 6.6, each of which belongs to a different instance set (of 50 and 100-customer). Among these tables, we report the best known

Instance	$\overline{GRASP}$	T_{GRASP} (min)	$\overline{ILS}$	T_{ILS} (min)	$\overline{Change_{BKS}}$ (%)
B	121.05	0.41	118.80	0.66	-1.97
C	230.193	0.30	225.25	0.55	-2.01
D	320.91	0.29	314.40	0.55	-1.75
E	193.17	4.65	190.70	3.63	-0.66
F	337.94	2.30	327.07	4.32	-3.01
G	453.74	1.88	438.24	3.77	-2.50
Mean		0.95		1.51	-1.98

Table 6.4: Performance of ILS on TSP-D on larger instances (50 and 100-customer).

solutions (Column "BKS"), average values (Column "Avg"), and averaged running times (Column "Time") and the change in percentage of the best known value between $q = 1$ and other values of q (Column " $Change_{BKS}$ "). A negative value of $Change_{BKS}$ indicates an improvement in the solution when drone is capable of serving multiple customers in a flight.

In overall, we can observe that by allowing drone to visit multiple customer locations, we can significantly increase the system performance. In average, with $q = 2$, we can achieve an improvement in BKS of approximately 7% while with $q = 5$, these improvements are approximately 11% and 16% for two set of instances.

Furthermore, the results show that allowing the drone to visit more customers could improve the service quality, however, this improvement is not linear with the maximum number of visits q . In detail, by increasing the drone visits by just one, we could receive results that are nearly 7% better. Nonetheless, increasing q to 5 does not necessarily increase the improvement by the same ratio (i.e. 35%). In fact, this number is only doubled approximately.

Moreover, we can also observe a significant impact on raising number of drone visits among high-density instances (the ones with label B, E, F) where solutions are improved up to 3 times. This is because in these instances, allowing the drone to visit multiple customer locations before coming back to the truck could help to save the repeatedly done recover time which takes up to 2 minutes each launch. This amount of time in high-density instances is a major impact to the solution where travel time between locations are relatively small and near to 2 minutes.

	q = 1 (TSP-D)			q = 2				q = 5			
Instance	BKS	Avg	Time	BKS	Avg	Time	<i>Change_{BKS}</i> (%)	BKS	Avg	Time	<i>Change_{BKS}</i> (%)
B1	116.24	119.41	0.66	103.60	108.07	0.62	-10.87	90.39	96.20	0.88	-22.24
B2	118.39	119.20	0.5	104.82	108.45	0.48	-11.46	96.58	98.51	1.08	-18.42
B3	116.72	118.80	0.69	104.56	106.53	0.56	-10.42	92.57	96.00	1.07	-20.69
B4	119.32	120.87	0.85	105.52	109.16	0.59	-11.57	90.22	95.11	1.01	-24.39
B5	117.26	118.56	0.5	103.21	107.17	0.57	-11.98	93.46	97.53	1.11	-20.30
B6	115.63	117.56	0.8	103.08	105.05	0.73	-10.85	84.08	91.19	0.95	-27.29
B7	115.96	116.80	0.77	103.50	105.81	0.68	-10.75	91.3	95.13	1.09	-21.27
B8	117.90	119.09	0.52	104.96	107.67	0.51	-10.98	92.62	96.77	0.98	-21.44
B9	117.72	118.59	0.59	105.22	106.63	0.60	-10.62	94.76	98.21	0.93	-19.50
B10	118.08	119.09	0.71	102.14	104.35	0.57	-13.50	89.48	92.58	0.9	-24.22
C1	219.13	221.34	0.57	194.09	197.45	0.56	-11.43	188.87	190.47	1.02	-13.81
C2	210.12	210.72	0.59	196.31	199.41	0.42	-6.57	186.92	189.39	0.97	-11.04
C3	212.02	213.70	0.46	207.40	208.96	0.38	-2.18	201.62	206.48	0.95	-4.91
C4	215.42	218.37	0.57	200.94	205.20	0.42	-6.72	199.16	202.95	0.96	-7.55
C5	220.50	224.48	0.47	219.12	221.72	0.37	-0.63	215.33	218.73	0.93	-2.34
C6	234.01	235.76	0.52	226.17	229.00	0.50	-3.35	219.81	224.66	0.94	-6.07
C7	223.68	226.96	0.58	205.48	210.05	0.46	-8.14	205.43	209.19	0.92	-8.16
C8	237.16	238.89	0.48	221.06	225.50	0.37	-6.79	219.71	223.37	0.95	-7.36
C9	230.58	234.77	0.57	218.21	224.70	0.45	-5.36	213.01	221.12	0.93	-7.62
C10	226.52	227.51	0.64	215.57	218.70	0.47	-4.83	211.23	214.44	0.93	-6.75
D1	311.08	314.98	0.57	304.48	307.65	0.50	-2.12	302.21	307.97	1.02	-2.85
D2	316.94	319.20	0.59	313.04	317.27	0.49	-1.23	313.33	316.82	0.93	-1.14
D3	298.11	300.83	0.36	288.57	289.80	0.36	-3.20	286.66	290.43	0.82	-3.84
D4	326.19	328.14	0.63	313.16	317.36	0.44	-3.99	311.23	315.71	0.95	-4.59
D5	319.44	320.78	0.46	314.16	315.46	0.46	-1.65	312.36	313.95	0.81	-2.22
D6	313.32	314.36	0.51	302.15	304.47	0.40	-3.57	298.14	300.85	0.76	-4.84
D7	318.53	322.85	0.69	306.09	308.51	0.43	-3.91	304.58	308.15	0.86	-4.38
D8	293.95	296.18	0.57	286.19	290.50	0.49	-2.64	285.98	288.84	0.75	-2.71
D9	316.74	319.70	0.58	308.36	312.18	0.40	-2.65	308.36	311.75	0.84	-2.65
D10	304.06	306.95	0.53	297.97	301.67	0.41	-2.00	295.78	300.21	0.94	-2.72
Mean			0.57			0.48	-6.53			0.94	-10.91

Table 6.5: TSP-mVD vs. TSP-D solution quality under 50-customer instances.

Instance	q = 1 (TSP-D)			q = 2				q = 5			
	BKS	Avg	Time	BKS	Avg	Time	Change _{BKS} (%)	BKS	Avg	Time	Change _{BKS} (%)
E1	188.23	190.41	3.63	170.64	173.77	3.72	-9.34	138.2	147.57	6.21	-26.58
E2	187.89	189.38	3.73	171.21	174.31	3.43	-8.88	139.38	147.1	5.35	-25.82
E3	188.67	190.03	4.02	171.05	175.35	2.99	-9.34	140.27	149.8	5.45	-25.65
E4	186.51	188.61	3.7	166.75	173.11	3.07	-10.59	133.55	146.24	6.34	-28.40
E5	188.07	190.61	3.06	172.58	175.86	2.96	-8.24	144.71	150.5	5.29	-23.06
E6	189.75	191.45	3.59	170.6	175.05	3.65	-10.09	136.88	145.95	5.06	-27.86
E7	190.66	192.08	3.68	173.32	176.52	3.07	-9.09	146.47	151	5.5	-23.18
E8	189.6	191.39	4.53	168.4	175.15	3.91	-11.18	140.21	147.53	6.24	-26.05
E9	189.08	191.15	3.36	173.65	176.83	3.44	-8.16	140.6	150.73	6.52	-25.64
E10	189.46	191.87	2.97	173.9	177.15	3.39	-8.21	144.09	152.79	6.49	-23.95
F1	334.93	339.16	4.17	302.48	317.44	3.08	-9.69	291.59	300.74	7.46	-12.94
F2	314.37	320.09	4.38	291.77	298.19	3.45	-7.19	271.38	282.79	7.08	-13.67
F3	322.84	328.52	4.3	296.97	309.16	2.73	-8.01	289.68	297.27	6.5	-10.27
F4	317.11	322.91	4.73	297.36	309.11	2.91	-6.23	273.76	291.69	7.32	-13.67
F5	327.93	334.11	4.05	300.12	314.24	3.04	-8.48	285.15	296.88	6.68	-13.05
F6	302.07	307.59	4.53	279.51	290.28	3.13	-7.47	258.47	270.03	6.77	-14.43
F7	316.76	324.15	4.68	298.68	306.49	2.73	-5.71	259.63	280.04	6.27	-18.04
F8	334.19	340.89	3.91	314.32	322.39	2.82	-5.95	292.15	304.99	6.12	-12.58
F9	319.22	325.94	4.28	303.96	314.79	2.73	-4.78	283.89	298.7	6.45	-11.07
F10	321.16	327.35	4.17	302.33	310.66	3.11	-5.86	280.37	297.94	6.11	-12.70
G1	426.44	435.55	4.24	405.14	420.35	3.23	-4.99	399.62	411.9	7.51	-6.29
G2	406.64	412	4.12	377.32	385.85	2.87	-7.21	361.37	379.49	6.9	-11.13
G3	432.8	442.97	3.83	421.25	430.89	2.53	-2.67	404.99	423.18	7.2	-6.43
G4	444.24	451.22	3.45	415.63	431.45	2.35	-6.44	405.67	423.95	5.95	-8.68
G5	426.38	432.73	4.06	397.68	415.46	2.98	-6.73	392.85	407.31	7.62	-7.86
G6	434.05	447.59	3.71	413.88	431.72	2.5	-4.65	403.13	423.84	7.36	-7.12
G7	416.78	423.34	3.83	387.26	405.04	3.05	-7.08	378.64	405.73	6.79	-9.15
G8	422.07	429.95	3.86	401.5	413.73	3.03	-4.87	400.13	413.53	7.23	-5.20
G9	451.36	459.3	3.51	428.28	441.87	2.84	-5.11	413.68	433.16	6.76	-8.35
G10	442.4	447.73	3.06	414.72	427.19	3.09	-6.26	403.43	424.59	6.41	-8.81
Mean			3.88			3.04	-7.28			6.46	-15.59

Table 6.6: TSP-mVD vs. TSP-D solution quality under 100-customer instances.

6.6 CONCLUSION

The Multi-Visits Traveling Salesman Problem with Drone(TSP-mVD) presents a new extension of the Traveling Salesman Problem with Drone (TSP-D). The drone has been improved to be able to fly with multiple cargo compartments. In this chapter, we consider TSP-mVD whose objective is to minimize the time required to service all customers and return both vehicles to the depot. We propose a model, a mixed integer linear programming formulation and a heuristic method based on ILS. We were so happy when the MILP formulation found optimal objectives of all instances of Murray and Chu (2015), as well as mention a mistake in that paper. The results also show the effectiveness of the drone can serve many customers in one launch. It also shows dominance performance of ILS comparing to the previous methods. Future researches could aim for proposing more efficient metaheuristics which based on our split procedure. The extension of

the proposed methods to solve problems with multiple vehicles and multiple drones could be also an interesting research direction.

CONCLUSION

The focus of this thesis is on the Traveling Salesman Problem with Drone where the drone could visit single or multiple customer locations in a launch. The objective is to either minimize the completion time or the operational cost of two vehicles. This is a very trending topic in the academic community as hundreds of publications have been proposed over the past few years. To the best of our knowledge, one of the result in this thesis could be considered to be the very first research on TSP-D after the work of [MC15] and [BAS17].

We now briefly summarize the results of this thesis and discuss the future directions.

7.1 RESULTS

The main achievements of this thesis is the introduction of two MIP formulations, two versions of split algorithm to solve the TSP-D and MVTSP-D along with three metaheuristics with tailored-made elements. We now briefly summarize these in detail, as follows:

- Chapter 4 introduces the first MIP formulation for the Min-cost TSP-D. The split algorithm for both min-cost and min-time TSP-D is also presented. These split methods are then used in a GRASP to effectively solve the TSP-D in both objectives. The results show that our proposed algorithm can outperform the previous work in an acceptable time.
- Chapter 5 discusses the HGA algorithm with a new crossover that can largely improve the best known results reported in the two previous works, with the change up to 16% and 4% for both objectives. The running time of HGA is also acceptable with the averaged running time is less than 5 minutes for large instances.
- Chapter 6, again, introduces another MIP formulation as the mathematical model for a variant of the TSP-D called TSP-mVD. The objective is to minimize the completion time. A split algorithm for TSPD-mVD is also developed and is used in an ILS algorithm with a tailored-made perturbation technique. The result shows that TSP-mVD with an increasing number of drone visit limit can significantly improve the solution quality of a drone-truck delivering operation.

7.2 FUTURE WORKS

In the future work, one could explore the following research directions:

- **The TSP-D with dynamic parcel properties** In the traditional TSP-D, parcels are only divided into two categories based on the capability of the drone to carry them. This does not reflect the real-world situation where parcels have various values in terms of size and weight. A future direction could take into account this diversity in size and weight of parcel to the TSP-D as these factors would make the drone endurance no longer a constant value but a function of parcel's information.
- **The TSP-D with Time Window** Another practical future direction for the TSP-D is to add the time window constraint to the traditional problem. With this time window property, the TSP-D would become closer to the real-world problem where customers are not always available.
- **The TSP-D with Multi-Drone** This variant of the TSP-D would allow the truck to carry more than one drone. Although we are aware of multiple existing works of this variant in the literature, this Multi-Drone TSP-D still has many aspects that could be explored such as the proposition of a MIP formulation, the development of different heuristics to solve different objectives.
- **The Capacitated VRP-D** The problem itself is a generalization of the TSP-D and it has been studied extensively for the past 3 years. However, most of the researches only focus on the theoretical analysis of the problem. There's a lack of a MIP formulation as well as an efficient heuristic that could solve large instances of this problem.
- **The Capacitated VRP-D with Time Window** Again, as a logic evolution, the CVRP-D with Time Window is the next promising research direction since it would reflect closely the real-world transportation problem where a fleet of pairs of truck and drone would be responsible for delivering parcels to customers with a fixed time window for each of them.

BIBLIOGRAPHY

- [ABS16] Niels Agatz, Paul Bouman, and Marie Schmidt. "Optimization approaches for the traveling salesman problem with drone." In: *ERIM Report Series Reference No. ERS-2015-011-LIS* (2016).
- [ABS18] Niels Agatz, Paul Bouman, and Marie Schmidt. "Optimization approaches for the traveling salesman problem with drone." In: *Transportation Science* 52.4 (2018), pp. 739–103.
- [App+06] David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. *Concorde TSP solver*. 2006.
- [BA03] Barrie M Baker and MA Ayechev. "A genetic algorithm for the vehicle routing problem." In: *Computers & Operations Research* 30.5 (2003), pp. 787–800.
- [Ban13] Steve Banker. *Amazon And Drones – Here Is Why It Will Work*. [Accessed November 13th, 2017]. 2013. URL: <http://www.forbes.com/sites/stevebanker/2013/12/19/amazon-drones-here-is-why-it-will-work/>.
- [BAS17] Paul Bouman, Niels Agatz, and Marie Schmidt. "Dynamic Programming Approaches for the Traveling Salesman Problem with Drone." In: *Erasmus Research Institute of Management - ERIM Research Paper Series* (2017).
- [CS17] John Gunnar Carlsson and Siyuan Song. "Coordinated logistics with a truck and a drone." In: *Management Science* (2017).
- [Cat+14] Diego Cattaruzza, Nabil Absi, Dominique Feillet, and Thibaut Vidal. "A memetic algorithm for the multi trip vehicle routing problem." In: *European Journal of Operational Research* 236.3 (2014), pp. 833–848.
- [Cut16] Anthony Cuthbertson. *Australia Post to launch drone delivery service*. [Accessed November 13th, 2017]. 2016. URL: <http://www.newsweek.com/australia-post-drone-delivery-service-drones-449442>.
- [Duh+10] Christophe Duhamel, Philippe Lacomme, Christian Prins, and Caroline Prodhon. "A GRASP× ELS approach for the capacitated location-routing problem." In: *Computers & Operations Research* 37.11 (2010), pp. 1912–1923.

- [FR95] Thomas A Feo and Mauricio GC Resende. "Greedy randomized adaptive search procedures." In: *Journal of global optimization* 6.2 (1995), pp. 109–133.
- [FW86] Philip J Fleming and John J Wallace. "How not to lie with statistics: the correct way to summarize benchmark results." In: *Communications of the ACM* 29.3 (1986), pp. 218–221.
- [FM96] Bernd Freisleben and Peter Merz. "A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems." In: *Proceedings of IEEE international conference on evolutionary computation*. IEEE. 1996, pp. 616–621.
- [FP18] Júlia Cária de Freitas and Puca Huachi Vaz Penna. "A variable neighborhood search for flying sidekick traveling salesman problem." In: *International Transactions in Operational Research* (2018).
- [Fre15] Sally French. *Drone delivery is already here and it works*. [Accessed November 13th, 2017]. 2015. URL: <http://www.marketwatch.com/story/drone-delivery-is-already-here-and-it-works-2015-11-30>.
- [GP10] Michel Gendreau and Jean-Yves Potvin. *Handbook of metaheuristics*. Vol. 2. Springer, 2010.
- [GH11] Fred Glover and Jin-Kao Hao. "The case for strategic oscillation." In: *Annals of Operations Research* 183.1 (2011), pp. 163–173.
- [Gre+85] John Grefenstette, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. "Genetic algorithms for the traveling salesman problem." In: *Proceedings of the first International Conference on Genetic Algorithms and their Applications*. Vol. 160. 168. Lawrence Erlbaum. 1985, pp. 160–168.
- [Gro16] Michael Grothaus. *This is How Google's Project Wing Drone Delivery Service Could Work*. [Accessed November 13th, 2017]. 2016. URL: <http://www.fastcompany.com/3055961/fast-feed/this-is-how-googles-project-wing-drone-delivery-service-could-work>.
- [Ha+18a] Quang Minh Ha, Yves Deville, Quang Dung Pham, and Minh Hoàng Hà. "A Hybrid Genetic Algorithm for the Traveling Salesman Problem with Drone." In: *arXiv preprint arXiv:1812.09351* (2018).

- [Ha+18b] Quang Minh Ha, Yves Deville, Quang Dung Pham, and Minh Hoàng Hà. "On the min-cost traveling salesman problem with drone." In: *Transportation Research Part C: Emerging Technologies* 86 (2018), pp. 597–621.
- [HPRMSG09] Hipólito Hernández-Pérez, Inmaculada Rodríguez-Martín, and Juan José Salazar-González. "A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem." In: *Computers & Operations Research* 36.5 (2009), pp. 1639–1645.
- [Ho+08] William Ho, George TS Ho, Ping Ji, and Henry CW Lau. "A hybrid genetic algorithm for the multi-depot vehicle routing problem." In: *Engineering Applications of Artificial Intelligence* 21.4 (2008), pp. 548–557.
- [Hol+92] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [Iba+08] Toshihide Ibaraki, Shinji Imahori, Koji Nonobe, Kensuke Sobue, Takeaki Uno, and Mutsunori Yagiura. "An iterated local search algorithm for the vehicle routing problem with convex time penalty functions." In: *Discrete Applied Mathematics* 156.11 (2008), pp. 2050–2069.
- [KN99] Kengo Katayama and Hiroyuki Narihisa. "Iterated local search approach using genetic transformation to the traveling salesman problem." In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*. Morgan Kaufmann Publishers Inc. 1999, pp. 321–328.
- [KS97] Gerard AP Kindervater and Martin WP Savelsbergh. "Vehicle routing: handling edge exchanges." In: *Local search in combinatorial optimization* (1997), pp. 337–360.
- [Kit+19] Patchara Kitjacharoenchai, Mario Ventresca, Mohammad Moshref-Javadi, Seokcheon Lee, Jose MA Tanchoco, and Patrick A Brunese. "Multiple traveling salesman problem with drones: Mathematical model and heuristic approach." In: *Computers & Industrial Engineering* 129 (2019), pp. 14–30.
- [KB95] George Kontoravdis and Jonathan F Bard. "A GRASP for the vehicle routing problem with time windows." In: *ORSA journal on Computing* 7.1 (1995), pp. 10–23.

- [Lap92] Gilbert Laporte. "The traveling salesman problem: An overview of exact and approximate algorithms." In: *European Journal of Operational Research* 59.2 (1992), pp. 231–247.
- [Lar+99] Pedro Larranaga, Cindy M. H. Kuijpers, Roberto H. Murga, Inaki Inza, and Sejla Dizdarevic. "Genetic algorithms for the travelling salesman problem: A review of representations and operators." In: *Artificial Intelligence Review* 13.2 (1999), pp. 129–170.
- [Liu+19] Yao Liu, Jianmai Shi, Guohua Wu, Zhong Liu, and Witold Pedrycz. "Two-Echelon Routing Problem for Parcel Delivery by Cooperated Truck and Drone." In: (2019).
- [Mar12] Yannis Marinakis. "Multiple phase neighborhood search-GRASP for the capacitated vehicle routing problem." In: *Expert Systems with Applications* 39.8 (2012), pp. 6807–6815.
- [MMP05a] Yannis Marinakis, Athanasios Migdalas, and Panos M Pardalos. "A hybrid genetic—GRASP algorithm using lagrangean relaxation for the traveling salesman problem." In: *Journal of Combinatorial Optimization* 10.4 (2005), pp. 311–326.
- [MMP05b] Yannis Marinakis, Athanasios Migdalas, and Panos M Pardalos. "Expanding neighborhood GRASP for the traveling salesman problem." In: *Computational Optimization and Applications* 32.3 (2005), pp. 231–257.
- [MTZ60] Clair E Miller, Albert W Tucker, and Richard A Zemel. "Integer programming formulation of traveling salesman problems." In: *Journal of the ACM (JACM)* 7.4 (1960), pp. 326–329.
- [Moo+02] Chiung Moon, Jongsoo Kim, Gyunghyun Choi, and Yoonho Seo. "An efficient genetic algorithm for the traveling salesman problem with precedence constraints." In: *European Journal of Operational Research* 140.3 (2002), pp. 606–617.
- [Mur16] Mike Murphy. *This is how Google wants its drones to deliver stuff to you*. [Accessed November 13th, 2017]. 2016. URL: <http://qz.com/670670/this-is-how-google-wants-its-drones-to-deliver-stuff-to-you/>.
- [MC15] Chase C Murray and Amanda G Chu. "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery." In: *Transportation Re-*

- search Part C: Emerging Technologies* 54 (2015), pp. 86–109.
- [MR19] Chase Murray and Ritwik Raj. “The Multiple Flying Sidekicks Traveling Salesman Problem: Parcel Delivery with Multiple Drones.” In: *Available at SSRN* 3338436 (2019).
- [New13] CBS News. *Amazon unveils futuristic plan: Delivery by drone*. [Accessed November 13th, 2017]. 2013. URL: <http://www.cbsnews.com/news/amazon-unveils-futuristic-plan-delivery-by-drone/>.
- [New16] CTV News. *Japanese company launches drone delivery service on the golf course*. [Accessed November 13th, 2017]. 2016. URL: <http://www.ctvnews.ca/sci-tech/japanese-company-launches-drone-delivery-service-on-the-golf-course-1.2875191>.
- [OSH87] IM Oliver, DJd Smith, and John RC Holland. “Study of permutation crossover operators on the traveling salesman problem.” In: *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlbaum Associates, 1987. 1987.
- [ORHo6] Beatrice Ombuki, Brian J Ross, and Franklin Hanshar. “Multi-objective genetic algorithms for vehicle routing problem with time windows.” In: *Applied Intelligence* 24.1 (2006), pp. 17–30.
- [PSO13] Puca Huachi Vaz Penna, Anand Subramanian, and Luiz Satoru Ochi. “An iterated local search heuristic for the heterogeneous fleet vehicle routing problem.” In: *Journal of Heuristics* 19.2 (2013), pp. 201–232.
- [Pog16] David Pogue. *Exclusive: Amazon Reveals Details About Its Crazy Drone Delivery Program*. [Accessed November 13th, 2017]. 2016. URL: <https://www.yahoo.com/tech/exclusive-amazon-reveals-details-about-1343951725436982.html>.
- [Poi18] Stefan Poikonen. “Multi-visit Drone Routing Problem.” In: (2018).
- [PWG17] Stefan Poikonen, Xingyin Wang, and Bruce Golden. “The vehicle routing problem with drones: Extended models and connections.” In: *Wiley Networks journal* 70.1 (2017).

- [Pon16] Andrea Ponza. "Optimization of Drone-assisted parcel delivery." In: *Master thesis* (2016). [Accessed November 13th, 2017]. URL: http://tesi.cab.unipd.it/51947/1/Andrea_Ponza_-_Analisi_e_optimizzazione_nell'uso_di_droni_per_la_consegna_di_prodotti.pdf.
- [Pot96] Jean-Yves Potvin. "Genetic algorithms for the traveling salesman problem." In: *Annals of Operations Research* 63.3 (1996), pp. 337–370.
- [Pri04] Christian Prins. "A simple and effective evolutionary algorithm for the vehicle routing problem." In: *Computers & Operations Research* 31.12 (2004), pp. 1985–2002.
- [Pri09a] Christian Prins. "A GRASP× evolutionary local search hybrid for the vehicle routing problem." In: *Bio-inspired algorithms for the vehicle routing problem*. Springer, 2009, pp. 35–53.
- [Pri09b] Christian Prins. "Two memetic algorithms for heterogeneous fleet vehicle routing problems." In: *Engineering Applications of Artificial Intelligence* 22.6 (2009), pp. 916–928.
- [PLR09] Christian Prins, Nacima Labadi, and Mohamed Reghioui. "Tour splitting algorithms for vehicle routing problems." In: *International Journal of Production Research* 47.2 (2009), pp. 507–535.
- [Riz+07] Andrea Emilio Rizzoli, Roberto Montemanni, Enzo Lucibello, and Luca Maria Gambardella. "Ant colony optimization for real-world vehicle routing problems." In: *Swarm Intelligence* 1.2 (2007), pp. 135–151.
- [RSL77] Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis II. "An analysis of several heuristics for the traveling salesman problem." In: *SIAM journal on computing* 6.3 (1977), pp. 563–581.
- [Too16] Amar Toor. *This startup is using drones to deliver medicine in Rwanda*. [Accessed November 13th, 2017]. 2016. URL: <http://www.theverge.com/2016/4/5/11367274/zipline-drone-delivery-rwanda-medicine-blood>.
- [TV03] Paolo Toth and Daniele Vigo. "The granular tabu search and its application to the vehicle-routing problem." In: *Informatics Journal on computing* 15.4 (2003), pp. 333–346.

- [Uld+90] Nico LJ Ulder, Emile HL Aarts, Hans-Jürgen Bandelt, Peter JM Van Laarhoven, and Erwin Pesch. "Genetic local search algorithms for the traveling salesman problem." In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1990, pp. 109–116.
- [Vid16] Thibaut Vidal. "Split algorithm in $O(n)$ for the capacitated vehicle routing problem." In: *Computers & Operations Research* 69 (2016), pp. 40–47.
- [Vid+12] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems." In: *Operations Research* 60.3 (2012), pp. 611–624.
- [Vid+13a] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows." In: *Computers & OR* 40.1 (2013), pp. 475–489.
- [Vid+13b] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows." In: *Computers & operations research* 40.1 (2013), pp. 475–489.
- [Vid+14] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. "A unified solution framework for multi-attribute vehicle routing problems." In: *European Journal of Operational Research* 234.3 (2014), pp. 658–673.
- [VT99] Christos Voudouris and Edward Tsang. "Guided local search and its application to the traveling salesman problem." In: *European journal of operational research* 113.2 (1999), pp. 469–499.
- [WPG16] Xingyin Wang, Stefan Poikonen, and Bruce Golden. "The vehicle routing problem with drones: several worst-case results." In: *Optimization Letters* 11.4 (2016), pp. 1–19.
- [WS19] Zheng Wang and Jiuh-Biing Sheu. "Vehicle routing problem with drones." In: *Transportation Research Part B: Methodological* 122 (2019), pp. 350–364.

- [Woh14] Marcus Wohlsen. *The Next Big Thing You Missed: Amazon's Delivery Drones Could Work—They Just Need Trucks*. [Accessed November 13th, 2017]. 2014. URL: <http://www.wired.com/2014/06/the-next-big-thing-you-missed-delivery-drones-launched-from-trucks-are-the-future-of-shipping/>.