UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA GESTIONALE

TESI DI LAUREA MAGISTRALE

OPTIMIZATION OF DRONE-ASSISTED PARCEL DELIVERY

Relatore italiano: Prof.ssa DARIA BATTINI

Relatore danese: Prof. STEFAN RØPKE

Laureando: ANDREA PONZA

Matricola 1079510

ANNO ACCADEMICO 2015-2016

To my family

Solutions nearly always come from the direction you least expect, which means there's no point trying to look in that direction because it won't be coming from there.

— Douglas Noël Adams, *The Salmon of Doubt*

Abstract

Cooperation between a truck and a drone for last-mile delivery is gaining momentum and is shaping up to be a key aspect to look into for a competitive and innovative company in the courier business. Among the various ways a problem involving drone-assisted parcel delivery can be formulated, the FSTSP by Murray and Chu [Murray and Chu, 2015] describes a variant of the classical traveling salesman problem where the drone is launched from the truck, goes to deliver goods to a customer and then rendezvouses with the truck in a third customer location. While the drone flies the sortie, the truck is free to deliver to other customers so long as the drone has enough battery to hover waiting for the vehicle. This mixed integer programming formulation of the problem is here solved with the Simulated Annealing metaheuristic, an algorithm proven to be able to provide good solutions for real world applications. Furthermore a numerical analysis helps investigate the tradeoffs of using faster drones or drones with longer autonomy, more iterations in the algorithm and also an analysis on the distribution of the objective function. A comparison is also made with Record-to-Record Travel to better assess the differences in performance for the SA versus the RRT. All these analyses show that significant savings are associated with using the combination of truck and drone instead of truck-only delivery.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In Chapter 2.7 of the book "Development strategies for the postal sector: an economic perspective" [Union, 2014] a thorough explanation of the parcel post sector is presented with data starting from 2006 and ending in 2011. Most of the book is centered towards finding current trends in the postal sector, build models to represent how data could behave in the future and show opportunities for the evolution of the market itself.

An important slice of the market is shown to be shifting its logistics towards new positionings to take advantage of the recent growths lead by globalization and e-commerce, to further develop some of the segments of their business such as packets, parcels and express deliveries.

Looking at the data collected by UPU, it can be seen that the even during the financial crisis of 2008, the global parcel-post volume for domestic and international service continued its positive development across all the world regions and particularly in Eastern Europe. Expansion is the keyword in these years for this business, with an increment of almost 30% for ordinary domestic and international parcels. A definite slowdown is present in the increments starting from 2008, and recent figures show that this could eventually take to a stagnation of postal revenues. Fact is, that this would not be the case if countermeasures were taken to maybe try and reinvent the current business models to respond to the fast-changing postal business environment.

One of the most innovative ideas was proposed by Amazon's CEO Jeff Bezos, who announced on December 1st, 2013 that the world's largest e-commerce company was testing drone parcel delivery. Nobody was at the time expecting an announcement involving drone deliveries, and there was really minor talk surrounding the topic prior to that event, as can be seen by a quick search done in Google Trends, visible in Fig. 1.1.

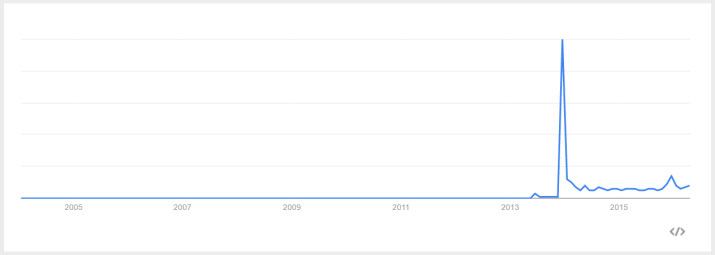


Fig. 1.1: History of the search results for "drone delivery" in the past years

The market was pushing towards quicker delivery times and lower courier costs, as indicated by the migration of customers towards more cost-effective and non-premium services [Cobweb Information Ltd, 2013]. Hardly ever, in the B2C side of the business at least, a customer would choose to pay for same-day and next-day courier and express services for its online order. Even more so, the online side of the business was showing year-on-year growth rates around 14% [Onl, 2012], and DHL reported that a good 25% of its parcels resulted from online retailiing [Onl, 2008].

With this in mind, a drone doesn't seem to be the first thing to come to mind to solve these necessity of reducing costs and speeding up delivery times. In those years, drones were almost always used for military or surveillance purposes, and only in very recent times they made it to the consumer market for civilian applications. Nonetheless, or maybe due to this very reason, drones have evolved to have price tags two orders of

magnitude inferior to the price of a truck or a tir, and show average and top speeds well above the ones provided by these slow means of road-constrained transportation. Due to these reasons, it can be inferred that savings coming from the usage of drones must be significant, both in terms of truck cost and in terms of delivery times.

When Bezos' announcemente was made, however, many still rushed in dismissing it as a PR stunt carefully devised for Cyber Monday [Little, 2013, Ball, 2013]. After little more than a couple of years, contrary to Bezos's predictions in that same announcement, drone deliveries are already a reality like it can be seen in DHL's [Hern, 2015], or Matternet's case [French, 2015a, Evans, 2015]. Technology, competition, research and commoditization have driven the market for these Unmanned Aerial Vehicles (UAVs) to over 6 billions and in continuous growth [Cap, 2015, French, 2015b]. Companies all over the world have started meddling with drone deliveries [Smiley, 2015]: among these are also the previously cited German courier company and world's largest DHL, with its Parcelcopter, Google with its X-Labs' "Project Wing" [Madrigal, 2014], FedEx [Bermingham, 2014] and many more (see Fig. 1 for some examples).

Fig. 1.2: HorseFly's drone and truck combo

Fig. 1.3: Amazon's Prime Air service

Fig. 1.4: DHL's Paketkopter

Fig. 1.5: Google X's Project Wing

Harmless as they might seem, drones have also occupied the news for controversial events:

- Human-less aircraft serve well in some countries as military drones, and are mainly used in "missions characterized by the three Ds: dull, dirty, and dangerous", as was already done in the in the 90s [Capt. Tice, 1991];
- Over to the civilian side, drones have been armed with guns, machine guns, and even flamethrowers to "test the technology" in some recent events that made the news [Popper, 2015b, Koebler, 2013];
- Weapon use is certainly the biggest concern, but on the side of privacy, the high tech Peeping Tom has also hit the news by starting to use drones to his advantage [Tremonti, 2015].

In all this controversy, FAA has tried to regulate the problem in multiple ways, only recently agreeing to give delivery services some airspace [Duncan, 2015a, Duncan, 2015b] to permit limited testing that will still provide ground for many companies to grow a business out of.

Tab. 1.1: Drones and trucks exhibit complementary features

	truck	drone
speed	low	**high**
weight	heavy	**light**
capacity	**many**	one
range	**long**	short

Success came after a yearlong struggle, but it was surely worth it: as noted by [Agatz et al., 2015], drones offer a service complementary to the one offered by trucks (see Tab. 1.1) without requiring a costly human pilot, and are moreover detached from the constraint of journeys thorugh the road network system. Overall, the advantages displayed outsize the drawbacks, making drone delivery an interesting topic for further research. As a matter of fact this work aims to find better and improved methods to solve the problems concerned with the scheduling of combined truck and drone deliveries by using the *Simulated Annealing* (SA) metaheuristic framework to find good delivery tours in a reasonable amount of time. Results are expected to show an improvement over the optimal truck-only tour for the same nodes, other than computations done in a reasonable amount of time and iterations of the SA to have practical use for this method.

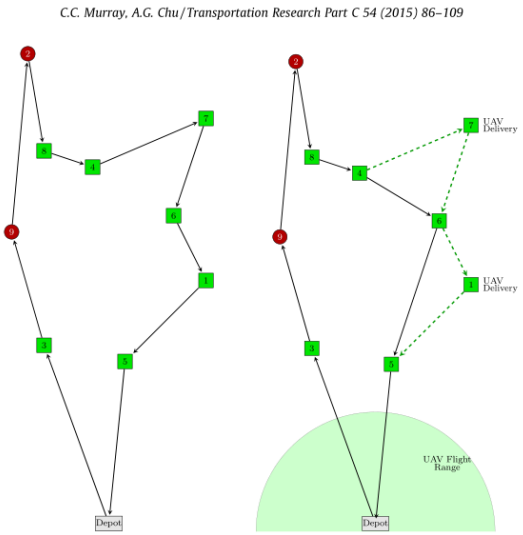


Fig. 1.6: Difference between a normal TSP and the FSTSP (by Murray and Chu)

To sum up and quickly give an outline of the problem with whom this thesis is concerned, it all starts with Murray and Chu (Fig. 1.6) and their FSTSP. There it is described a variant of the classical TSP where the drone is firstly launched from the truck, proceeds to deliver goods to a customer and finally joins back the truck in a third location. While the drone flies in a sortie, the truck can move to deliver to other customers on the map, but it has to stop and go catch the drone at the rendezvous node before this one finishes its endurance. Combinations of drone sorties and truck deliveries make it possible to cover all the customers, defining thus the problem in a way such that the drone is a "flying sidekick" to the truck, hence the name. The problem is therefore one of minimizing the

total duration of the tour that can be built with these two building blocks.

1.1 Thesis plan

The thesis is organized as follows:

IN CHAPTER 2 can be found an analysis of previous works on drones and drone deliveries: the literature is still in its early phases, but it is growing. Likewise, related literature is also discussed.

IN CHAPTER 3 the mathematical model, inspired by [Murray and Chu, 2015], is described starting from the notation utilized and progressing with the model itself and a thorough explanation of the constraints adopted. After briefly touching the main changes from the original version, a subsection is dedicated to further understanding of how an example timeline is computed from top to bottom for both the truck and the drone.

IN CHAPTER 4, after briefly reviewing different interesting solution approaches, metaheuristics are discussed, leading ultimately to the reasoned choice of the Simulated Annealing as the solution method to attempt. The SA implementation is then painstakingly described from the data structures chosen to how the solutions are accepted, and then also how a move is picked and how it is executed. The most detail-oriented sections are in this case the ones describing in toto one of the identified moves and, subsequently, how the solution is evaluated via a process called *delta evaluation*.

IN CHAPTER 5 the reader can first and foremost find an explanation of the instances used in the experimental phase that follows and how they were generated. The chapter then goes on with the parameter tuning: to work better, a Simulated Annealing needs parameters like number of iterations, start temperature, final temperature, etc. to be adapted to suit the scale of the problems that will be solved. Firstly, some pre-tuning will give a rough idea of where the parameters should stay at. Then, those approximate parameters are used to find a number of representative problems, for higher precision. With the problems chosen, it is ultimately time to set in stone the best temperatures for the representative instances, which will serve for all of them in the main experiment. After a good set of parameters is successfully obtained, the experiments can be performed on all the instances: following the tuning is an explanation of which examinations were performed, what the results are and how they can be interpreted.

Chapter 2

Literature review

The literature review is articulated in three sections: Delivery applications 2.1 is concerned with parcel delivery in the Operations Research sense, Surveillance applications 2.2 is about two examples of disaster management which can also be applied to surveillance, and Military applications 2.3 deals with drone usage in the army and for general military use.

2.1 Delivery applications

A substantial body of literature exists on the TSP and the VRP. Although neither of the problems is applicable as-is to the examined problem, the FSTSP is a generalization of the TSP and future work could expand it based on the VRP. Notably, the TSP has many variants, like the TSP with Time Windows or the TSP with Precedences: the FSTSP is nothing less than another one of the possible variants.

A particularly interesting read on the Traveling Salesman Problem is certainly the work of [Applegate et al., 2011] which brilliantly sums up what the TSP is and describes classical and exotic methods for its solution. The discussion in this paper, written by the creators of the TSP solver Concorde, begins from the definition of the problem, but quite sharply takes a turn to the artistic side of it and before digressing about exact methods and heuristics, it takes an interesting sidestep in the analysis of how a solution should look like and on sociological and biological perspectives on finding solutions. For a more strictly focused read, [Jünger et al., 1995] would be the book to go to: here all the solution methods are given more detailed and thorough theoretical and practical explanations. Some heuristic methods to solve the TSP are further discussed by [Nilsson, 2003], who briefly highlights constructive heuristics, improvement methods and also metaheuristics in his paper.

Another well-known problem, the Vehicle Routing Problem, is extensively presented along with many of its variants in [Toth and Vigo, 2014]. It isn't directly related to this work, but should be definitely taken in consideration for further research on the topic since most real-life applications involve multiple trucks, which in light of the work done here, could consider having one or multiple drones each on board to expand their delivery range and shorten their delivery times.

A specific VRP variant that can be associated with the study is *VRP with Synchronization Constraints*: in [Drexl, 2012] a thorough review of some of the possibilities for these constraints undergo an intensive study, as they represent a topic of growing interest in the field. In particular the paper involves a threefold objective of classifying the different types of synchronization, discussing exact and heuristic solutions to the problem and identifying promising algorithmic solution approaches. The main influence of this VRP variant in this thesis is evidently shown in the synchronization constraints devised to make drone launches and rendezvous work as intended by the problem's ideators. According to Drexl, the type of synchronization in the Murray's and Chu's work would be classified in "operation synchronization" as the drone launch would be the operation to facilitate the drone delivery at a different location, and the rendezvous with the truck would facilitate the use of the drone for further deliveries.

As a concrete example of synchronization, the paper also enstablishes what is known as the *Vehicle Routing Problem with Trailers and Transshipments* or also *Truck and Trailer*

VRP. In this specific problem, there is an important distinction with two different types of vehicles: trucks and trailers. Trucks can move freely, they are autonomous vehicles, and instead trailers are nonautonomous vehicles, which can move only if pulled by an autonomous vehicle. This last constraint can, with a stretch of imagination, resemble the semiautonomous nature of the drone, constrained to only be able to fly for short distances (compared to the truck) due to a limited endurance given by the battery.

The work which seems to be most closely related to the problem at hand without directly involving drones is the VRP variant presented by [Lin, 2011]: she addresses a pickup and delivery problem, using a heavy resource (e.g. a truck) which may carry both delivery items and one or more units of the lighter resource (in her case, she was referring to foot couriers and scooters) on its route assignment. Other than using more trucks at a time, this problem differs from the one examined here in that, due to its limited payload , the drone can only visit one customer per flight and some of the deliveries might be infeasible due to too heavy parcels. Moreover, drones have a limited battery supply which translates to flight endurance.

This thesis is based on the FSTSP formulation by [Murray and Chu, 2015]: the problem as thereby defined describes the optimization of a single truck – single drone scenario where the total time spent in the tour has to be minimized. The reader can refer to Sec. 3.4 for the main changes from that model to the one presented in this work. In the same paper a different problem is also defined, the Parallel Drone Scheduling TSP, which acts as a complementary idea to the FSTSP: when a lot of customers are located in the vicinity of the depot and only some of them are at a greater distance, then the drone can serve the former set and the truck can independently work the latter one. Murray and Chy also devise one heuristic for each of the two formulations: regarding the one for the FSTSP, some of the possible improvements that could be made to it are treated in Sec. 4.1.3.

A different formulation for the problem is the TSP-D by [Agatz et al., 2015] where the main difference is that they assume that truck and drone travel both on the road network. This has multiple advantages and also some drawbacks. The main advantages are that they manage to develop heuristics with approximation guarantees and provide a bound on the maximum possible gains of the truck and drone coupling over a tradition truck-only system. With the ability to get a lower bound on the optimal solution there is the double edge of tying the drone to the ground network, without taking full advantage of its speed and ability to take shortcuts via euclidean distances. Moreover, on a neutral note, another difference between the two formulations is that the TSP-D allows the drone's launch and rendezvous to be the same point, whereas the FSTSP bounds them to be different nodes. For this new formulations new heuristics, as previously anticipated, are built using the *route first cluster second* procedure.

The name TSP-D is also used in [Ha et al., 2015], albeit with a formulation that excludes the possibility to have a launch and rendezvous in the same customer node. In this paper, a pair of heuristics are provided: a *cluster first route second* and a *route first cluster second*. They also compare their formulation and heuristics with both optimal TSP tours and the FSTSP on their instances : results show that out of 8 instances, the FSTSP outperforms their TSP-D 6 times with an average gap ($Gap\ \% = \frac{TSP-D_{Obj}-FSTSP_{Obj}}{FSTSP_{Obj}}$) of 30%. In the two instances that outperform the FSTSP, the gap is instead respectively 4.42% and 0.49%.

Lastly, on the side of closely related papers on this subject, [Mathew et al., 2015] discusses a *Heterogeneous Delivery Problem* (HDP) as a discrete optimal path planning problem that seeks to minimize the total cost of deliveries in routes computed for a truck and a quadrotor. The authors seem to be unaware of the existence of the previously cited body of work and thus devise a conversion for their HDP to obtain a Generalized

Traveling Salesman Problem (GTSP) and be able to use already available solvers. An important difference in the paper is in splitting the objective function to take into three sums of costs: one for quadrotor flight edges, one for truck street edges and one for when the drone is docked on the truck and they travel together. They also define the *Multiple Warehouse Delivery Problem* (MWDP), a variant to the HDP which only considers drones and their ability to fulfill warehouse requests to exchange parts between themselves. They also provide a conversion for this problem to a GTSP to solve this problem with known solvers.

[Boone et al., 2015] creates a new clustering approach and analyzes the method for solving the cluster. This leads to much better results over their baseline method, *K-means clustering*. They use this knowledge to solve a variation of the Multiple TSP (MTSP) which involves many UAVs. It doesn't seem to be directly relatable to the work done here, and the commonalities end at the mention of UAVs, but it is nonetheless a way to optimize drone-assisted parcel delivery. It is not clear if the paper refers to military or civil drones, and due to this ambiguity it might be usable in both scenarios.

Tab. 2.1 presents an overview of all the formulations with delivery application. In it, "Formulation" highlights the one or two models proposed in the respective paper, or the type they belong to if no name is provided. The third and fourth columns regard the number of vehicles of each type that are allowed by that formulation, while the "Launch = Rendez" column specifies if the formulation allows a sortie to have launch and rendezvous in the same node. "Proposed solution" sums up the method(s) the author suggests to use for solving their formulation.

Tab. 2.1: Overview of the various formulations for the drone delivery problem

Paper	Formulation	# trucks	# drones	Launch = Rendez	Proposed solution
[Murray and Chu, 2015]	FSTSP	1	1	Not Avail	Integer Programming, Savings, Nearest Neighbour, Sweep
	PDSTSP	1	n	Avail	Combination of (Integer Programming, Savings, Nearest Neighbour) with (Integer Programming, Longest Processing Time)
[Agatz et al., 2015]	TSP-D	1	1	Avail	Route first cluster second
[Ha et al., 2015]	Column Generation	1	1	Not Avail	Cluster first route second, Route first cluster second
[Mathew et al., 2015]	HDP	1	1	Avail	Reduction to GTSP
	MWDP	0	1	Not Avail	Reduction to TSP
[Boone et al., 2015]	MTSP	0	n	Not Avail	Clustering with Nearest Neighbour and 2-Opt

2.2 Surveillance applications

The two next papers, [Quaritsch et al., 2010, Shang et al., 2014], delve in the field of disaster management, and use civil drones primarily as a means to obtain pictures of disaster areas. The first paper provides a first formulation of the minimum problem to generate an overview map of an area, which incidentally is similar to the problem of covering the area with a sensor network. The second paper builds on the research done in the first one and provides improved results using the shortest path algorithm. Their problem is a different one from the one argued in this thesis, but these papers provide nonetheless an insight on a possible promising and applicable solution method.

2.3 Military applications

The next papers are specifically about military drones, where the literature is much more ample and exploration has been active for decades now. The main non obvious difference of military drones from civil ones is that they are provided with a fuel supply instead of a rechargeable battery, so both endurance and speed are vastly superior.

[Savuran and Karakaya, 2015] is the closest military application to the ones discussed here with a combination of drone and truck. In particular their problem is about a UAV and its carrier that have to visit a set of targets: the calculation of the drone take-off and land-on locations is crucial in minimizing the total tour length. The drone in this case has multiple target nodes, and the carrier moves continuously while the drone flies. The problem is solved via Genetic Algorithm and compared to Nearest Neighbour solutions: the solutions obtained with the former are around 90% shorter overall.

The final four papers are specifically related to Wireless Sensor Network and provide methods to avoid radars or communicate with low-energy beacons so as to preserve their battery life to optimize their usage duration. In particular, Bortoff [Bortoff, 2000] generates stealthy drone-only paths through enemy radars via a two step algorithm. The algorithm itself could be useful in avoiding obstacles in parcel deliveries while still minimizing path length and time duration. A similar issue is tackled in [Zheng et al., 2005], who builds a planner for single and multiple UAV missions to also comply with a set of given constraints to the tours: minimum route length, minimum flight altitude to avoid obstacles or radars, and others.

The problem [Kashuba et al., 2015] analyze is that of performing sensor data gathering in an efficient way that minimizes route cost while providing adequate coverage. The provided Path Planning algorithm, based on the Shortest Path algorithm, could be used in parcel delivery with drones similar in structure to the ones developed in Google X's Project Wing, which have a vertical delivery configuration that seems to replicate how a military drone behaves, only adding a slight pause for the delivery to actually take place.

Finally [Richards et al., 2002] addresses trajectory planning for a fleet of UAVs devising two methods to solve the problem: a combined method which tackles the solution the problem with a single MILP formulation combining the need for trajectory designing and task assignment; the second method, an approximate one, yields a solution in much faster times by using euclidean distances to estimate finishing times. They could both relate to the FSTSP in the obstacle avoiding part of it and could provide interesting insight on the use of multiple drones per truck.

Tab. 2.2 summarizes the review, categorizing the papers to show their main differences.

Tab. 2.2: Summary table of the literature reviewed

Paper	Categories					
	Civil drones	Military drones	Parcel delivery	Wireless Sensor Network	Drone Only	Drone and Truck
[Applegate et al., 2011]			X			
[Jünger et al., 1995]			X			
[Nilsson, 2003]			X			
[Toth and Vigo, 2014]			X			
[Drexl, 2012]			X			
[Lin, 2011]			X			
[Murray and Chu, 2015]	X		X			X
[Agatz et al., 2015]	X		X			X
[Ha et al., 2015]	X		X			X
[Mathew et al., 2015]	X		X		X	X
[Boone et al., 2015]	X	X	X		X	
[Quaritsch et al., 2010]	X			X	X	
[Shang et al., 2014]	X				X	
[Savuran and Karakaya, 2015]		X				X
[Bortoff, 2000]		X		X	X	
[Zheng et al., 2005]		X		X	X	
[Kashuba et al., 2015]		X		X	X	
[Richards et al., 2002]		X			X	

Chapter 3

Flying Sidekick Traveling Salesman Problem

In this chapter the mathematical model to the problem at hand gets introduced. First of all in Model notation (3.1) there is a brief explanation of the notation used in the model later presented. Then in Mathematical Model (3.2) the actual model is presented. In Model description (3.3) is provided a brief description of every constraint in the model. This is followed by Changes from the original model (3.4) where some important changes and modifications from the original model, on which this is very strongly based upon, are thorougly discussed.

In Further considerations on the variables (3.5) some space is dedicated to how part of the variables are calculated and to related assumptions. Finally, in Visual example of a timeline (3.6), the reader is provided with a timeline visual example as well as a step-by-step evaluation of a the solution.

3.1 Model notation

This is the notation used here to describe the mathematical model and the same used in the Mixed Integer Linear Programming (MILP) formulation in OPL:

- $C = \{1, 2, ..., c\}$: set of all customers;
- $C^D \subseteq C$: set of the serviceable customers for the drones;
- $N = \{0, 1, ..., c+1\}$: set of all nodes (0, c+1 are the depot);
- $N_0 = \{0, 1, ..., c\}$: departure nodes for a vehicle;
- $N_+ = \{1, 2, ..., c+1\}$: arrival nodes for a vehicle;
- $\tau^T_{i,j}$: travel time from node $i \in N$ to node $j \in N$ for vehicles;
- $\tau^D_{i,j}$: travel time from node $i \in N$ to node $j \in N$ for drones;
- SL: service time before the drone's launch;
- SR: service time after the drone's rendezvous;
- E: flight endurance of the drone, that is the time it can stay afloat on a full battery charge;
- M: big enough number;
- $\langle \text{launch}, \text{deliv}, \text{rendez} \rangle$: tuple used to represent feasible drone sorties, comprised of two flight paths respectively from launch to delivery and from delivery to rendezvous, such that launch $\in N_0$, deliv $\in \{C^D : \text{deliv} \neq \text{launch}\}$ and rendez $\in \big\{N_+ : \text{rendez} \neq \text{launch}, \text{rendez} \neq \text{deliv}, \tau^D_{\text{launch,deliv}} + \tau^D_{\text{deliv,rendez}} \leq E\big\}$;
- F: set of all the possible tuples defined in the previous bullet point; as such, it represents all the possible sorties the drone can feasibly make;
- $x_{i,j}$: binary decision variable that equals 1 if and only if the vehicle travels from $i \in N_0$ to $j \in N_+$ with $i \neq j$;

- $y_{i,j,k}$: binary decision variable that equals 1 if and only if the drone travels from $i \in N_0$ to $j \in C^D$ and back to $k \in N_+$ with $\langle i,j,k \rangle \in F$
- t_j^T: decision variable for the time at which the vehicle is ready to leave $j \in N$;
- t_j^D: decision variable for the time at which the drone is ready to leave $j \in N$;
- $p_{i,j}$: auxiliary binary decision variable that equals 1 if and only if $i \in N_0$ is visited by the truck before $j \in C$ (with $i \neq j$);
- u_i: auxiliary integer variable that specifies the position of node $i \in N_+$ in the vehicle's path.

3.2 Mathematical Model

Here's the adapted mathematical model heavily inspired by Murray and Chu's *FSTSP* [Murray and Chu, 2015].

$$\min\ t_{c+1}^T \tag{3.2.1}$$

$$\text{s.t.:} \sum_{\substack{i \in N_0 \\ i \neq j}} x_{i,j} + \sum_{\substack{i \in N_0 \\ i \neq j}} \sum_{\substack{k \in N_+ \\ \langle i,j,k \rangle \in F}} y_{i,j,k} = 1 \qquad \forall j \in C \tag{3.2.2}$$

$$\sum_{j \in N_+} x_{0,j} = 1 \tag{3.2.3}$$

$$\sum_{i \in N_0} x_{i,c+1} = 1 \tag{3.2.4}$$

$$u_i - u_j + 1 \leq (c+2)(1 - x_{i,j}) \qquad \forall i \in C, j \in N_+, j \neq i \tag{3.2.5}$$

$$\sum_{\substack{i \in N_0 \\ i \neq j}} x_{i,j} = \sum_{\substack{k \in N_+ \\ k \neq j}} x_{j,k} \qquad \forall j \in C \tag{3.2.6}$$

$$\sum_{\substack{j \in C^D \\ j \neq i}} \sum_{\substack{k \in N_+ \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \leq 1 \qquad \forall i \in N_0 \tag{3.2.7}$$

$$\sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C^D \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \leq 1 \qquad \forall k \in N_+ \tag{3.2.8}$$

$$2y_{i,j,k} \leq \sum_{\substack{h \in N_+ \\ h \neq i}} x_{i,h} + \sum_{\substack{l \in N_0 \\ l \neq k}} x_{l,k} \qquad \forall i \in N_0, j \in C^D, k \in N_+ \tag{3.2.9}$$

$$y_{0,j,k} \leq \sum_{\substack{h \in N_0 \\ h \neq k}} x_{h,k} \qquad \forall j \in C^D, k \in N_+, \langle 0,j,k \rangle \in F \tag{3.2.10}$$

$$t_i^D \geq t_i^T - M \left(1 - \sum_{\substack{j \in C^D \\ j \neq i}} \sum_{\substack{k \in N_+ \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) \qquad \forall i \in C \tag{3.2.11}$$

$$t_i^T \geq t_i^D - M \left(1 - \sum_{\substack{j \in C^D \\ j \neq i}} \sum_{\substack{k \in N_+ \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) \qquad \forall i \in C \tag{3.2.12}$$

$$t_k^D \geq t_k^T - M \left(1 - \sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C^D \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) \qquad \forall k \in N \tag{3.2.13}$$

$$t_k^T \geq t_k^D - M \left(1 - \sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C^D \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) \qquad \forall k \in N \tag{3.2.14}$$

$$t_k^T \geq t_h^T + \tau_{h,k}^T + SL \left(\sum_{\substack{l \in C^D \\ l \neq h}} \sum_{\substack{m \in N_+ \\ \langle h,l,m \rangle \in F}} y_{h,l,m} \right) + SR \left(\sum_{\substack{i \in N_0 \\ i \neq k}} \sum_{\substack{j \in C^D \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) + \\ - M \left(1 - x_{h,k} \right) \qquad \forall h \in N_0, k \in N_+, k \neq h \tag{3.2.15}$$

$$t_j^D \geq t_i^D + \tau_{i,j}^D + SL - M \left(1 - \sum_{\substack{k \in N_+ \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) \qquad \forall j \in C^D, i \in N_0, i \neq j \tag{3.2.16}$$

$$t_k^D \geq t_j^D + \tau_{j,k}^D + SR - M \left(1 - \sum_{\substack{i \in N_0 \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) \qquad \forall j \in C^D, k \in N_+, k \neq j \tag{3.2.17}$$

$$t_k^D - t_i^D \leq E + M \left(1 - \sum_{\substack{j \in C^D \\ \langle i,j,k \rangle \in F}} y_{i,j,k} \right) \qquad \forall k \in N_+, i \in N_0 \tag{3.2.18}$$

$$u_i - u_j \geq 1 - (c+2)\, p_{i,j} \qquad \forall i \in C, j \in C, j \neq i \tag{3.2.19}$$

$$u_i - u_j \leq -1 + (c+2)\,(1 - p_{i,j}) \qquad \forall i \in C, j \in C, j \neq i \tag{3.2.20}$$

$$p_{i,j} + p_{j,i} = 1 \tag{3.2.21}$$

$$t_l^D \geq t_k^D - M \left(3 - \sum_{\substack{j \in C^D \\ \langle i,j,k \rangle \in F \\ j \neq l}} y_{i,j,k} - \sum_{\substack{m \in C^D \\ m \neq i \\ m \neq k \\ m \neq l}} \sum_{\substack{n \in N_+ \\ n \neq i \\ n \neq k \\ \langle l,m,n \rangle \in F}} y_{l,m,n} - p_{i,l} \right) \\ \forall i \in N_0, k \in N_+, l \in C, k \neq i, l \neq i, l \neq k \tag{3.2.22}$$

$$t_0^T = 0 \tag{3.2.23}$$

$$t_0^D = 0 \tag{3.2.24}$$

$$p_{0,j} = 1 \qquad \forall j \in C \tag{3.2.25}$$

$$x_{i,j} \in \{0,1\} \qquad \forall i \in N_0, j \in N_+, j \neq i \tag{3.2.26}$$

$$y_{i,j,k} \in \{0,1\} \qquad \forall i \in N_0, j \in C^D, k \in N_+, j \neq i, \langle i,j,k \rangle \in F \tag{3.2.27}$$

$$1 \leq u_i \leq c+2 \qquad \forall i \in N_+ \tag{3.2.28}$$

$$t_i^T \geq 0 \qquad \forall i \in N \tag{3.2.29}$$

$$t_i^D \geq 0 \qquad \forall i \in N \tag{3.2.30}$$

$$p_{i,j} \in \{0,1\} \qquad \forall i \in N_0, j \in C, j \neq i \tag{3.2.31}$$

3.3 Model description

The objective function (3.2.1) is equivalent to $\min\left\{\max\left\{t^T_{c+1}, t^D_{c+1}\right\}\right\}$ due to the implementation of constraints (3.2.13) and (3.2.14) that link the times for rendezvous; its aim is thus to minimize the latest possible time of arrival to depot for either the drone or the vehicle.

Constraint (3.2.2) serves to have one and only one visit to each customer, made by either the drone or the vehicle. Constraints (3.2.3) and (3.2.4) ensure respectively that the truck departs and arrives from and to the depot only once. Constraint (3.2.5) provides the model with a way to eliminate subtours, with further bounds to auxiliary decision variable u_i set by constraint (3.2.28). Constraint(3.2.6) enforces the flow conservation for the truck.

Constraints (3.2.7) and (3.2.8) ensure that the drone can respectively launch and rendezvous at most once from every particular node. Constraint (3.2.9) couples the drone's and truck's movements between customers, because if the drone flies from i to j to k, then the truck needs to pass in i to launch it and in k to collect it. Constraint (3.2.10) is for the special case where i is the depot.

Constraints (3.2.11) and (3.2.12) are imposed for the coordination of launch time of drone and truck, for customer i, in such a way that only the biggest of the two elapsed times for the sortie is chosen. (3.2.13) and (3.2.14) are the analogous for rendezvous time. Both sets ensure that truck and drone depart or arrive at the same time from/to node i or k.

Constraint (3.2.15) takes into account the passing of time for the truck: arriving in node k means the new time is the previous time plus the travel time from h to k plus the service time for the drone if applicable (the last term ensures that if the truck doesn't travel from h to k, the value for the time doesn't get increased). (3.2.16) and (3.2.17) analogously keep account of the drone's flights' times from launch to delivery and from delivery to rendezvous. They essentially are apt to give the correct values to variables t^T_j and t^D_j.

The drone's flight endurance is addressed in (3.2.18): $t^D_k - t^D_i$ is the total flight time, which must be lower or equal to the endurance if the drone travels from i to k delivering to a generic node j. Constraints (3.2.19), (3.2.20) and (3.2.21) determine the proper values for the precedences of the truck. Constraints (3.2.23) and (3.2.24) represent the earliest time at which the truck and drone may leave the depot, respectively. Constraints (3.2.26), (3.2.27), (3.2.28), (3.2.29), (3.2.30) and (3.2.31) specify the domains for the decision variables.

Finally, constraint (3.2.22) prevents launches in l from preceding a rendezvous in k and, to work as intended, it also requires (3.2.25). It is also important to notice that constraint (3.2.22) helps avoiding two dangerous situations in a generic solution, which are in this work dubbed *Prohibited Move 1* and *Prohibited Move 2* (later on PM1 and PM2).

PM1 happens when a drone is launched before it has had a rendezvous, and is visually represented in Figure 3.1.

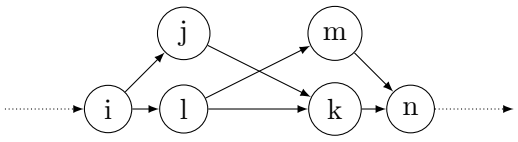


Fig. 3.1: Visual representation of a PM of the first type

If this were to happen, then the M would be nullified since $y_{i,j,k} = 1$, $y_{l,m,n} = 1$ and $p_{i,l} = 1$. This would make the disequation become $t_l^D \geq t_k^D - M\,(3 - 1 - 1 - 1)$, thus enforcing $t_l^D \geq t_k^D$ and rendering this situation infeasible.

PM2 happens when a drone has both a launch and a rendezvous preceded by a launch, and is visually represented in Figure 3.2.

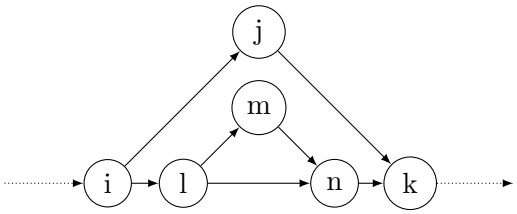


Fig. 3.2: Visual representation of a PM of the second type

If this were to happen, the situation would be the same described for PM1, and the constraint would enforce $t_l^D \geq t_k^D$, creating a contradiction that would render the situation infeasible.

3.4 Changes from the original model

This model is different from [Murray and Chu, 2015]'s model in many ways: to better reflect the author's interpretation of the examined problem, some of the definitions and some of the constraints have been modified. Following is a brief explanation of the most impacting changes apported to that model.

A series of changes were performed to constraints (3.2.15) and (3.2.16) for time accounting. In particular, now SL is added to t_k if the generic sortie $\langle h, l, m\rangle$ is performed, instead of $\langle k, l, m\rangle$: this changes the interpretation of both time variables t_k^T and t_k^D to be the ready-time of respectively the truck and the drone. Time management is thus more consistent and realistic. Moreover, SL is added to (3.2.16) to reflect the change in the previous constraint and give an overall more consistent nature to the model and to how one would expect to account for time passing during the tour.

Constraint (3.2.18) is strengthened by changing $t_j^D - \tau_{i,j}^D$ with the equivalent t_i^D for clearliness and simplicity of reading, thus making the reduction of the number of constraints possible by summing on the generic delivery j performed during the sortie.

3.5 Further considerations on the variables

SL and SR are service times to account for package loading, battery changing, drone recovery, inspection and possibly light service. Due to package loading being the main time-consuming bit, SL is always assumed to be slightly bigger than SR, and for all self generated instances it is assumed $SL = 40\ s$ and $SR = 30\ s$.

Travel time $\tau_{i,j}^T$ is calculated starting from a symmetrical distance matrix and assuming a constant average speed of 35 mph. Similarly, $\tau_{i,j}^D$ is calculated from the same distance matrix assuming a constant average speed of 50 mph. Any τ is calculated beginning from a euclidean $n+2 \times n+2$ distance matrix D of element $d_{i,j}$ as such: $\tau_{i,j}^{T,D} = d_{i,j}/speed^{T,D} * \nu$ where ν is a generic coefficient to account for accelerations, decelerations, traffic lights and road network, assumed to be $\nu = 0.8$ for all of the examined instances. It isn't a precise way to treat the problem, and precision could be improved by getting the real distances in the road network, getting more precise timings for the drone, etc. However, such a precision is deemed unnecessary and uninteresting at this level of research: although it can be done, it is also true that at this stage, this level of approximation seems more

appropriate and still gives meaningful and precious insight into the problem. Deliveries are assumed to be done in constant time for both trucks and drones and are assumed to be accounted for in τ times. The speeds used reflect the best case scenario according to current state of art. In particular, for drone deliveries many standards exist: in the US, the FAA sets a maximum allowed speed of 100 mph [Duncan, 2015a], although for example Amazon wants to use drones of speeds of up to 50 mph [Giz, 2015], and HorseFly (which received the same exemption as Amazon's just months later [Duncan, 2015b]) plans on using similarly powerful drones (see [Hor, 2015, Zito, 2016]). In the UK, FPS seems to have a lower top speed, reported as 35 mph also [FPS, 2015] for drone speeds). Given the constant evolution of technology and competition that will form in such a market, it only seems reasonable to assume the top speeds showed by Amazon and HorseFly as data for the problem.

In a similar fashion, E is calculated starting from Amazon's data [Giz, 2015] where it is stated that its drones will fly in a 10 mi radius from the warehouse: this means that the drone will be able to go a full roundtrip, thus making the conversion from distance units to time units be $E = E_{time} = 2 * E_{dist}/speed$. If the speed is irrealistically assumed to be always the top speed, the result is $E = 1440$ s (24 min) which is still not an irrealistic number, since for example HorseFly states its endurance to be 1800 s (30 min in [Hor, 2015]).This is probably due to Amazon accounting also for some safety factor and approximations in its estimate of 10 mi, thus making $E = 1440$ s be the realistic choice in this work.

M needs a constructive process to find a suitable value to enforce constraints from (3.2.11) to (3.2.18) and (3.2.22). Since it should be greater or equal to the latest possible return time for the drone and the vehicle, the Nearest Neighbour heuristic was used to determine an upper bound on the time required to complete a TSP tour. A route gets built from the depot ($i = 0$) and then visiting the nearest customer to the previous visited until all of them have been visited. In the end the vehicle completes the tour by going back to the depot ($j = c + 1$). M is set to be the total time to do this tour.

3.6 Visual example of a timeline

The timeline for an instance's solution is shown in Figure 3.3 (times are rounded to nearest integer value), in particular the one for `Instance_010.4`'s optimal solution.

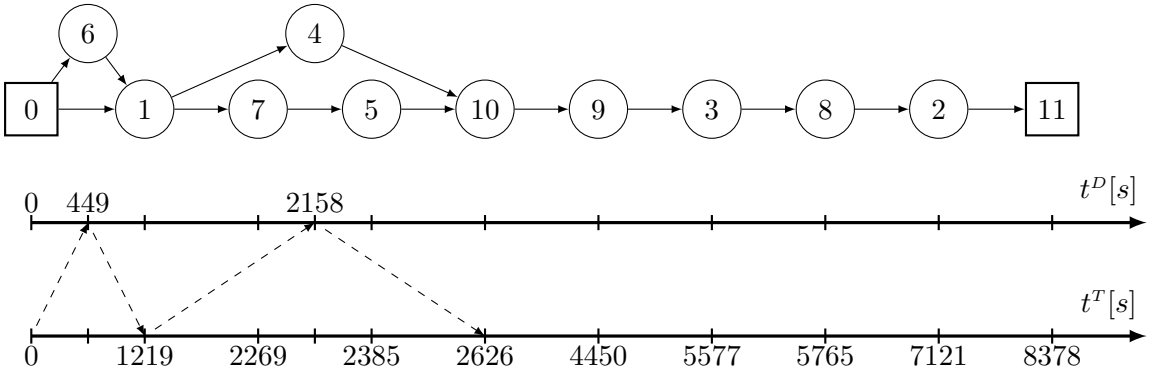


Fig. 3.3: Timeline for the optimal solution of the self-generated `Instance_010.4`

Beginning from the depot, the starting times are set at 0, as per constraints (3.2.23) and (3.2.24). The drone is launched from 0 and travels to 6, which means that the drone time at 6 is the sum of the service at launch plus the time to fly from 0 to 6 (see constraint

(3.2.16)). The drone then travels to 1 to complete the sortie.

Concurrently to this all, the truck has gone from 0 to 1. The time set by the model to be the ready time in 1 for both drone and truck is the maximum between the times to go from 0 to 1 for both vehicles (see constraints (3.2.13) and (3.2.14)). This means that the drone would add to its time at 6, the time to travel from 6 to 1 and receive service once there (constraint (3.2.17) sets this). For the truck, instead, this means adding to its time in 0, the time for the service at launch, the time to travel to 1 via land, and then the service at rendezvous as per constraint (3.2.15).

Only keeping the maximum of the two makes either the drone or the truck wait for the other vehicle, and that's also why this time can't exceed the drone endurance: the UAV can't hover endlessly, it has to be catched before the battery runs out. In this case, the drone has to wait for the truck for the difference between the two members in the max, which is $\left[t_0^T + SL + \tau_{0,1}^T + SR\right] - \left[t_6^6 + \tau_{6,1}^D + SR\right] = [0\ s + 40\ s + 1149\ s + 30\ s] - [449\ s + 665\ s + 30\ s] = 1219\ s - 1144\ s = 75\ s$.

$$
\begin{aligned}
&t_0^T = t_0^D = 0\ s \\
&t_6^D = SL + \tau_{0,6}^D = 40\ s + 409\ s = 449\ s \\
&t_1^T = t_1^D = \max\left\{t_6^6 + \tau_{6,1}^D + SR, t_0^T + SL + \tau_{0,1}^T + SR\right\} = \\
&\quad = \max\{449\ s + 665\ s + 30\ s, 0\ s + 40\ s + 1149\ s + 30\ s\} = 1219\ s
\end{aligned}
$$

The same can be said for the second sortie, which starts from node 1 for both truck and drone. The truck launches the drone after the service, and travels to 7. Then it proceeds to 5 adding to the time in 7 the travel time to its destination. Finally in 10 it is joined once again by the drone which has since travelled to 4 and delivered its goods there. Once again it is the drone which has to wait for the truck to arrive, for approximately $2626\ s - 2579\ s = 47\ s$.

$$
\begin{aligned}
&t_4^D = t_1^D + SL + \tau_{1,4}^D = 1219\ s + 40\ s + 899\ s = 2158\ s \\
&t_7^T = t_1^T + SL + \tau_{1,7}^T = 1219\ s + 40\ s + 1010\ s = 2269\ s \\
&t_5^T = t_7^T + \tau_{7,5}^T = 2269\ s + 116\ s = 2385\ s \\
&t_{10}^T = t_{10}^D = \max\left\{t_4^D + \tau_{4,10}^D + SR, t_5^T + \tau_{5,10}^T + SR\right\} = \\
&\quad = \max\{2158\ s + 391\ s + 30\ s, 2385\ s + 211\ s + 30\ s\} = 2626\ s
\end{aligned}
$$

After the second sortie, this solution has only truck deliveries. This means that each truck time that follows is calculated based on the time at the previous node, and adding to that the time to travel from the previous node to the current one as per constraint (3.2.15).

$$
\begin{aligned}
&t_9^T = t_{10}^T + \tau_{10,9}^T = 2626\ s + 1824\ s = 4450\ s \\
&t_3^T = t_9^T + \tau_{9,3}^T = 4450\ s + 1127\ s = 5577\ s \\
&t_8^T = t_3^T + \tau_{3,8}^T = 5577\ s + 188\ s = 5765\ s \\
&t_2^T = t_8^T + \tau_{8,2}^T = 5765\ s + 1356\ s = 7121\ s \\
&t_{11}^T = t_2^T + \tau_{2,11}^T = 7121\ s + 1257\ s = 8378\ s
\end{aligned}
$$

Chapter 4

Simulated Annealing

Some of the possible methods to heuristically solve the FSTSP are introduced in Possible solution approaches (4.1): the Simulated Annealing (4.1.1), Ant Colony Optimization (4.1.2) and the Naïve approach (4.1.3).

Out of all three, it is explained that the former is the preferred one, and in Implementation of the Simulated Annealing (4.2) it is thoroughly explained how the SA was implemented. In Data structures (4.2.1) the reader will find a description of the data structures adopted in the implementation, and why they were preferred over other structures. In Moves in the Simulated Annealing (4.2.2) the ideated moves are presented and the move `relocateCustomer()` is analyzed down to the last detail, while the other three are presented in a broader way. Finally, in Solution evaluation (4.2.3) the solution evaluation process is outlined in both the possible ways it can be executed.

4.1 Possible solution approaches

Any instance of the TSP [Karp, 1972] can be transformed to an instance of the FSTSP with a polynomial time transformation (by disallowing the use of drone sorties) so, since the TSP is NP-hard, it follows that the flying sidekick TSP is NP-hard as well. Test runs with the smaller self-generated instances described in 5.1 indicate that the problem is difficult to solve using the mathematical model. For example `Instance_010.2` has shown to require almost a full day to be solved to optimality, and the average for 10-customers instances is several hours, making infeasible the generation of optimal solutions for practical use-cases.

As discussed in [Rosenkrantz et al., 2009, Nilsson, 2003], several heuristic approaches exist for the TSP. Nilsson categorizes them in constructive and improvement algorithms. As a part of the improvement algorithms and later chapters, he also discusses metaheuristics, such as Tabu Search, Simulated Annealing, Genetic Algorithms, and Ant Colony Optimization.

Exploring the possibilities to find a good heuristic solution, there are three approaches that seem interesting and also promising enough to be candidates for further investigations: Simulated Annealing (4.1.1), Ant Colony Optimization (4.1.2), and naïve approach (4.1.3). The second and third methods have not been implemented and they are addressed in this section because they were still analyzed as options to a certain extent, making them effectively be part of this study.

4.1.1 Simulated Annealing

Simulated Annealing (SA) [Kirkpatrick et al., 1983, Černỳ, 1985] is an iterative metaheuristic based on the Monte Carlo algorithm that was conceived in the 80s and has since then seen widespread use to find approximate solutions to the TSP and many other discrete optimization problems.

Simulated Annealing draws similarities between statistical mechanics, that is the behaviour of many degrees of freedom systems in thermal equilibrium at a determined temperature, with combinatorial optimization, which consists in finding the mininmum of a given function depending on many parameters. Each permutation of the nodes in the TSP tour is a configuration of the statistical system, where the length of the trip represents its energy in that particular configuration. But large systems at a given temperature are

known to approach spontaneously an equilibrium state with minimal energy in relation to the current temperature. The system starts at a very high temperature. By simulating its change of configuration and gradually decreasing the temperature, one can find progressively smaller mean energy values, thus leading the system to progressively lower energy states.

In OR terms, this translates to being able to traverse the solution space by generating neighboring solutions of the current one. These are generated by means of different types of so-called moves performed on the current path. Acceptance of new solutions works as follows: while a better tour is always accepted, a worse one is accepted based probabilistically on the difference in quality and on a temperature parameter. The temperature is decreased according to a cooling schedule as the algorithm progresses, in order to alter the nature of the search, allowing at first for more bad solutions to be accepted and gradually moving onto accepting only the better ones. This way the algorithm has also the ability to escape local optima and possibly converge on the true optimum.

The algorithm can be described as follows:

1. Generate the initial solution s_0;
2. Let $s = s^* = s_0$;
3. Perform a randomly selected move on s to obtain s_{new};
4. If the probability of acceptance for s_{new} (usually $e^{\frac{f(s_{new})-f(s)}{T}}$) is bigger than a randomly generated number, let $s = s_{new}$;
5. If the objective of solution s_{new} is smaller than the objective of the best solution s^*, let $s^* = s_{new}$;
6. Update the temperature according to the cooling schedule α: $T_{new} = \alpha * T$;
7. If the stopping criterion is not yet met, go back to step 3.

The criteria to stop the algorithm can be of multiple nature, some examples include reaching a fixed number of iterations, spending a certain amount of time, not having improved the best solution for a certain number of iterations, and more.

As with other metaheuristics, the outcome of this method is heavily dependent on the parameter setting. Typical parameters include the start and end temperatures and the number of iterations to perform, based on which one can compute the necessary cooling schedule, and also the probability with which a given move can be chosen among the others. Parameters must thus be tuned for a specific problem. Even though an absolute best set of parameters doesn't exist, there are strategies to converge to a good enough choice.

It is also worth noting that, since it is a stochastic algorithm, each execution of the SA can produce a different solution. That is why it is usually necessary to run the algorithm many times to get a good idea of the mean and standard deviation of the best solution found.

Simulated Annealing has the advantages of being able to deal with highly nonlinear models with many constraints, and it is a robust and general technique. With respect to other local search methods, it is more flexible (see [Busetti, 2003]), and it has been proven to be able to find the global optimum with a careful control of the cooling rate (see [Johnson and Jacobson, 2002]: the optimum will be found with as high a set probability as one desires, requiring longer runtime as the main drawback). The biggest drawbacks to using this method regard the necessary tradeoff between solution quality and computation time, the hard tailoring work required to account for different classes of constraints,

fine tuning of parameters and choices necessary to turn the metaheuristic into an actual algorithm.

Seeing that SA has, to the author's knowledge, never been tried before on the FSTSP or drone-related problems, and seeing that it doesn't seem to show a high entrance barrier, it shall have the top priority in trying to find optimal solutions to the FSTSP.

4.1.2 Ant Colony Optimization

Ant Colony Optimization is an iterative metaheuristic that takes inspiration from the behaviour of ants foraging for a colony. This kind of ants deposit a special type of pheromone on the ground to mark favourable paths that ought to be followed by other ants. The metaheuristic maintains that idea to solve combinatorial optimization problems [Dorigo et al., 2006].

For the TSP, ACO translates to a number of solutions to the problem being generated by artificial ants (or agents) moving on the instance's graph. Each edge of the graph has a linked value of pheromone that can be read and modified by the ants. In every iteration a different number of ants build one solution each by walking from node to node without visiting previously visited nodes. The node selection to build the solution is governed by a stochastic mechanism guided by the pheromone: the more pheromone present in one of the edges going out of that vertex to a not-yet-visited one, the more likely that ant will traverse that edge. At the end of every iteration, the pheromone values on the edges are updated in order to bias future ants to construct new solutions similar to the best ones previously built.

For the FSTSP the ACO approach would most likely be to have two kinds of pheromone, one for truck and one for drone paths, and then follow the basic framework of the metaheuristic:

1. Initialize the whole structure setting parameters like number of ants per iterations, initial position of the ants on the nodes and initial pheromone values;

2. Generate the new ants' solutions;

3. Improve found solutions (optional);

4. Increase pheromone values associated with promising solutions and decrease values for the bad ones;

5. If termination condition is not reached, go back to 2.

ACO can be run continuously and can react in real time and relatively quickly to changes in the graph, which is ideal for real world vehicle routing. This seems also promising for drone use, since a drone can be quickly displaced from any point in the route, making last minute deliveries of consumables faster and easier (e.g. frequently requested SKUs like Amazon's Add-on Items). Moreover, it has proven to produce near-optimal solutions for TSP instances and has the advantage over Simulated Annealing (and other metaheuristics) of seemingly being better at escaping local optima.

This is the second most interesting approach in need of analysis: it is never been tried before for the FSTSP or drone-related problems, has very promising characteristics, but it is slightly more difficult to approach as a metaheuristic than SA and more experience with similar methods is needed before attempting this algorithm's implementation.

4.1.3 Naïve approach

The naïve approach consists of combining a constructive algorithm coupled with an improvement algorithm, to generate a feasible truck tour, with a way to insert drone deliveries on this tour in the best way possible, according to different metrics.

This approach was eventually postponed to after other approaches as it has already been partially treated by both [Murray and Chu, 2015, Shang et al., 2014], and as such would need a more in-depth analysis to improve their results. Still, a few of the possibilities were explored in the preliminary studies.

Constructive algorithms The basic truck tour can be any tour as long as it is feasible.

The two main ideas to generate the basic tours were to use either a heuristic method, or to generate the optimal TSP tour. Since no method has yet been proven to lead directly to the truck path underlying the optimal solution in an FSTSP scenario, both seem to be equally good starting points for subsequent sortie insertion.

The heuristic method of choice is the Nearest Neighbor algorithm, which constructs a path as follows:

1. Select a random node in the graph;
2. Find the nearest unvisited node to the last added node and add to the path the edge connecting these two nodes;
3. Are there any unvisited nodes left? If yes, go back to step 2;
4. Add to the path the edge connecting the last node to the first node.

This is possibly the simplest and most straightforward TSP heuristic, it can be coded to have time complexity $O\left(n^2\right)$ and generates relatively poor solutions, often 25% from optimality [Nilsson, 2003].

It is also important to note that the Nearest Neighbour solution can already be improved by running the algorithm starting from every node and picking as the initial constructed tour the one with minimal cost among these. It is however still of dubious relevance the necessity to get a better initial tour.

To generate the optimal TSP tour a solver like Concorde [Applegate et al., 2006] can be used. Concorde is a state of the art solver for the TSP, and although the time needed for it to solve a given instance can vary considerably with the number of nodes [Applegate et al., 2007], for a use-case like the ones that are analyzed in this thesis, it is perfectly suitable.

Improvement algorithms The simplest way to improve a non-optimal truck solution is the 2-opt algorithm.

It is the simplest form of the k-opt local search algorithm, that works by removing two edges from the tour and reconnecting the two paths created in the only possible way so as to keep a valid tour (see Figure 4.1).

The main idea, as such, is to take a route which crosses over itself and reorder it so that it doesn't. The method to construct a new path is:

1. add to the new path the portion of old path going from the beginning to node A;
2. add to the new path in reverse order the portion of old path going from node C to node B;
3. add to the new path the portion of old path from node D to the end.

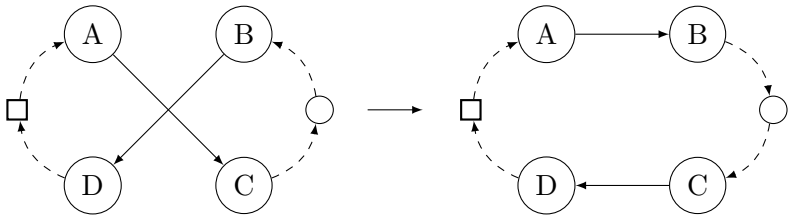


Fig. 4.1: How the 2-opt algorithm works

Insertion of drone deliveries The insertion of drone deliveries can be done in different ways, and here two of them are shown.

The first method consists of inserting step-by-step the sortie that saves the most until no more sorties have a positive saving value. It can be summarized like so:

1. Compute all the savings for every sortie still in F;
2. Continue if at least one sortie has positive saving;
3. Order the list in decreasing savings order and start with the first in the list;
4. If that sortie is feasible, add it to the solution and remove it from F, otherwise go to the next biggest saving sortie and repeat this step;
5. Go back to step 1.

This method provides a quick and easy way to find a feasible solution, and is loosely based on Clarke & Wright's Savings Algorithm. The solution found can also serve as a base solution for further improvements via other methods discussed later on (see 4.1.1 and 4.1.2).

However, this approach is very similar to what [Murray and Chu, 2015] do in their FSTSP heuristic, so other methods are prioritized, leaving this one an option for further considerations after other never-before-tried approaches will have been tested. Among the naïve approaches, it should be the last to receive further investigations, as the other method seems to be more promising in terms of results and time complexity.

The second method consists of finding the optimal savings insertion by conceiving a mathematical model. The objective function should be the one in Eq. (4.1.1), where $s_{i,j,k}$ is the saving associated with flight $\langle i, j, k\rangle \in F$.

$$\max \sum_{\langle i,j,k\rangle \in F} s_{i,j,k} * y_{i,j,k} \tag{4.1.1}$$

There should be a set of constraints to ensure solution feasibility: prohibiting PM1 and PM2, and enforcing an analogous version of the endurance constraint from the main mathematical model (Constraint (3.2.18)). There should also be a set of constraints to calculate the savings for F.

This method would be a considerable improvement over the savings heuristic presented in the previous paragraph. Being still tied to the same idea as [Murray and Chu, 2015]'s FSTSP heuristic, this approach should be left as an option for further investigations only after other approaches will have been tested. Among the naïve approaches, it should be the preferred to be considered for in-depth analyses, since it should give better results than the simplistic first method.

4.2 Implementation of the Simulated Annealing

As discussed in Possible solution approaches (4.1) the first approach that should be examined is the Simulated Annealing metaheuristic. This was done with an implementation of the algorithm in Java SE 8.

Algorithm 4.1 Main method for the implemented Simulated Annealing

```
1:  INITIALIZE startTemperature, endTemperature, maxNumberIterations
2:  coolingRate = e^(log(endTemperature/startTemperature)/maxNumberIterations)
3:  INITIALIZE dataStructures
4:  COMPUTE initialSolution
5:  INITIALIZE currentSolution, bestSolution, currentObjective, bestObjective
6:  currentTemperature = startTemperature * initialObjective
7:  globalIteration = 0
8:  WHILE currentTemperature > endTemperature * initialObjective DO
9:    currentTemperature = currentTemperature * coolingRate
10:   EXECUTE randomMove(movesWeight, currentSolution)
11:   tentativeObjective = currentObjective + moveCost
12:   accept = e^(currentObjective − tentativeObjective)/currentTemperature)
13:   IF random([0,1]) ≤ accept THEN
14:     currentSolution = tentativeSolution
15:     IF tentativeObjective < bestObjective THEN
16:       bestSolution = tentativeSolution
17:     END IF
18:   END IF
19:   globalIteration = globalIteration +1
20: END WHILE
21: PRINT  bestSolution, bestObjective
```

The Simulated Annealing's main method (see Alg. 4.1), closely resembles the loose description provided in 4.1.1.

Some considerations can be made on the pseudocode, starting with the decision on the stopping condition for the while loop. In this implementation of the SA that condition is tied to the maximum number of iterations N set by the user, and chosen as best as possible during the parameter tuning phase together with the start temperature T_s and the end temperature T_f. The cooling schedule's coefficient $0 < \alpha < 1$ is then calculated to reflect this decision by simple means of algebra manipulations which eventually lead to Eq. (4.2.1) (see Line 2). The temperature is updated constantly throughout iterations, as can be seen in Line 9.

$$\alpha = \mathrm{e}^{\frac{\log \frac{T_f}{T_s}}{N}} \tag{4.2.1}$$

Step 1 of the algorithm in 4.1.1 is done in Line 4 and is implemented as a Nearest Neighbour algorithm, with starting point the depot, and which provides the starting tour for the SA. The initial objective is also given after a conversion from distance to time, identical to the one used in calculating the travel time matrices (see Sec. 3.5). The valid flights are also computed in the initialization phase (see Line 3) and stored for fast validation on sorties.

It is worth noting that the actual temperatures between which the Simulated Annealing will run are also proportional to the initial objective, as seen in Lines 6 and 8. This doesn't

influence the running time of the algorithm, which is still going to be the maximum number of iterations mentioned a couple of paragraphs earlier, but it will influence the acceptance of solutions. Moreover, this allows the temperature to adjust to the size of the objective: instances with huge starting objectives have larger start temperatures, instances with smaller ones have smaller temperature values.

As a matter of fact, the acceptance probability is a value in the real interval $(0, 1)$ and is calculated in Line 12 with Eq. (4.2.2). It is then compared to a random number between 0 and 1 in Line 13.

$$\text{accept} = \text{e}^{\frac{Obj_{curr} - Obj_{tent}}{T_{curr}}} \tag{4.2.2}$$

This formula returns a number greater or equal to one for any tentative objective smaller or equal to the current objective. This ensures that better objective values are always accepted. In case of tentative objectives strictly greater than the current one, the returned number is instead influenced by the current temperature which is a function of the initial objective, as previously discussed. e's exponent is inversely proportional to the temperature, linking the acceptance probability to the size of the instance, effectively allowing for a bigger — albeit marginally — chance of accepting worse solutions for bigger instances at higher temperatures.

Following, the reader can find an in-depth analysis of the developed moves (see 4.2.2) and the solution evaluation process (see 4.2.3). But first, a comment on the data structures adopted in this implementation.

4.2.1 Data structures

To have a fast working algorithm, able to run millions of iterations in a reasonable amount of time, the program must be coded with the right data structures used in the right processes. Time complexity plays an important role in being able to have an efficient process, and starting out with the right abstract representation form can go a long way in saving time in the long run.

The distance matrix is read from an external `Instance_nnn.x.txt` file (where `nnn` is the number of customers and `x` is the progressive number of the instance) and stored as an `int[][]` integer array. It is only used twice, for the calculation of both travel time matrices and the generation of the Nearest Neighbour solution: in both cases it is needed to access the data, and every access done in $O(1)$ thanks to this structure.

After being initialized, the time matrices τ^T for the truck and τ^D for the drone are only used for access, so a `double[][]` floating point array is a perfect fit for both, having access in $O(1)$.

The set of serviceable customers C^D is generated starting from the excludedCustomers `Set<Integer>` and transformed into the serviceableCustomers `int[]` that is included in . This process was summed up in the pseudocode as Line 3 because it is more meaningful to keep track of the customers excluded from drone delivery rather than the ones allowed. In addition to that, the final result is the same since, even though this initialization is done in $O(n)$, it is only done once per execution, so the overall impact is minimal. The integer array choice for serviceableCustomers is due to having the need for both access, which is the prevalent use and is done in $O(1)$, and for searching over it: being generated as already sorted, the method applied for searching is binarySearch, which is $O(\log n)$.

The generic solution is stored in a triplet of `ArrayList<Integer>`, and it is shown in Fig. 4.2. The choice to use an integer array for the truck solution (T) comes from the fact that access is called for many more times than insertion or deletion, since many tentative moves end up being rejected. Thus, having $O(1)$ access is a priority. That said, the same choice done to represent the drone solution wasn't an immediate and evident choice.

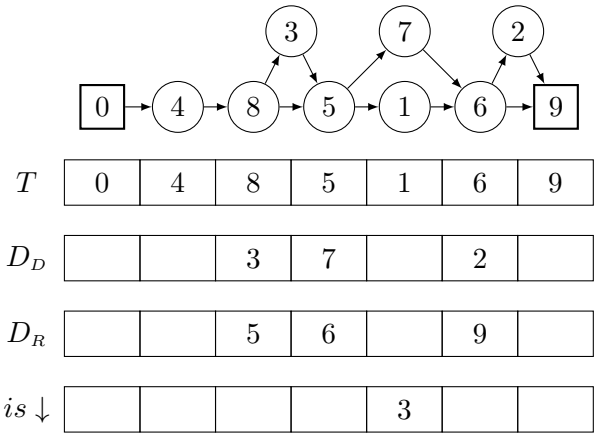


Fig. 4.2: Solution structure of `Instance_008.1` for the Simulated Annealing

Any sortie can be generically represented by a triplet of values, so either a tuple or a triplet of arrays would serve for this purpose. However, considering the latter, the "launch" array would constitute redundant information, as it is already stored in the truck solution. Thus, the drone sorties can be represented by a pair of arrays, containing respectively the drone delivery solution (D_D) and the drone rendezvous solution (D_R). In such a way, access is $O(1)$, and adding and removing are done in $O(n)$. In Fig. 4.2 blank spaces are substituted to -1, which is the actual value stored in the implementation, for easier readability.

This choice is to be preferred to the tuple because the second would require a supplemental `indexOf()` call every time a sortie needs to be added, removed, or anyhow manipulated during a move, and that call would be $O(n)$ complexity-wise.

Together with the solution, the `ArrayList<Integer> isUnder` ($is\downarrow$) is used to keep track of truck visits done between the launch and the rendezvous of the UAV. It is used mainly for checks, so the array implementation is, again, a perfect match, and it serves as a facilitator in many of the computations. In it, are stored either the value -1 or the index of the launch of the sortie to which this node is "under". For example, in Fig. 4.2 customer 1 is under sortie $\langle 5, 7, 6\rangle$ which starts at index 3 in the truck solution (T).

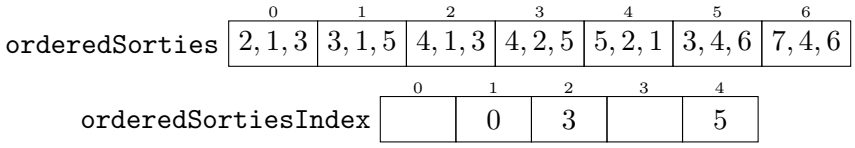


Fig. 4.3: Visual example for the inner workings to generate random sorties

To facilitate and improve the SA's performances, three structures are implemented and initialized before the start of the SA (see Line 3 in Alg. 4.1):

- `validFlights`: a `boolean[][][]` used only to check if a particular combination of launch, delivery and rendezvous is a validly defined sortie;
- `orderedSorties`: an `ArrayList<Sortie>` used to create `orderedSortiesIndex`;
- `orderedSortiesIndex`: an `ArrayList<Integer>` used for random valid sortie generation.

The `Sortie` object is a Java Object defined with a launch, a delivery and a rendezvous that can be sorted by increasing delivery customer value. In Fig. 4.3 the reader can see

a small example of how `orderedSorties` and `orderedSortiesIndex` would be after the whole initialization is done. `orderedSorties` is an array that stores all the valid sorties defined also in `validFlights`, and has them ordered by delivery. `orderedSortiesIndex` is an `ArrayList<Integer>` computed right after `orderedSorties` which further facilitates sortie generation by storing at position x the index at which the first sortie with delivery customer x appears in `orderedSorties`. This eliminates the need for continuous calls of `indexOf()` on `orderedSorties` which gets very taxing on performances extremely quickly.

4.2.2 Moves in the Simulated Annealing

Moves are needed to make a Simulated Annealing work. In Line 10, Alg. 4.1 the call is made to perform a random move on the current solution.

Before a move can be performed, one has to be randomly chosen, and this is done by using the weights initialized in Line 3 of Alg. 4.1: the choice is implemented as a Roulette-Wheel Selection [Lipowski and Lipowska, 2012], but what this ultimately means is that if a weight was given for each move, for example 50 20 20 10, then the first move would be chosen five times more than the fourth one. The weight effectively represents the probability of a move being chosen.

Once a move has been chosen, it has to be performed. Four moves are here presented: `relocateCustomer()`, `relocateCluster()`, `swapCustomers()` and `crossPath()`. Even though four were the moves originally conceived, only one made it in the final implementation. After a thorough explanation of `relocateCustomer()`, the other three are briefly discussed in their main idea.

Move `relocateCustomer()` This moves a random customer in any random position in the tentative solution. The move is fairly complex, and permits a good exploration of the neighborhood overall. The main part of the algorithm for this move can be seen in Alg. 4.2 but before that, a moment is taken to analyze what this move means for a generic solution.

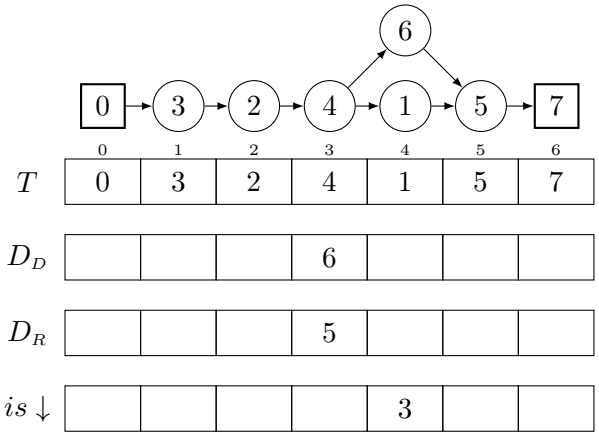


Fig. 4.4: Generic solution for the explanation of the move `relocateCustomer()`

In a generic solution, a generic customer can be truck-only (3, 2, 1 in Fig. 4.4), drone-only (6 in Fig. 4.4) or mixed (4, 5 in Fig. 4.4). Mixed customers are the truck customers that have a launch or a rendezvous of the drone, drone-only customers are the customers served with a drone delivery, and finally truck-only customers are the remaining truck customers.

For truck-only customers, the move is fairly straightfoward: if the randomly chosen delivery method is the truck, that customer positioned in the truck path, and wether it

goes under a sortie or it doesn't, changes only the feasibility of the move via the endurance constraint. This would mean moving for example customer 1 in position 6 (between 5 and 7) or customer 2 in position 4 (between 4 and 1). In the second example, the truck path between launch and sortie increases, so the endurance constraint might be violated since the drone could possibly have to wait hovering for too long for the arrival of the truck in 5. If instead the picked delivery method is the drone, then that customer is setup as a drone delivery whose launch and rendezvous are randomly initialized. This would mean for example moving customer 3 in sortie $\langle 0, 3, 2\rangle$ or $\langle 5, 3, 7\rangle$ or customer 1 in sortie $\langle 2, 1, 4\rangle$ or $\langle 0, 1, 4\rangle$, provided they are feasibly defined sorties and don't violate the endurance constraint. The example sorties were picked so as not to incur in Prohibited Moves (see Sec. 3.3): if customer 2 was to be moved in sortie $\langle 3, 2, 1\rangle$ (PM1) or in sortie $\langle 4, 2, 7\rangle$ (PM2) it would make the solution infeasible and the move would be retried.

Drone-only customers are similarly straightforward: it would mean moving customer 6 to a truck delivery (any position between 1 and 6 is valid) or a (different) drone delivery, for example $\langle 0, 6, 3\rangle$ or $\langle 1, 6, 7\rangle$.

Algorithm 4.2 Main `relocateCustomer()` algorithm

```
1:  REPEAT
2:    deliveryMethod = random(truck,drone)
3:    REPEAT
4:      tentativeSolution = copyOf(currentSolution)
5:      fixedVehicleIt = fixedVehicleIt + 1
6:      INITIALIZE randomCustomer
7:      IF deliveryMethod = truck THEN
8:        REPEAT
9:          INITIALIZE randomPosition
10:       UNTIL randomPosition is different from original randomCustomer position
11:       IF randomCustomer is a mixed customer THEN
12:         performMixedToTruck(tentativeSolution)
13:       ELSE IF randomCustomer is a drone-only customer THEN
14:         performDroneToTruck(tentativeSolution)
15:       ELSE IF randomCustomer is a truck-only customer THEN
16:         performTruckToTruck(tentativeSolution)
17:       END IF
18:     ELSE
19:       IF randomCustomer isn't an excluded customer THEN
20:         randomSortie = generateRandomSortie(randomCustomer)
21:         IF randomCustomer is a mixed customer THEN
22:           performMixedToDrone(tentativeSolution)
23:         ELSE IF randomCustomer is a drone-only customer THEN
24:           performDroneToDrone(tentativeSolution)
25:         ELSE IF randomCustomer is a truck-only customer THEN
26:           performTruckToDrone(tentativeSolution)
27:         END IF
28:       END IF
29:     END IF
30:   UNTIL move isn't feasible AND fixedVehicleIt < MAX_FIXED_VEHICLE_IT
31: UNTIL move isn't feasible
```

The same can't hold true for mixed customers, as the situations that arise are vastly different. If the customer were to be moved to a drone delivery, the move would require a number of strict conditions to hold true to make the move feasible. If instead the customer

is set for a truck delivery, then the sortie is also affected by the move, in that it can expand the number of customers under it, or it can be reversed. This would mean for example moving 4 in any position, so that between position 1 and 5 the sortie would stay in the same $\langle 4, 6, 5 \rangle$ configuration, whereas if 4 were to be moved in position 6 the sortie would be reversed into $\langle 5, 6, 4 \rangle$. To see all the possible cases and how the move is implemented for mixed customers, refer to Alg. 4.5.

Now back to the analysis of how `relocateCustomer()` is performed, as per Alg. 4.2. The move is coded so that it only returns feasible solutions (see Line 31) but to allow a little bit of leeway there is also a counter to check that the code doesn't get stuck for too long (see Line 30) while trying to keep the same deilvery method (see Line 2). Once inside this second loop the tentative solution gets initialized as a copy of the current solution (see Line 4) and a random customer gets generated between 1 and c (see Line 6).

Now, if the delivery is to be performed via truck, then a random position where to put the new truck customer is generated as an integer value between 1 and the length of the truck solution -1 (one over and one under to avoid inserting customers before the initial depot or after the final depot). This is repeated until the random position is different from the previous position of the random customer, to avoid null moves (see Lines 8–10).

The algorithm then proceeds to split the move execution depending on the type of random customer drawn: if it is a mixed customer it proceeds in Line 12 which is treated in Alg. 4.5, if it is a drone-only customer it proceeds in Line 14 which is treated in Alg. 4.4, and finally if it is a truck-only customer it goes to Line 16, seen in Alg. 4.3.

Algorithm 4.3 How a truck-only to truck move is performed

```
1: removeTruckCustomerFromPosition(randomCustomer)
2: addTruckCustomerToPosition(randomCustomer, randomPosition)
3: updateTentativeIsUnderSortie()
4: IF endurance constraint isn't violated THEN
5:     isMoveFeasible = true
6: END IF
```

Alg. 4.3 is called from Line 16 and moves `randomCustomer` to `randomPosition` by removing it from its current position and adding it to the new one. The only check needed is for the endurance constraint: if the truck-only customer goes under a sortie, and the drone has to wait for more than what its battery is worth, the solution is infeasible.

Algorithm 4.4 How a drone-only to truck move is performed

```
1: removeSortieWithDelivery(randomCustomer)
2: addTruckCustomerToPosition(randomCustomer, randomPosition)
3: updateTentativeIsUnderSortie()
4: IF endurance constraint isn't violated THEN
5:     isMoveFeasible = true
6: END IF
```

Alg. 4.4 is called from Line 14 and moves `randomCustomer` to `randomPosition` by removing its current sortie and adding the customer to the new position in the solution. The only check needed is again for the endurance constraint, for the same reason as for the truck-only to truck move.

Alg. 4.5 is called from Line 12 and is not as simple as the two preceding bits. The algorithm has to split up again right at the beginning due to the three possible configurations a mixed customer can be in: it can be the launch of a sortie, the rendezvous of another sortie or both. All three configurations are illustrated in Fig. 4.5: node i has

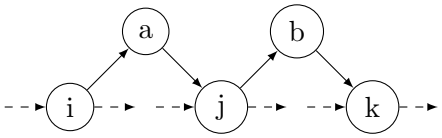


Fig. 4.5: The three configurations of a mixed customer (i, j and k)

a launchSortie (i, a, j), node k has a rendezSortie (j, b, k) and node j has both a launch (j, b, k) and a rendezvous sortie (i, a, j).

Algorithm 4.5 How to perform a mixed customer to truck move

```
1: IF randomCustomer has a launchSortie THEN
2:    IF randomCustomer has a rendezSortie THEN
3:       performMixedLR(tentativeSolution)
4:    ELSE
5:       performMixedL(tentativeSolution)
6:    END IF
7: ELSE
8:    performMixedR(tentativeSolution)
9: END IF
```

To clarify the mixed customer move, it is first better to take a look at Fig. 4.6, which presents the three main situations that can happen with a customer that has a launch sortie. The same can simmetrically happen for a customer with a rendezvous sortie. In (a) the customer has also a rendezvous sortie, so the code to execute is the one Line 3 of Alg. 4.5, later described in Alg. 4.6. Both in (b) and (c) the situation is described in Alg. 4.7 and is called in Line 5 of Alg. 4.5, but in the case of (b) the sortie isn't reversed, whereas in (c) it is.

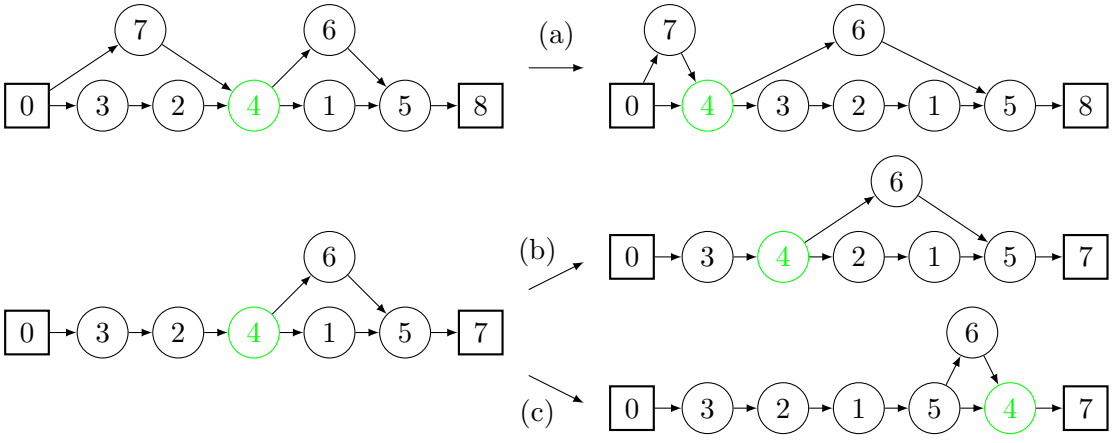


Fig. 4.6: Possibilities for mixed customers with a launchSortie in `relocateCustomer()`

Back to the pseudocode, if the mixed customer has both launch and rendezvous sortie (Alg. 4.6), the only way the move can be feasible is if `randomPosition` is between the launch of the rendezvous sortie and the rendezvous of the launch sortie (Line 1). In the context of Fig. 4.5, this means $i < rP < k$, whereas in Fig. 4.6 (a) this means $0 < rP < 5$, with an appropriate translation of customers to indices. Once this condition is verified, then the move consists of removing `randomCustomer` from its position, adding it back to the new `randomPosition` and finally adding back the launch sortie that would have otherwise been lost. No Prohibited Moves are possible with such a move so, before confirming the

Algorithm 4.6 `performMixedLR()` pseudocode

```
1: IF rendezSortie.launch < randomPosition < launchSortie.rendez THEN
2:   removeTruckCustomerFromPosition(randomCustomer)
3:   addTruckCustomerToPosition(randomCustomer, randomPosition)
4:   addSortieBackToCustomer(launchSortie)
5:   updateTentativeIsUnderSortie()
6:   IF endurance constraint isn't violated THEN
7:     isMoveFeasible = true
8:   END IF
9: END IF
```

feasibility of the move, only the endurance constraint violation has to be verified.

The only big difference between the moves seen until now and the ones for mixed customers with just a launch or rendezvous sortie is that in this case the sortie can be reversed by having `randomPosition` relocated respectively after the rendezvous or before the launch. This is checked for, respectively, in Line 3 Alg. 4.7 and in Line 3 Alg. 4.8. Their feasibility is checked with the `validFlight` boolean matrix. After that the move is performed and checked for feasibility with regards to both the endurance constraint and the Prohibited Moves.

Algorithm 4.7 `performMixedL()` pseudocode

```
1: removeTruckCustomerFromPosition(randomCustomer)
2: addTruckCustomerToPosition(randomCustomer, randomPosition)
3: IF launchSortie isn't reversed THEN
4:   addSortieBackToCustomer(launchSortie)
5:   updateTentativeIsUnderSortie()
6:   IF endurance constraint isn't violated AND isn't a Prohibited Move THEN
7:     isMoveFeasible = true
8:   END IF
9: ELSE
10:  IF revLaunchSortie is feasible AND isn't double launch THEN
11:    addSortieBackToCustomer(revLaunchSortie)
12:    updateTentativeIsUnderSortie()
13:    IF endurance constraint isn't violated AND isn't a Prohibited Move THEN
14:      isMoveFeasible = true
15:    END IF
16:  END IF
17: END IF
```

If, instead, the drone has to complete the delivery, then the sortie generation process is called for (see Line 20), which can be seen in Alg. 4.9.

To generate a valid random sortie, as stated in Sec. 4.2.1 some structures have been put in place. Computing `indexOfFirstSortie` (see Line 1) is just a matter of accessing the array `orderedSortiesIndex` in the random customer's position. Same goes for `indexOfLastSortie` (see Line 2), which is found via an access to the same array in the next customer's position.

If that particular random customer doesn't have a sortie available, either because it is an excluded customer or it simply is out of reach from the drone's perspective, the method returns a null value, which issues a change of random customer. If, instead, at least one sortie exists, then a while loop ensures that a valid one is picked among the available ones, again with a failsafe number of iterations to go through before changing customer (see

Algorithm 4.8 `performMixedR()` pseudocode

```
1: removeTruckCustomerFromPosition(randomCustomer)
2: addTruckCustomerToPosition(randomCustomer, randomPosition)
3: IF rendezSortie isn't reversed THEN
4:    updateTentativeIsUnderSortie()
5:    IF endurance constraint isn't violated AND isn't a Prohibited Move THEN
6:       isMoveFeasible = true
7:    END IF
8: ELSE
9:    IF revRendezSortie is feasible AND isn't double rendezvous THEN
10:      removeSortieWithDelivery(rendezSortie.deliv)
11:      addSortieBackToCustomer(revRendezvousSortie)
12:      updateTentativeIsUnderSortie()
13:      IF endurance constraint isn't violated AND isn't a Prohibited Move THEN
14:         isMoveFeasible = true
15:      END IF
16:   END IF
17: END IF
```

Algorithm 4.9 How to generate a valid random sortie

```
1: COMPUTE indexOfFirstSortie
2: COMPUTE indexOfLastSortie
3: IF customer has at least one sortie available THEN
4:    WHILE haven't found a valid sortie AND fixedCustIt < MAX_FIXED_CUST_IT
      DO
5:       fixedCustIt = fixedCustIt +1
6:       COMPUTE randomSortie
7:       IF passTestrandomSortie THEN
8:          foundValidSortie = true
9:          RETURN randomSortie
10:      END IF
11:   END WHILE
12: END IF
13: RETURN null
```

Line 4).

Finally the random sortie is generated (see Line 6) as a random integer between `indexOfFirstSortie` and `indexOfLastSortie`. This is eventually used as an index to access `orderedSorties`. In this way, the whole process is done in constant time, like the generation of the random position for the truck customer.

The generated sortie is returned for further computations if and only if all the common possible compatibility problems with the current solution, addressed in `passTest()` (see Line 7), aren't verified:

- If `randomCustomer` is a drone customer, `randomSortie` might be a null move;
- `randomSortie` might create a double launch or a double rendezvous from or to a singular node;
- It might have what currently is a drone customer as launch or rendezvous;
- `randomSortie` might have a rendezvous before the launch in the current solution for the truck.

If the drawn `deliveryMethod` is instead the drone, the situation is extremely similar to the one for the truck in case of truck-only or drone-only customers: the only difference is that instead of adding `randomCustomer` back to the solution in `randomPosition`, it is `randomSortie` that gets added to the solution.

The unfortunate case of having a mixed customer, instead, makes the move impractical to the point that feasibility is possible only for extremely specific conditions. Since this is the case, a mixed customer being drawn for the drone would mean many wasted tries to actually achieve a feasible tentative solution. The method exists but has been disallowed to be performed due to performance degradation and limited neighborhood improvement.

Move `relocateCluster()` This non implemented move was set to relocate a random cluster of nodes of the same type (i.e. uninterrupted pieces of truck path or a full drone sortie) in any random position in the tentative solution. It would complete only as long as this relocation resulted in a feasible move: it mustn't create a PM1 or PM2, and the endurance constraint must not be violated.

It is visually represented in Fig. 4.7 where in (a) the truck cluster $(3, 2)$ gets moved in position 5, whereas in (b) the drone cluster $(4, 6, 5)$ gets moved with the launch in position 1 and the rendezvous in position 4.

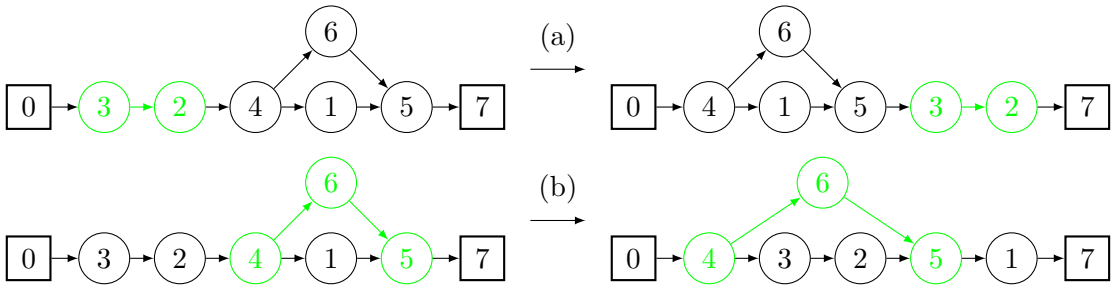


Fig. 4.7: Visual description of how `relocateCluster()` works

Move `swapCustomers()` This non implemented move is set to swap a random customer with another random one in the tentative solution. It would complete only as long as this swap resulted in a feasible move: the endurance constraint must not be violated.

It is visually represented in Fig. 4.8 where in (a) the truck-only customer 3 gets swapped with the mixed customer 5, whereas in (b) the drone-only customer 6 gets swapped with 1, which is under that same sortie.

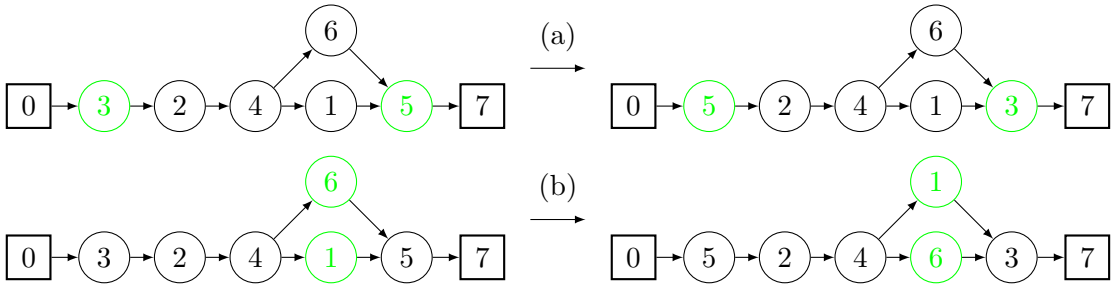


Fig. 4.8: Visual description of how `swapCustomers()` works

Move `crossPath()` This final non implemented move is a 2-opt move performed on the solution: two edges are removed from the tentative solution and the two paths created are reconnected in the only possible way so as to keep a valid tour.

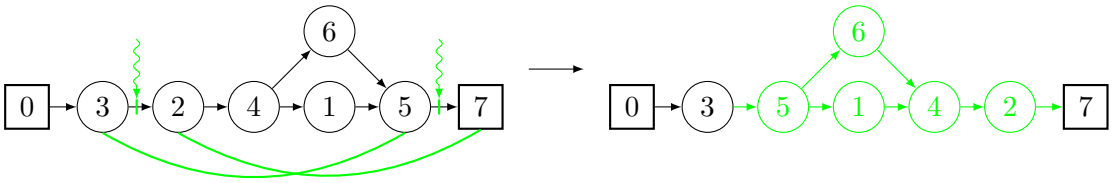


Fig. 4.9: Visual description of how `crossPath()` works

To be feasible, the move requires that if there are sorties in the reversed path, the reversed sorties must be feasible endurance-wise. Moreover, the removed edges must be between truck-only or mixed nodes, and in both cases none of the edges must be with a customer who is under a sortie.

The move is visually represented in Fig. 4.9 where edges $3-2$ and $5-8$ are removed, so edges $3-5$ and $2-8$ are created, reversing the path between 5 and 2.

4.2.3 Solution evaluation

The last thing that remains to discuss about the implementation is the way to evaluate the solution.

There are two ways to evaluate a solution, that is, to compute the time at which both truck and drone are at the final depot:

- A complete evaluation of the solution from the starting depot to the end depot;
- Delta evaluation to find the cost of the move performed.

The complete evaluation method is easier to code and provides consistent results from the get go. It is, however, by its own nature redundant, in that it computes times for bits of the solution that stay untouched. This very characteristic makes it $O(n)$ both in best and worst case scenario.

On the other hand, the delta evaluation is only in the worst case scenario $O(n)$, whereas in the best case it is $O(1)$. It is possible to evaluate the sole cost of the move, generalizing it starting from the way it is generically performed and going more and more into detail until all possible cases are covered in the cost computation.This is what has been done, and in Alg. 4.1 Line 11 the final way to compute the cost of a tentative solution is by means of the cost of its predecessor plus the cost of the move.

Following is a list of cases that might happen after a move `relocateCustomer()` and how the cost for that particular case is computed.

A common notation for all these cases will be having i = previous node in the old solution, j = moved node, k = next node in the old solution, α = previous node in the new solution and β = next node in the new solution. For the mixed customers, moreover, A = launch node of a rendezvous sortie and B = rendezvous node of a launch sortie. The old solution will be denoted with $oldS$ and the new one with $newS$. There is $t^{T,D}\left(sortie\right)|_S$ which stands in for the max formulas seen in Sec. 3.6 to calculate the truck and drone times for the path from the *sortie*'s launch to its rendezvous in solution S and which is shown in Eq. (4.2.3). Finally $t^T\left(x,y\right)|_S$ is just an abbreviated form to indicate the truck's time between nodes x and y in the denoted solution S, which is shown in Eq. (4.2.4).

$$t^{T,D}\left(sortie\right)|_S = \max\{\tau^D_{sortie.l,sortie.d} + \tau^D_{sortie.d,sortie.r} + SR, \tau^T_{sortie.l,\dots} + \tau^T_{\dots,\dots} + \tau^T_{\dots,sortie.r} + SR\} \quad \text{evaluated in solution S} \tag{4.2.3}$$

$$t^T\left(x,y\right)|_S = \tau^T_{x,\dots} + \tau^T_{\dots,\dots} + \tau^T_{\dots,y} \quad \text{evaluated in solution S} \tag{4.2.4}$$

Truck-only to truck This kind of move is the one that resembles the most classical TSP delta evaluation: the main difference in this case is that the moved node can be under a sortie either in the old or in the new solution, so different contributions are added to the final cost based on that.

If j in the old solution isn't under a sortie ($oldOS$), the final cost has contribution c_1 added to it, else it has c_2. Similarly, if j in the new solution isn't under a sortie ($newOS$), the final cost has contribution c_3 added to it, else it has c_4. The final cost is thus always comprised of two contributions.

$$
\begin{aligned}
c_1 &= [\tau^T_{i,k}] - [\tau^T_{i,j} + \tau^T_{j,k}] \\
c_2 &= [\max\{t^{T,D}\ (oldOS)|_{newS}\}] - [\max\{t^{T,D}\ (oldOS)|_{oldS}\}] \\
c_3 &= [\tau^T_{\alpha,j} + \tau^T_{j,\beta}] - [\tau^T_{\alpha,\beta}] \\
c_4 &= [\max\{t^{T,D}\ (newOS)|_{newS}\}] - [\max\{t^{T,D}\ (newOS\ )|_{oldS}\}]
\end{aligned}
$$

In the case of Fig. 4.10, for example, customer j is moved from a position not under a sortie (use c_1) to one which is in this condition (use c_4), thus the cost for this move would explicitly be ($newOS$ being the blank delivery with launch in k and rendezvous in β):

$$
c = [\tau^T_{i,k} + \max\{t^{T,D}\ (newOS)|_{newS}\}] - [\tau^T_{i,j} + \tau^T_{j,k} + \max\{t^{T,D}\ (newOS\ )|_{oldS}\}]
$$

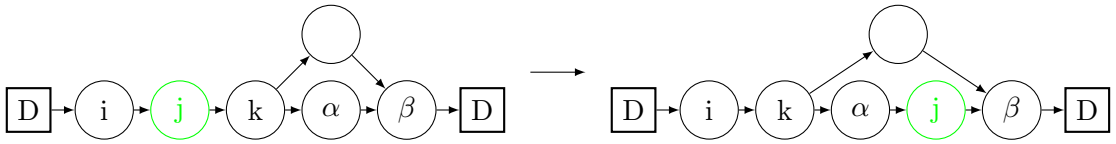


Fig. 4.10: Example of truck-only to truck for delta evaluation

Truck-only to drone In this case there are three possible situations in total:

1. Node j in the old solution is under a sortie ($oldOS$);
2. The edge $i-k$ is under the newly created drone sortie ($newDS$);
3. The edge $i-k$ is anywhere else in the solution.

For each one of these situations a corresponding c is here elencated:

$$
\begin{aligned}
c_1 &= [SL + \max\{t^{T,D}\ (newDS)|_{newS}\} + \max\{t^{T,D}\ (oldOS)|_{newS}\}] + \\
&\quad - [t^T\ (\alpha,\beta)|_{oldS} + \max\{t^{T,D}\ (oldOS)|_{oldS}\}] \\
c_2 &= [SL + \max\{t^{T,D}\ (newDS)|_{newS}\}] - [t^T\ (\alpha,\beta)|_{oldS}] \\
c_3 &= [\tau^T_{i,k} + SL + \max\{t^{T,D}\ (newDS)|_{newS}\}] - [\tau^T_{i,j} + \tau^T_{j,k} + t^T\ (\alpha,\beta)|_{oldS}]
\end{aligned}
$$

In the case of Fig. 4.11, for example, customer j is moved from a position not under a sortie (excluded the use c_1) to a drone sortie which has $i-k$ under it (use c_2), thus the cost for this move would explicitly be c_2, where $newDS$ is $\langle\alpha, j, \beta\rangle$).

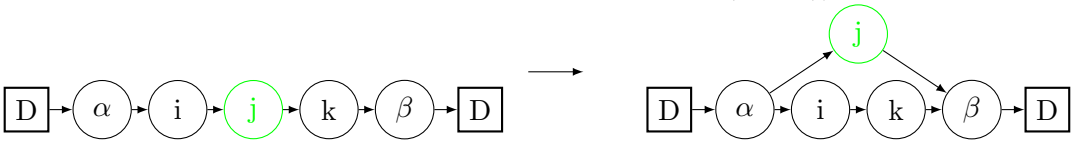


Fig. 4.11: Example of truck-only to drone for delta evaluation

Drone-only to truck In this case as well there are three possible situations in total, but different from the previous ones:

1. Node j in the new solution is under a sortie ($newOS$);
2. Node j is between i and k in the new solution;
3. Node j is not in the two previous conditions.

For each one of these situations a corresponding c is here elencated:

$$
\begin{aligned}
c_1 &= [\max\{t^{T,D}\ (newOS)|_{newS}\} + t^T\ (i,k)|_{newS}] + \\
&\quad - [\max\{t^{T,D}\ (newOS)|_{oldS}\} + SL + \max\{t^{T,D}\ (oldDS)|_{oldS}\}] \\
c_2 &= [t^T\ (i,k)|_{newS}] - [SL + \max\{t^{T,D}\ (oldDS)|_{oldS}\}] \\
c_3 &= [\tau^T_{\alpha,j} + \tau^T_{j,\beta} + t^T\ (i,k)|_{newS}] - [\tau^T_{\alpha,\beta} + SL + \max\{t^{T,D}\ (oldDS)|_{oldS}\}]
\end{aligned}
$$

Drone-only to drone The cost of this move can only be calculated in one way, and the computation is based on the old and the new drone sortie (respectively $oldDS$ and $newDS$):

$$
\begin{aligned}
c &= [\max\{t^{T,D}\ (newDS)|_{newS}\} + t^T\ (oldDS)|_{newS}] + \\
&\quad - [\max\{t^{T,D}\ (oldDS)|_{oldS}\} + \max\{t^{T,D}\ (newDS)|_{oldS}\}]
\end{aligned}
$$

Launch sortie and rendezvous sortie A mixed customer with both a launch and a rendezvous sortie has a straightforward formula for its move cost:

$$
\begin{aligned}
c &= [\max\{t^{T,D}\ (A,j)|_{newS}\} + \max\{t^{T,D}\ (j,B)|_{newS}\}] + \\
&\quad - [\max\{t^{T,D}\ (A,j)|_{oldS}\} + \max\{t^{T,D}\ (j,B)|_{oldS}\}]
\end{aligned}
$$

Normal launch sortie For a launch sortie which hasn't been reversed, either edge $i-k$ is before the launch of the launch sortie in the new solution (i.e. j), so c_1 is the cost to use, or else c_2 should be used.

$$
\begin{aligned}
c_1 &= [t^T\ (i,j)|_{newS} + \max\{t^{T,D}\ (j,B)|_{newS}\}] - [\tau^T_{i,j} + \max\{t^{T,D}\ (j,B)|_{oldS}\}] \\
c_2 &= [\tau^T_{\alpha,j} + \max\{t^{T,D}\ (j,B)|_{newS}\}] - [t^T\ (\alpha,j)|_{oldS} + \max\{t^{T,D}\ (j,B)|_{oldS}\}]
\end{aligned}
$$

In the case of Fig. 4.12, for example, customer j is moved to position 1 in the solution, which makes edge $i-k$ be under the launch sortie, thus the cost for this move would be c_2. Please note that in this case α is the initial depot.

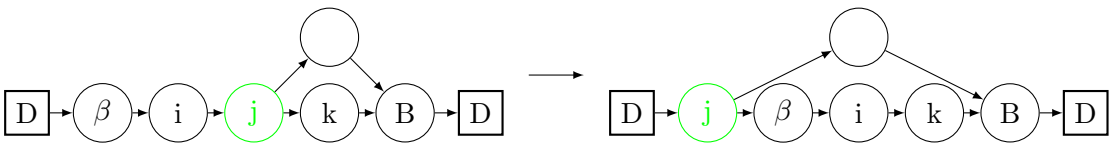


Fig. 4.12: Example of normal launch sortie for delta evaluation

Reversed launch sortie If the launch sortie is reversed, the formula changes instead to:

$$
\begin{aligned}
c &= [t^T\ (i,B)|_{newS} + \max\{t^{T,D}\ (B,j)|_{newS}\} + \tau^T_{j,\beta}] + \\
&\quad - [\tau^T_{i,j} + \max\{t^{T,D}\ (j,B)|_{oldS}\} + t^T\ (B,\beta)|_{oldS}]
\end{aligned}
$$

In the case of Fig. 4.13, for example, customer j is moved in a position that makes the sortie reverse (position 6). Please note that in this case β coincides with the end depot.

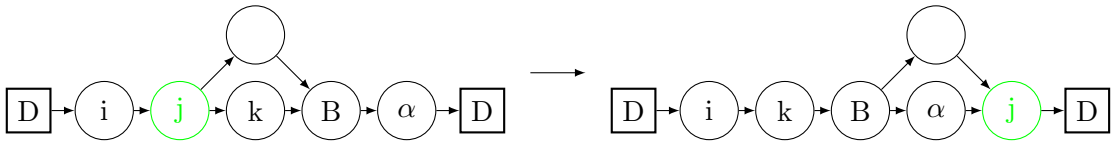


Fig. 4.13: Example of normal launch sortie for delta evaluation

Normal rendezvous sortie These formulas are perfectly symmetrical to the ones devised for launch sorties, and are written for completeness' sake. For a non-reversed rendezvous sortie, either edge $i-k$ is after the rendezvous of the rendezvous sortie in the new solution (which means j), so c_1 shall be used in the computations, or else the choice must go on c_2.

$$c_1 = [\max\{t^{T,D}(A,j)|_{newS}\} + t^T(j,k)|_{newS}] - [\max\{t^{T,D}(A,j)|_{oldS}\} + \tau^T_{j,k}]$$
$$c_2 = [\max\{t^{T,D}(A,j)|_{newS}\} + \tau^T_{j,\beta}] - [\max\{t^{T,D}(A,j)|_{oldS}\} + t^T(j,\beta)|_{oldS}]$$

Reversed rendezvous sortie This formula is perfectly symmetrical to the reversed launch sortie case, and is reported for exhaustiveness:

$$c = [\tau^T_{\alpha,j} + \max\{t^{T,D}(j,A)|_{newS}\} + t^T(A,k)|_{newS}] +$$
$$- [t^T(\alpha,A)|_{oldS} + \max\{t^{T,D}(A,j)|_{oldS}\} + \tau^T_{j,k}]$$

Chapter 5

Computations and results

Test instances (5.1) discusses in detail the generated instances used for testing. As any other Simulated Annealing implementation, the coded SA needs a parameter tuning, described in Sec. Parameter tuning (5.2). In particular the tuning is divided in two main steps: Problem selection (5.2.1) and Temperature selection (5.2.2).

Finally all the experiments run on the model are reviewed in Experiments and results (5.3), with an explanation of the setup of each experiment and the analysis of the results found.

5.1 Test instances

Test instances have been generated to reflect the following criteria.

The number of customers in the instances were chosen to analyze the difference between optimal solutions found by the mathematical model and the results of the Simulated Annealing and to reflect real world scenarios: the smaller instances (5, 6, 7, 8, 9 and 10 customers) are the ones also solvable to optimality, the bigger ones (50, 100, 150 and 200 customers) reflect real world cases. For each of these number of customers, 5 instances were generated, for a total of 50 different instances.

Every instance has a map of 20 $mi \times 20$ mi (see Fig. 5.1), to reflect the idea that Amazon and HorseFly drones have a 10 mi endurance and thus be able to get a sufficient amount of feasible drone sorties. In the map, the x and y coordinates are randomly generated numbers for every node except the depot, which is always set in $(0, 0)$.

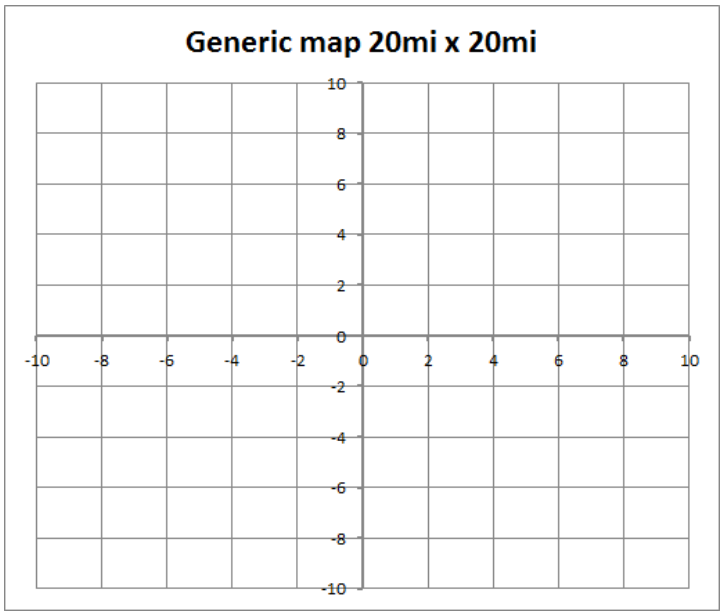


Fig. 5.1: Generic instance map

Such a map covers a rather big area and it is not unrealistic to think that it could contain 200 daily customers for some package delivery service. It is however somewhat unlikely that 200 customers can be feasibly served within one working day with one vehicle

only, even if accompanied by its own drone. In this sense, the instances with 150 and 200 customers are only to be thought of as tests for the limit of the model.

A matrix of euclidean distances between the nodes is then calculated and all other measurements are dependent on this matrix, in particular the time matrices for the drone and the truck are calculated based on this and according to Sec. 3.5's formula.

The customers excluded from drone delivery are randomly generated numbers such that for every instance there are 80% serviceable customers and 20% excluded customers. The reason for their exclusion was deemed unimportant, but it might be due to geographical limitations, too heavy parcels to deliver, or other criteria.

These instances have also been run through Concorde to find the optimal TSP tour for the resulting graph, giving a value to test against in the final analyses.

The naming found throughout this work is built to show the number of customers in the instance (`XXX`) and the progressive number of it (`Y`) such that the generic name is `Instance_XXX.Y`.

5.2 Parameter tuning

As previously discussed in Sec. 4.1.1, the Simulated Annealing needs to have its parameters tuned to work as best as possible. It is unfeasible time-wise to tune the parameters on all the instances, and it would overfit them, so a small representative sample is going to be chosen in Sec. 5.2.1 that will undergo further testing.

Some pre-tuning was done to not have terrible starting points for the necessary parameters by means of looking at the resulting graphs for iterations and current objective.

For example, after some testing it was shown that the shape of the graph looks adequate with starting temperature of 0.01 and end temperature of 0.001. Upon setting those numbers, some other simulation runs were done to find a suitable number of iterations. As an example, in Fig. 5.2 one can see that the situation doesn't really change much after 3 million iterations: tentativeObjective represent attempted solutions, while currentObjective stands for the accepted solutions.

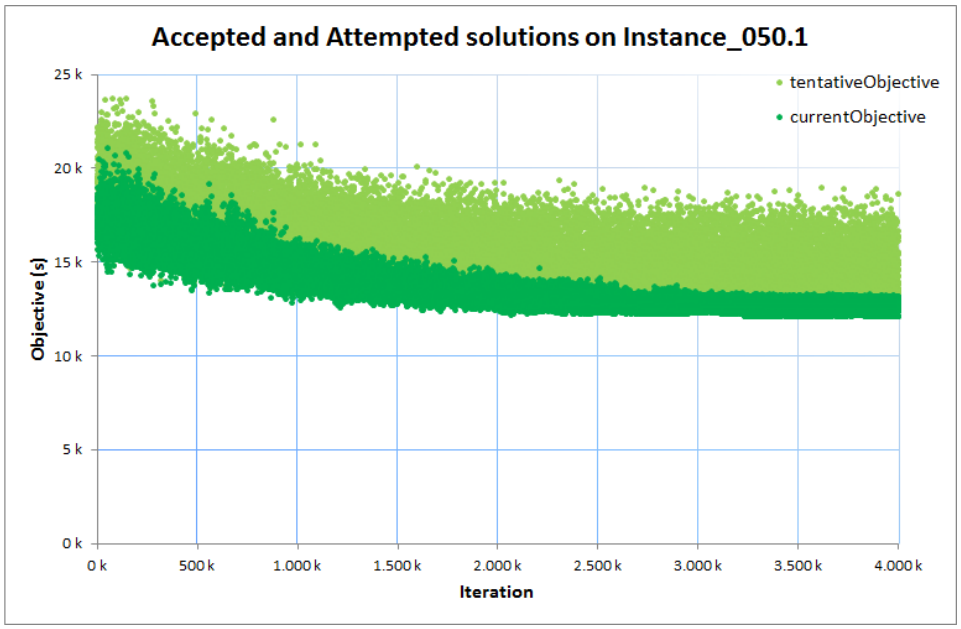


Fig. 5.2: Pre-tuning on `Instance_050.1`

To allow for some leeway, the number of iterations is thus set to 4 million. This is also a compromise number chosen to be able to explore an adequate amount of the solution

space even for the 200-customers instances in a reasonable amount of time. A rough estimate for the neighborhood size often used in the TSP field is to square the number of nodes in a graph. This means that a 200 nodes graph has a neighborhood size of roughly 40˙000, which consequently means that 4 million iterations could fully explore 100 neighborhoods. This is a slight overestimation for TSP neighbourhoods, since they are exacty large $n \times n - 1$, whereas for the FSTSP it is probably an underestimation: the real value for the neighborhood size would lie between n^2 and n^3 due to the possibility of also having drone sorties as possible positions for the customers.

The main ideas in parameter tuning are to find a set of representative instances among all the supplied ones, and tune all the other parameters on those selected few. Since the smaller instances are more of a reference for the mathematical model than a real struggle to find the optimal solution for, only the 10-customer ones will be considered in the population of instances for tuning, out of those: the final population will thus consist of the instances with 10, 50, 100, 150 and 200 customers, five of each type.

The parameter tuning will follow this hard criteria, regardless of the results obtained in each step:

1. Start with $It = 4\dot{0}00\dot{0}00$, $T_s = 0.01$ and $T_f = 0.001$;
2. Run ten times each instance, to obtain ten objective values describing how Simulated Annealing works on every one of them;
3. Given μ = sample mean and σ = sample standard deviation, compute for each instance the *Coefficient of Variation* $c_v = \sigma/\mu$;
4. Order the instances in decreasing c_v values and pick the three quartiles as the chosen representative instances; since the data set is made of 25 instances, the chosen ones will be #7, #13, and #19 in the ordered list;
5. Briefly search for a good enough pair of (T_s, T_f) for each of the three instances;
6. Create 3 matrices A^1, A^2 and A^3 (one for each instance) where every element is the average objective out of ten runs with a pair of parameters from the 5×5 set;
7. Given that the smallest objective found for an instance in all the 250 runs is named z^{**}, create the final R matrix with generic element $r_{i,j} = \dfrac{\frac{a^1_{i,j}}{z^{**}_1} + \frac{a^2_{i,j}}{z^{**}_2} + \frac{a^3_{i,j}}{z^{**}_3}}{3} - 1$;
8. The smallest value in R defines the final pair of (T_s, T_f).

Steps $1 - 4$ are treated in Problem selection (5.2.1) and steps $5 - 8$ in Temperature selection (5.2.2).

5.2.1 Problem selection

Three out of the twentyfive instances have to be chosen as a representative sample to tune the start and end temperature for the algorithm. The deciding factor should be the difficulty of a given instance: an instance is considered difficult if all the computed objectives for that instance are widely different from one another (no clear objective value seems to possibly represent the global minimum) and is on the other hand considered easy if all the computed objectives are equal to each other, since this seems to mean that that value is also the global minimum.

However, there is a problem: the instances have a different scale, that is to say, they present widely different mean objective. Hence the need for an objective and standardized

indicator, a dimensionless number representative of each instance's difficulty level, apt to compare them in a meaningful way.

The *Coefficient of Variation* (see Eq. (5.2.1)), also known as *Relative Standard Deviation*, is a standardized measure showing the variability (represented by standard deviation σ) of a probability distribution in relation to the mean of the population (μ), and is expressed as a percentage value.

$$c_v = \frac{\sigma}{\mu} \tag{5.2.1}$$

The actual value of the c_v is dimensionless, in that it is independent of the unit in which the measurement has been taken. This is why one should use the coefficient of variation instead of the standard deviation as such a means of comparison between data sets with widely different means.

The aggregate results of these ten runs for the twentyfive instances are shown in Table 5.1, whereas in Table 5.2 the reader can see the list of instances ordered over c_v in a decreasing fashion. As it can be seen in the table by the bold highlighting, the representative instances are `Instance_050.3` as the first quartile (#7), `Instance_100.4` as the median (#13), and finally `Instance_100.3` as the third quartile (#19).

Instance	Obj μ	Obj σ
010.1	5986,71	0,00
010.2	6394,39	0,00
010.3	6310,60	0,00
010.4	8377,92	0,00
010.5	9108,50	120,13
050.1	12548,87	260,52
050.2	12213,58	227,19
050.3	12480,34	155,06
050.4	12847,23	265,74
050.5	12159,30	326,97
100.1	18077,53	537,73
100.2	17261,93	458,88
100.3	17280,89	383,73
100.4	18506,22	323,94
100.5	17354,73	371,76
150.1	22946,84	245,12
150.2	22524,38	561,41
150.3	23326,29	555,44
150.4	22576,28	562,23
150.5	23304,92	367,70
200.1	27136,32	435,50
200.2	28769,93	264,42
200.3	27292,17	366,51
200.4	28668,55	476,89
200.5	27976,75	517,83

Tab. 5.1: Aggregate results for the problem selection: 10 runs × 25 instances

#	Instance	Obj c_v
1	010.1	0,00%
2	010.2	0,00%
3	010.3	0,00%
4	010.4	0,00%
5	200.2	0,92%
6	150.1	1,07%
7	**050.3**	1,24%
8	010.5	1,32%
9	200.3	1,34%
10	150.5	1,58%
11	200.1	1,60%
12	200.4	1,66%
13	**100.4**	1,75%
14	200.5	1,85%
15	050.2	1,86%
16	050.4	2,07%
17	050.1	2,08%
18	100.5	2,14%
19	**100.3**	2,22%
20	150.3	2,38%
21	150.4	2,49%
22	150.2	2,49%
23	100.2	2,66%
24	050.5	2,69%
25	100.1	2,97%

Tab. 5.2: Results ordered over c_v for the problem selection

5.2.2 Temperature selection

Some good enough values for T_s and T_f give a headstart on where to focus more on the search for the temperature tuning. With those, two equally distributed ranges of five temperatures are produced: [0.0080, 0.0090, 0.0100, 0.0110, 0.0120] for T_s and [0.0006, 0.0007, 0.0008, 0.0009, 0.0010] for T_f.

Every combination of the T_s and T_f is run ten times. The average objective of every combination is used to fill a 5×5 matrix, as can be seen in Fig. 5.3 for `Instance_050.3` (A^1), Fig. 5.4 for `Instance_100.4` (A^2), and Fig. 5.5 for `Instance_100.3` (A^3). In those figures, the conditional formatting highlights the best results as the darkest shade of green.

050.3	0,0010	0,0009	0,0008	0,0007	0,0006	endTemp
0,0120	12448,463	12503,188	12428,781	12467,886	12483,051	
0,0110	12503,858	12385,169	12333,593	12439,421	12541,041	
0,0100	12529,341	12474,572	12532,415	12516,824	12470,557	
0,0090	12533,833	12346,740	12617,588	12672,572	12520,577	
0,0080	13080,649	12723,016	12870,289	13241,707	12950,206	
startTemp						

Fig. 5.3: Temperature tuning for `Instance_050.3`

100.4	0,0010	0,0009	0,0008	0,0007	0,0006	endTemp
0,0120	18563,568	18631,746	18628,298	18597,421	18708,648	
0,0110	18912,593	18654,181	18733,106	18653,917	18599,673	
0,0100	18675,487	18718,567	18529,412	18594,720	18549,779	
0,0090	18629,205	18662,292	18504,491	18729,494	18577,809	
0,0080	18980,458	18465,577	18761,149	18595,791	18380,988	
startTemp						

Fig. 5.4: Temperature tuning for `Instance_100.4`

100.3	0,0010	0,0009	0,0008	0,0007	0,0006	endTemp
0,0120	17361,379	17390,181	17182,984	17544,899	17538,507	
0,0110	17430,565	17434,380	17388,835	17515,204	17562,525	
0,0100	17484,278	17461,592	17230,798	17433,905	17264,329	
0,0090	17383,230	17283,171	17466,112	17322,945	17247,221	
0,0080	17261,380	17472,188	17377,552	17534,213	17432,335	
startTemp						

Fig. 5.5: Temperature tuning for `Instance_100.3`

To finally choose the set of definitive temperatures, let z^{**} be the minimum objective value of the 25×10 runs for each matrix. The three values of z^{**} are found to be $z_1^{**} = 11907.865$, $z_2^{**} = 17743.975$, $z_3^{**} = 16669.008$.

With these three values and the three old matrices A^1, A^2, and A^3, the matrix R can be created, where every element is represented in Eq. (5.2.2).

$$r_{i,j} = \frac{\frac{a_{i,j}^1}{z_1^{**}} + \frac{a_{i,j}^2}{z_2^{**}} + \frac{a_{i,j}^3}{z_3^{**}}}{3} - 1 \tag{5.2.2}$$

The final matrix is presented in Fig. 5.6, where it is seen that the best pair of parameters is $T_s = 0.0120$ and $T_f = 0.0008$. The numbers in it, represent the mean gap between the

average result for that combination of parameters and the overall best objective obtained from the 250 runs. Lower average gap is better.

R	0,0010	0,0009	0,0008	0,0007	0,0006	endTemp
0,0120	4,437%	4,776%	4,147%	4,922%	5,161%	
0,0110	5,387%	4,577%	4,489%	4,890%	5,167%	
0,0100	5,120%	5,002%	4,347%	4,832%	4,279%	
0,0090	4,843%	4,182%	5,009%	5,300%	4,438%	
0,0080	6,790%	5,244%	6,022%	7,064%	5,641%	
startTemp						

Fig. 5.6: Results matrix for the temperature tuning

5.3 Experiments and results

With a tuned set of parameters, it is possible to run some experiments:

1. Main reference experiment (5.3.1): run the algorithm ten times on every instance;
2. Endurance changes (5.3.2): what would happen if the drone's endurance was changed;
3. Top speed changes (5.3.3): investigation of what to expect if the drone's top speed is changed;
4. Different number of iterations (5.3.4): exploring the possibility of having set another number of iterations;
5. Distribution of the objective function (5.3.5): analysis on how is the objective function statistically distributed;
6. Different acceptance probability (5.3.6): study on how the algorithm would work if the acceptance function was the one of *Record-to-Record Travel*.

5.3.1 Main reference experiment

The main contribution of this thesis is to ultimately see if Simulated Annealing can be a feasible means of finding a good solution to the FSTSP. To test this hypotesis and provide some evidence to this claim, the main experiment aims at running the SA over an adequate enough number of instances with different customer numbers that were previously described in Sec. 5.1.

In particular, the experiment will be done by running the SA ten times for every one of the 50 self-generated instances. For every instance the average best objective of each run is calculated and compared with both the optimal results from the Mathematical Model and, for the bigger instances, with the optimal TSP tour. Doing only ten runs will provide only a rough estimate of the results, but good enough to be able to see trends in the outcomes.

Results for the first 30 instances are presented in Tab. 5.3. From the table's column $\text{Obj}^{GAP\%}$, which is calculated as $\dfrac{\text{Obj}\mu - \text{Obj}_{OPT}}{\text{Obj}_{OPT}}$ for each instance, it appears evident that the SA can certainly reach optimality even with only 20 min of total runtime (runtime that can of course be improved with a better implementation). The handful of cases in which the column displays a number different from 0 is due to one "bad-run" in every case: since the average is greatly influenced by outliers the resulting gaps are relatively big, but

the outlier can easily be identified as such in these cases, so nothing is to be feared from this.

For every 10-run batch T^{SA}_{toBest} μ (s) represents the time in seconds required to the SA to get to the best solution found and T^{SA}_{total} μ (s) is the total execution time. The time for the execution of the Mathematical Model is instead showed in column T^{MM}_{total} (s). The ratio T^{SA}/T^{MM}, is thus shown in the last two columns, respectively for the runtime to get to the best solution and for the total runtime.

Looking in parallel at Tab. 5.3 at Fig. 5.7, where ratios are presented in a logarithmic scale, the reader can see both ratios follow the same trend, with the average gap between ratios decreasing with an increase in instance size. That trend appears to be one of a clear and strong increase in the time saved by running the SA instead of the MM. For the very small instances ratios are substantial, but this is due to both having parameters tuned for larger instances and to the fact that the total runtime is taken into consideration. If the focus is switched to the runtime to best, then the difference drastically decreases even at small sizes: the best solution is found in very little time for these smaller instances, even if slightly bigger than the time used by CPLEX.

Tab. 5.3: Instances' results whose optimal solutions can be computed in acceptable time

Instance	Obj μ	T^{SA}_{toBest} μ (s)	T^{SA}_{total} μ (s)	Obj_{OPT}	T^{MM}_{total} (s)	$\text{Obj}^{GAP\%}$	T^{ratio}_{toBest}	T^{ratio}_{total}
005.1	4456,832	5,48	280,15	4456,83	0,98	0,00%	5,59	285,87
005.2	3507,068	62,97	373,59	3507,07	0,83	0,00%	75,87	450,11
005.3	3275,689	0,01	20,59	3275,69	0,98	0,00%	0,01	21,01
005.4	5312,468	0,05	57,35	5312,47	0,42	0,00%	0,11	136,55
005.5	5510,165	16,90	578,54	5510,17	0,64	0,00%	26,40	903,97
006.1	7080,942	0,01	18,42	7080,94	0,94	0,00%	0,01	19,60
006.2	6147,956	1,30	125,26	6147,96	0,92	0,00%	1,41	136,16
006.3	6835,158	0,01	49,43	6835,16	0,86	0,00%	0,01	57,48
006.4	4402,083	3,65	322,42	4402,08	1,23	0,00%	2,97	262,13
006.5	5392,079	32,83	590,80	5392,08	1,46	0,00%	22,49	404,66
007.1	5533,853	0,46	39,12	5533,85	6,94	0,00%	0,07	5,64
007.2	5342,682	1,98	104,96	5342,68	8,54	0,00%	0,23	12,29
007.3	7783,901	8,79	47,54	7725,89	2,17	0,75%	4,05	21,91
007.4	7610,377	0,40	106,97	7610,38	2,33	0,00%	0,17	45,91
007.5	7010,989	9,05	28,49	7010,99	7,44	0,00%	1,22	3,83
008.1	6709,019	1,07	143,56	6709,02	36,78	0,00%	0,03	3,90
008.2	6587,178	0,10	48,26	6587,18	55,44	0,00%	0,00	0,87
008.3	5780,115	53,24	79,41	5780,12	47,14	0,00%	1,13	1,68
008.4	6914,785	0,10	38,84	6505,12	52,02	6,30%	0,00	0,75
008.5	5953,510	7,61	66,87	5953,51	70,61	0,00%	0,11	0,95
009.1	7338,773	0,76	42,56	7338,77	333,33	0,00%	0,00	0,13
009.2	6204,630	39,82	64,12	6204,63	377,46	0,00%	0,11	0,17
009.3	7698,112	16,80	31,96	7698,14	418,6	0,00%	0,04	0,08
009.4	6817,718	0,06	70,04	6817,72	245,47	0,00%	0,00	0,29
009.5	7802,665	6,64	38,99	7802,67	495,74	0,00%	0,01	0,08
010.1	5986,712	1,07	73,62	5986,71	18541,91	0,00%	0,00	0,00
010.2	6394,386	37,42	81,68	6394,39	80298,63	0,00%	0,00	0,00
010.3	6585,438	56,06	134,14	6310,60	25097,39	4,36%	0,00	0,01
010.4	8377,917	1,23	249,87	8377,92	3791,53	0,00%	0,00	0,07
010.5	8984,151	4,47	34,26	8934,41	6970,96	0,56%	0,00	0,00

An attentive reader might notice that runtimes show weird spikes in the smaller instances, in particular in Tab. 5.3 with instances 005.1, 005.2, 005.5, 006.4, 006.5, for example. These very high values for T^{SA}_{total} μ are to be ascribed to a simple but unaccounted for phenomenon. In running Simulated Annealing for these smaller instances, the combination of possible feasible flights are extremely limited, due to the nature of the maps: they have very few customers distributed in a very large area, so the drone is very limited in its flight possibilities. This leads the algorithm to get stuck in loops that are theoretically infinite when the chosen move is `moveRelocateCustomer` and the chosen transportation means is the drone, in some configurations. What happens is that SA tries to find a feasible sortie for a given customer that only has one or two sorties theoretically

feasible but probably infeasible in a given permutation of the customers due to mostly the endurance constraint, or maybe even the chance of having prohibited moves. What this meant was infinite loops. To break this behaviour, a counter was implemented to block the tries that lead to these loops and change delivery method at first, and customer after some more tries. The limit to the looping was set by the author to be 42. This number wasn't chosen by chance, but it was at the time observed that most loops would eventually solve themselves, if actually feasible, in 30-40 iterations, and would otherwise proceed into an infinite loop otherwise. Due to lack of time in the possibilities of testing, a rigorous paramter tuning was not performed. It can be however inferred that this wouldn't have changed much, since a smaller number would have of course limited the time elapsed in these "broken loops" for these smaller instances, but it would have severely limited the chances of a sortie to be found in the bigger instances! A bigger limit to the iterations, would have instead meant even longer total runtimes for smaller instances but maybe more chances of finding usable sorties in bigger instances, and thus an improved solution quality. This parameter tuning and testing is left to future research, but might reveal itself to be quite promising, as the problem is not a small one.

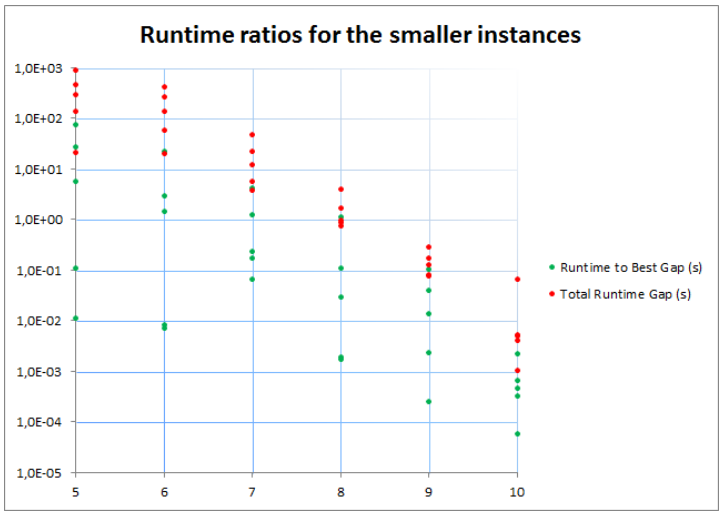


Fig. 5.7: Runtime for mathematical model and related runtime for the SA

Tab. 5.4 presents the results for the bigger 20 instances. The table's column Obj_{TSP} is splitted in two where the first represents the total distance of the TSP tour in miles, whereas the second column converts that result in seconds using the usual formula $t = \dfrac{d}{\text{top speed} * \nu}$.Column Saving % shows the saving from the TSP tour to the SA average solution of ten runs.

This value, calculated as $\dfrac{\text{Obj}_{TSP} - \text{Obj}\mu}{\text{Obj}_{TSP}}$, is also plotted in Fig. 5.8 where the final plot assumes quite the interesting shape. There seems to be a maximum saving in the instances with 50 and 100 customers. This result is to be taken with a grain of salt, as the sample size is relatively small and more testing should be done to gain a better insight, but anyways there are a few ideas that could explain such a result, if confirmed. First of all, the temperatures were tuned exactly on those two sizes of instances, and for how much they should be representative of the whole population, the method used still ended up favoring — if just for a bit — instances with that number of customers. Moreover, it can be speculated that, since the map stays the same size for all the instances, those customer quantities might be the ones that "fill" the map in a better way so as to favour the contributions of drone deliveries in the best way possible. On the opposite hand, this would mean that the 10 customer instances are too filled with void between the customers,

Tab. 5.4: Instances' results whose TSP tour can be obtained with Concorde

Instance	Obj μ	T^{SA}_{toBest} μ (s)	T^{SA}_{total} μ (s)	Obj_{TSP} (dist)	Obj_{TSP} (time)	Saving %
010.1	5986,712	1,07	73,62	77.972	6229,23	3,89%
010.2	6394,386	37,42	81,68	85.314	6815,79	6,18%
010.3	6585,438	56,06	134,14	103.555	8273,07	20,40%
010.4	8377,917	1,23	249,87	106.595	8515,94	1,62%
010.5	8984,151	4,47	34,26	120.319	9612,35	6,54%
050.1	12518,933	71,54	213,87	179.629	14350,67	12,76%
050.2	12475,138	91,11	208,36	180.980	14458,60	13,72%
050.3	12664,646	102,77	191,04	179.221	14318,07	11,55%
050.4	12908,182	98,72	184,85	184.566	14745,09	12,46%
050.5	12164,831	91,46	189,86	182.311	14564,93	16,48%
100.1	17974,852	247,24	267,42	247.879	19803,20	9,23%
100.2	17342,184	244,04	272,35	246.779	19715,32	12,04%
100.3	17181,877	240,67	265,45	245.895	19644,69	12,54%
100.4	18538,033	249,87	266,75	256.098	20459,82	9,39%
100.5	17407,425	287,34	312,77	248.568	19858,24	12,34%
150.1	22823,376	346,52	365,04	306.910	24519,22	6,92%
150.2	22549,553	373,08	383,72	304.745	24346,26	7,38%
150.3	23114,136	368,01	379,99	313.624	25055,60	7,75%
150.4	22650,995	366,37	382,67	300.891	24038,36	5,77%
150.5	22807,414	366,77	384,69	308.441	24641,53	7,44%
200.1	26991,206	446,85	456,74	339.843	27150,26	0,59%
200.2	27848,141	439,72	452,88	354.284	28303,95	1,61%
200.3	27143,783	493,69	510,11	345.488	27601,24	1,66%
200.4	28503,176	507,95	517,44	360.906	28832,99	1,14%
200.5	27875,865	498,87	515,30	356.213	28458,06	2,05%

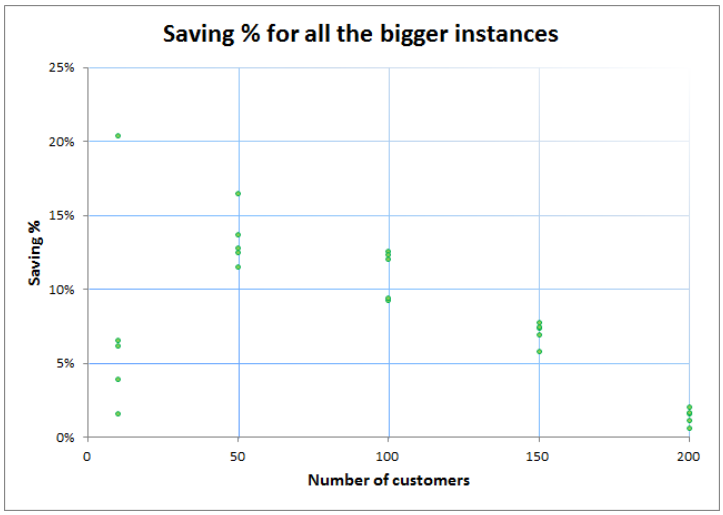


Fig. 5.8: Saving between optimal TSP tour and best SA solution found in 10 runs

which shouldn't be a problem, and the biggest instances would be too filled with customers. Regarding both statements, more insight can be gained by finding data on the usage of the drone for the different instance sizes, the average number of drone sorties per solution, the total and average distance flown by the drone compared to the total and average distance travelled by the truck, and the closeness to the endurance limit set for the drone.

In Fig. 5.9, Fig. 5.10 and Fig. 5.11 these critical points are analyzed and shown to confirm the prediction made in the previous paragraph. Fig. 5.9 shows that the number of sorties per instance size increases in a quasi-parabolic fashion, while the instance size remains the same. This hints at a saturation of the map with too many customers for the larger sizes, especially if Fig. 5.10 is also taken into account.

In this second graph, what earlier seemed to be a parabola is here found to have an asymptote, or what really looks like one. Bigger instances, as the ones here created, cram the customers more, leading to a worsening of the drone usage with bigger sizes. It seems

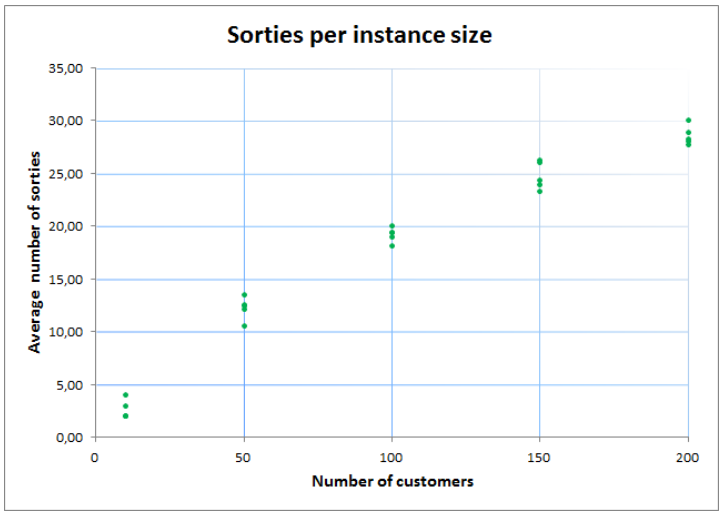


Fig. 5.9: Average number of sorties for a given number of customers

clear that the marginal gain for the use of drones decreases the more a map is filled with customers, for the way they are built in this study. The endurance limit, which was almost reached for smaller instances, is now far from saturation. In fact, the drone is used for less than 50% of its battery for 100, 150 and 200 customers. This means that shorter flights are performed, because the distance between the customers is now shorter due to having the same map size with more customers. Despite this, the savings graph (Fig. 5.8) shows still a nice saving for the 100 customer instances, and degrades a lot only at the 150 and especially the 200 mark.

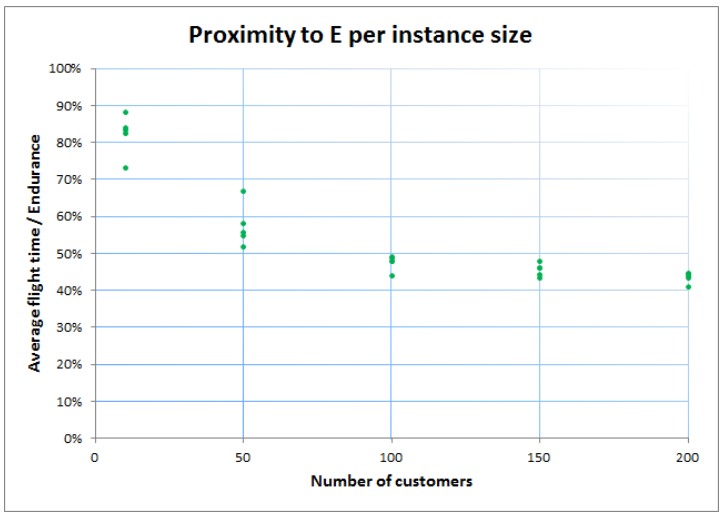
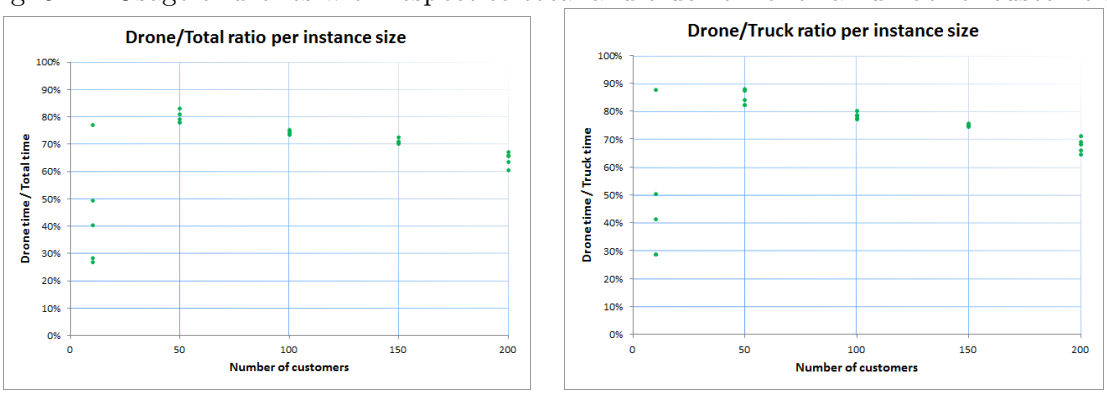


Fig. 5.10: Use of the drone's endurance for a given number of customers

In Fig. 5.11, what can instead be seen is that the flight time over both the total objective time and the truck-only time decreases as well. This further seems to confirm that instances with more customers are built in somewhat of a wrong way, as customers should have more space between them to benefit more of the usage of a drone. It can be speculated that this actually mostly resembles a real-life scenario, in which deliveries are surely grouped together by area, but are still well distributed in such area.

In Fig. 5.12 runtimes are presented, and the graph matches perfectly with time complexity predictions: the Mathematical Model is solved in what appears to be non polynomial time, whereas the results for the SA resemble constant time complexity $O(n)$. In

Fig. 5.11: Usage of drones with respect to total and truck time for a number of customers



particular it is also shown that the total runtime is worse for smaller instances, as also shown earlier with Fig. 5.7, but asymptotically catches up with the time needed to find the best solution in that run.

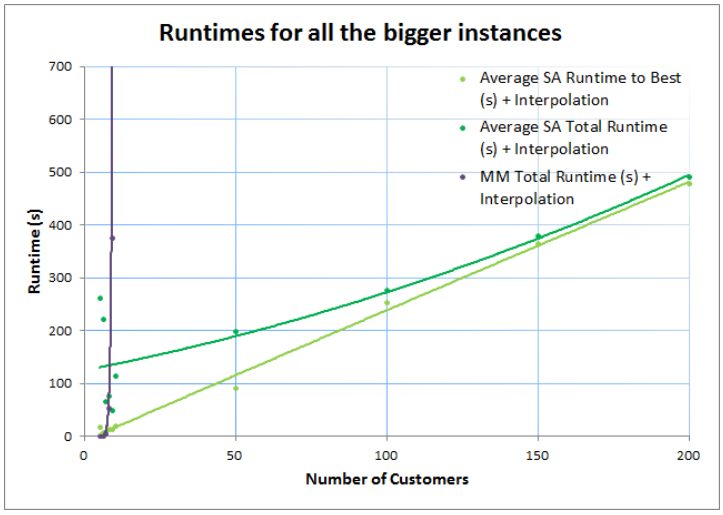


Fig. 5.12: Runtime for mathematical model versus runtimes for the SA

Finally in Fig. 5.13 and Fig. 5.14 are presented two solutions found by the SA, respectively for `Instance_010.3` and `Instance_100.4` so that the reader can best appreciate how sorties and truck would combine their work to deliver parcels together. They are compared side-by-side with the optimal TSP tour for the instance.

Fig. 5.13: Comparison of SA solution and optimal TSP tour for `Instance_010.3`

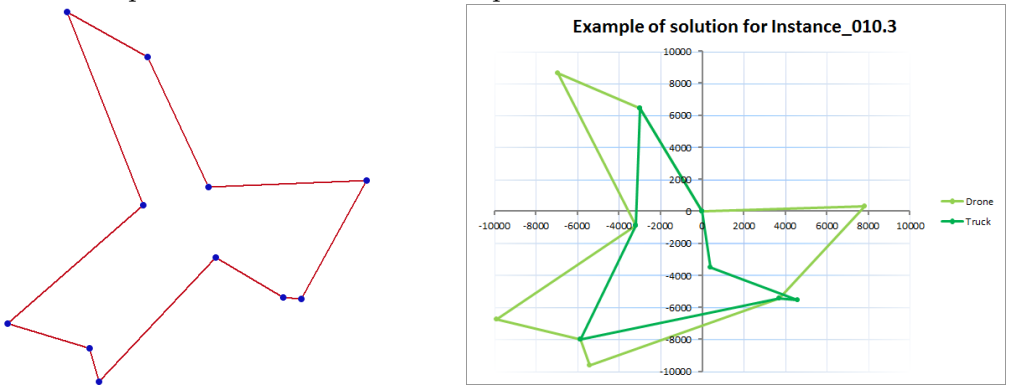
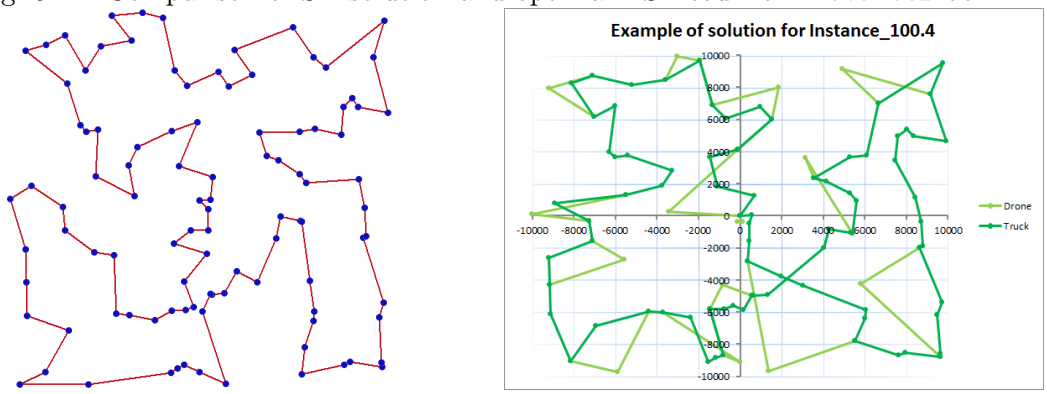


Fig. 5.14: Comparison of SA solution and optimal TSP tour for `Instance_100.4`



5.3.2 Endurance changes

A second factor of interest would be seeing if and how the objective changes with a different drone endurance. Ten runs of the three representative instances for three different endurance values should highlight a trend, if there is one. The three endurance values chosen for the test are the classical $E = 1440$ s (24 min), $E' = 900$ s (15 min) and $E'' = 1980$ s (33 min).

In Fig. 5.15 the reader can see that there appears to be an inversely proportional trend going on: the more the endurance the less the average objective function seems to be. The red line with a double arrow represents moreover the duration in seconds of the optimal TSP tour, for reference.

It can moreover be calculated that the average improvement from the smallest to the biggest value for the endurance is respectively 3.04% for `Instance_050.3`, 1.32% for `Instance_100.4`, and 2.97% for `Instance_100.3`. Considering the limited sample size, the statistical significance of these numbers is rather small, but it appears that an almost doubling of the endurance produces an average 2.5% improvement in the solution quality.

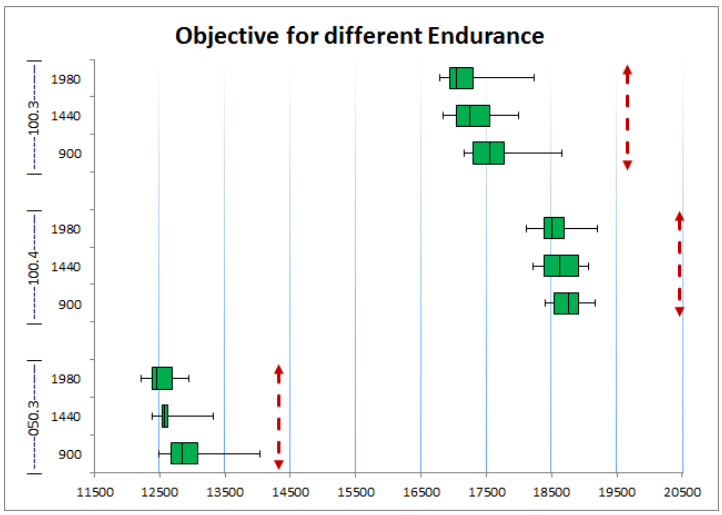


Fig. 5.15: Changes in the average objective if the drone's endurance is modified

5.3.3 Top speed changes

Along with endurance changes, another technology-influenced parameter for the drone that if changed could produce interesting results is the drone top speed. This is tested

for in the same way as for the endurance: ten runs of the three representative anstances for three different top speed values. The top speed of the drone is changed to $speed' = 40$ mph and $speed'' = 60$ mph, other than the classical value of $speed = 50$ mph.

Fig. 5.16 seems to highlight an inverse proportionality between objective function and drone speed. In fact, the less the drone speed, the worse the total time spent in the tour, and viceversa. Again, the red line with a double arrow represents the optimal TSP tour's duration in seconds, as a reference.

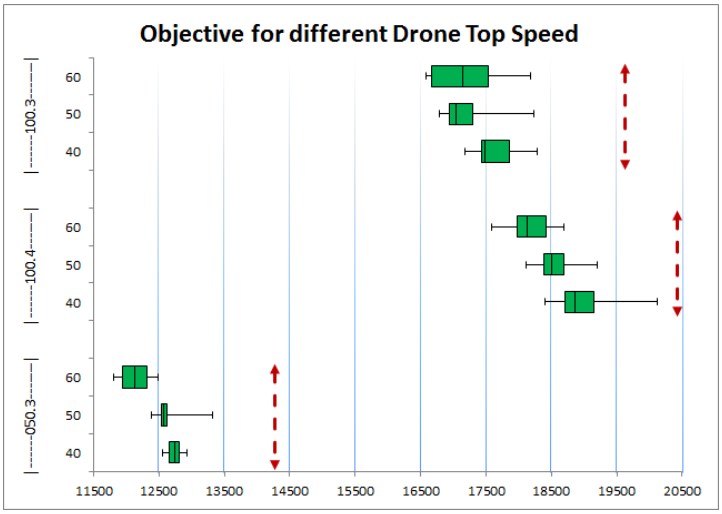


Fig. 5.16: Changes in the average objective if the drone's top speed is modified

The average improvement from the slowest to the fastest drone is estimated to be 4.80% for `Instance_050.3`, 3.87% for `Instance_100.4`, and 1.93% for `Instance_100.3`. On average, this means that a 50% increase in drone speed could imply a 3.5% improvement in the solution quality. Of course, the statistical significance of such a small sample only indicates a possible trend, and further investigation is required to enstablish proper correlation. However, with these two results, it could also be argued that improving drone speed is more important than improving its endurance, as it seems to lead to greater improvements in the solution quality.

5.3.4 Different number of iterations

An important aspect of the Simulated Annealing is the relation between number of iterations for which the algorithm runs and solution quality. In fact, with more iterations comes a more thorough neighborhood exploration, and this leads to better chances of finding a better solution. It is however a trade-off with computation time. To check if this holds true here as well, the algorithm is run ten times the algorithm over the three problems from the tuning sample, with the number of iterations changed to 2 millions and 8 millions other than the reference result with 4 millions.

As expected, the trend shows up in this case as well, with Fig. 5.17 showing that both box and wiskers in the boxplot move to smaller objectives if the number of iteration increases. For reference, there is once again a plot of the TSP optimal duration for each instance, depicted as a red double arrow.

The improvement shown from 2 million to 8 million iterations is that of a 2.77% for `Instance_050.3`, 4.73% for `Instance_100.4`, and 3.51% for `Instance_100.3`. An average figure for the improvement provided by increasing the number of iterations is again of 3.5%, but with respect to the same value found for changing the speed, this time the variance is smaller. It is a small sample, but it is showing that this seems to be the most

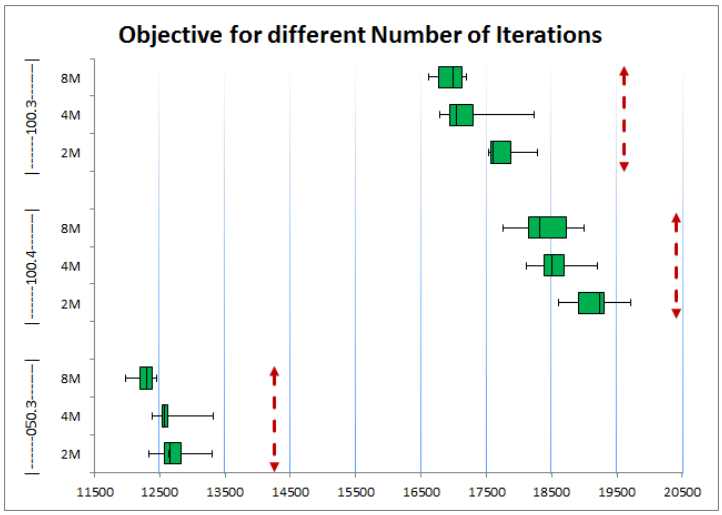


Fig. 5.17: Changes in the average objective if the number of iterations for the SA is modified

consistent method of the three to obtain better solutions.

While the two previous methods imply real-world technological advancement, this one relies barely on increasing the iterations for the SA, and with it, the runtime dedicated to finding a solution. The trade-off seems to be worth it, and the sacrifices to make to get a better solution can also be further limited by improving the algorithm.

5.3.5 Distribution of the objective function

Running the SA enough times on the same instance makes it possible to have a statistically significant sample of the objective function. 250 runs should be an adequate sample size to show how the objective is distributed in a proper way. For the analysis, the median among the problems in Tab. tab:probSelOrdered is chosen, that is `Instance_100.4`, and the parameters are all the ones from the parameter tuning performed in Sec. 5.2.

Since the distribution isn't known a priori, some descriptive statistics is used on the data from the 250 runs. Fig. 5.18 shows the data in terms of sample mean and sample standard deviation, which are respectively $\mu = 18625.67$ s and $\sigma = 385.25$ s. This first graph shows that only 4 of the 250 data points are out of the $\pm 3\sigma$ range and the results seem to be slightly positively skewed.

The boxplot in Fig. 5.19 confirms the positive skewness of the sample, and it can be argued that this means that the SA struggles to find very good solutions, which is to be expected from a tough minimization problem: if more moves were implemented and more time was available, a better tuning in accord with a more thorough neighborhood exploration would simplify the issue a little. On top of the boxplot is how the data would look if the distribution was a perfect fit for a normal curve, to be used in comparison with the following figures.

In fact for a more thorough analysis of the data and to visually verify if the data is normally distributed, two frequency bin plots are generated. In Fig. 5.21 the reader can find the plot using the bin size given by the Freedman-Diaconis rule: given $IQR =$ interquartile range and $n =$ sample size, the bin size should be $h = \dfrac{2 * IQR}{\sqrt[3]{n}}$. This measure returns a bin size of approximately $h = 164.5$ which leads to $\left\lceil \dfrac{\max - \min}{h} \right\rceil = 13$ bins . This rule is usually preferred to others due to the IQR being less sensitive to outliers than the sample standard deviation and due to the inverse proportionality to the cube root

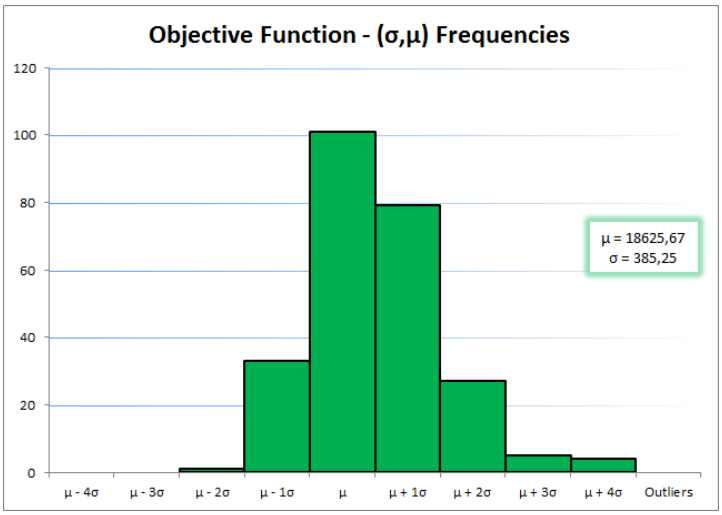


Fig. 5.18: Distribution of the sample data over the mean plus or minus three standard deviations

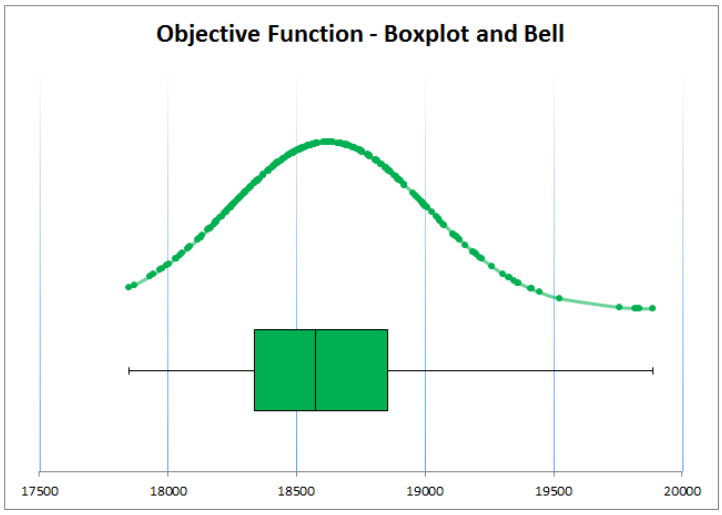


Fig. 5.19: Boxplot and bell curve for the sample data

of the sample. The resulting graph is difficult to interpret: no clear peak exists and is highlighted by the subdivision, but the data still seems to assume a quasi-bell shape.

To try and have a different point of view on the data, the simpler square root rule is used to determine the bin number instead of their size, calculating the squared root of the sample size and rounding up. This leads to $\sqrt[2]{n} = 16$ bins. The resulting histogram is shown in Fig. 5.22 where it can be seen that the uncertainty in the previous graph might be due to a double peak. This might also, however, be due to overfitting thanks to the high number of bins this second rule proposes. Except from the first and higher peak, the data would be perfectly bell-like. No reason seems to evidently explain the presence of the peak in the fifth bin.

It is also interesting to note that all the solutions in the sample are better than the optimal TSP tour time for this instance, which is 20460 s. The minimum value found in the sample, 17851 s, is 12.75% smaller than its TSP counterpart, while the maximum one, 19771 s, is 3.37% smaller.

Fig. 5.20: Distribution of the sample data over different binning techniques:

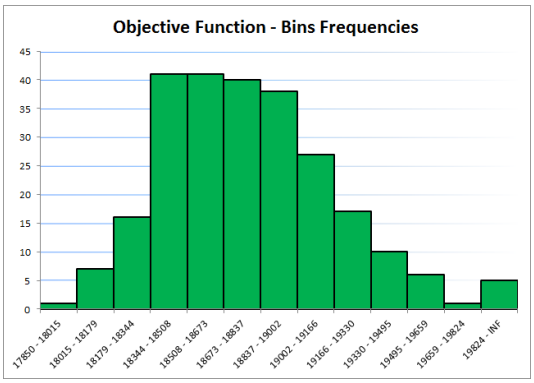


Fig. 5.21: Freedman-Diaconis choice for bins

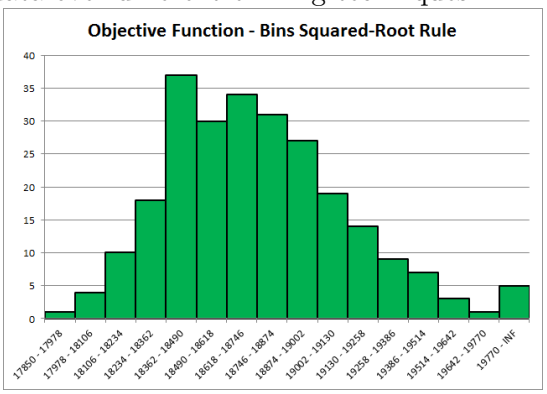


Fig. 5.22: Square-root choice for bins

5.3.6 Record-to-Record Travel

The Simulated Annealing metaheuristic can easily be changed to what is called *Treshold Accepting* or *Record-to-Record Travel* (RRT) [Dueck, 1993, Li et al., 2007] with mainly a change of the acceptance function. The main method would be quite similar to the main method for SA (see Alg. 4.1), but a tentative solution would be accepted only if tentativeSolution $<$ bestSolution$+$threshold%$*$bestSolution which can be easily rewritten as $\frac{\text{tentativeSolution}-\text{bestSolution}}{\text{bestSolution}} <$ threshold%.

One of the implementations of the RRT, the one implemented here, makes the threshold linearly decrease in value until at the very end of the iterations it gets to be 0: currentThreshold $=$ currentThreshold $-\frac{\text{initialThreshold}}{\text{MaxNumberIterations}}$.

Threshold	050.3	100.4	100.3	R
0,120	12635,313	18716,656	17504,418	5,53%
0,110	12639,160	18673,978	17676,336	5,81%
0,100	12513,978	18827,640	17556,923	5,51%
0,090	12725,043	18677,840	17577,485	5,86%
0,080	13155,775	18690,771	17608,155	7,15%
z**	11907,865	17743,975	16669,008	

Fig. 5.23: Calculation and results for the initial threshold value in RRT

By having only one threshold and no lower limit greater than zero to stop at, there is only one parameter to tune, and tuning is done with ten runs of every one of the representative instances, with a similar process to temperature tuning. In fact, Fig 5.23 presents a table very similar to the ones seen for temperature tuning in Sec. 5.2.2, but instead of having three matrices 5×5 here there are three columns 5×1. z^{**} have the lowest objective value per instance found overall between temperature and threshold tuning, so that results are comparable. As a matter of fact, the gaps seen here between z^{**} and the average value per ten runs, are worse by more than 1% in every cell of the matrices. This has mainly to do with not having done any pre-tuning for this problem, and can only yet be partly attributed to the algorithm being worse than SA. The chosen initial value for the threshold will still be the one with smallest R in that table (also seen in the same figure), which is in this case 0.100.

The results of the execution of ten runs for each instance are then shown in Tab. 5.5. In it, the first four columns resemble the ones in the SA tables, whereas the final three are calculated as the ratio between the value for the RRT and the one for the SA (gotten

from Tab. 5.3 and Tab. 5.4).

Tab. 5.5: Results of all the simulations for Record-to-Record Travel

Instance	Obj$^{RRT}\mu$	$T^{RRT}_{toBest}\mu$ (s)	$T^{RRT}_{total}\mu$ (s)	Obj^{RATIO}	T^{RATIO}_{toBest}	T^{RATIO}_{total}
005.1	4464,88863	0,0038	85,521	1,00	0,00	0,31
005.2	3507,06846	10,2804	61,7012	1,00	0,16	0,17
005.3	3275,68949	16,6099	65,9183	1,00	1496,39	3,20
005.4	5312,46792	0,004	170,773	1,00	0,09	2,98
005.5	5510,16502	3,4739	671,5124	1,00	0,21	1,16
006.1	7080,94194	0,3193	18,5943	1,00	41,47	1,01
006.2	6147,95563	0,3881	15,6865	1,00	0,30	0,13
006.3	6835,15844	3,5902	30,9453	1,00	579,06	0,63
006.4	4452,5966	0,0604	662,3267	1,01	0,02	2,05
006.5	5392,07871	0,0185	209,018	1,00	0,00	0,35
007.1	5533,85258	0,309	162,5837	1,00	0,67	4,16
007.2	5342,68241	0,0007	87,0119	1,00	0,00	0,83
007.3	8015,94111	11,318	55,9334	1,03	1,29	1,18
007.4	7610,37708	0,0379	90,232	1,00	0,10	0,84
007.5	7010,98949	37,5402	50,5544	1,00	4,15	1,77
008.1	6709,01894	0,0323	172,8448	1,00	0,03	1,20
008.2	6587,17795	0,2168	112,3159	1,00	2,28	2,33
008.3	5852,48659	64,2363	183,6886	1,01	1,21	2,31
008.4	6914,78538	0,0063	50,4008	1,00	0,06	1,30
008.5	5961,25119	35,6739	79,1291	1,00	4,69	1,18
009.1	7338,77256	0,0739	150,4852	1,00	0,10	3,54
009.2	6204,62976	51,7431	89,5329	1,00	1,30	1,40
009.3	7698,11241	26,8548	34,6861	1,00	1,60	1,09
009.4	6828,5752	0,015	1222,9561	1,00	0,24	17,46
009.5	7802,66509	0,0087	68,3035	1,00	0,00	1,75
010.1	5989,05712	0,1674	51,4537	1,00	0,16	0,70
010.2	6394,38587	14,5645	73,2005	1,00	0,39	0,90
010.3	7073,14733	68,3544	234,87	1,07	1,22	1,75
010.4	8377,91735	1,3925	276,0546	1,00	1,13	1,10
010.5	9183,11056	3,5051	49,3717	1,02	0,78	1,44
050.1	12676,5095	151,3596	224,6151	1,01	2,12	1,05
050.2	12293,3567	121,8595	212,8452	0,99	1,34	1,02
050.3	12513,9777	162,4965	247,0578	0,99	1,58	1,29
050.4	13136,4954	136,4423	182,989	1,02	1,38	0,99
050.5	12330,8456	133,9005	193,6886	1,01	1,46	1,02
100.1	18135,1372	306,2095	323,0247	1,01	1,24	1,21
100.2	17561,3743	301,1968	319,045	1,01	1,23	1,17
100.3	17556,9235	318,2933	339,7488	1,02	1,32	1,28
100.4	18827,64	286,5011	299,1983	1,02	1,15	1,12
100.5	17752,5932	308,1234	318,3444	1,02	1,07	1,02
150.1	22789,6625	430,0093	434,0195	1,00	1,24	1,19
150.2	22573,2789	454,6185	459,2116	1,00	1,22	1,20
150.3	23373,0017	469,7369	473,0695	1,01	1,28	1,24
150.4	22445,3613	434,7686	436,9038	0,99	1,19	1,14
150.5	23354,2395	425,6604	432,8136	1,02	1,16	1,13
200.1	26128,6034	775,5805	776,9429	0,97	1,74	1,70
200.2	28076,4362	824,3249	825,4466	1,01	1,87	1,82
200.3	26755,4297	595,9597	597,4263	0,99	1,21	1,17
200.4	27871,901	506,5174	507,3796	0,98	1,00	0,98
200.5	27519,7059	610,158	610,8755	0,99	1,22	1,19

Apart from a few outliers (Instances 005.3, 006.1, 006.3 and 009.4, which are cut off from the next graphs) all the values stay between very well defined and reasonable bounds.

The objective ratios can be seen in Fig. 5.24 where it can be seen that results for the smaller instances are mostly clustered around the unity, which means that the solutions are the exact same or very close-by. For the 50 and 150 customer instances results are again around one, but with more variance. For 100 customers instances all results are slightly above, which means that the RRT results are worse than the SA, whereas for 200 customer instances results are mostly under one, which means that the RRT outperforms the SA.

As can be seen in Fig. 5.26, however, computation times are vastly superior for the RRT on the bigger instances, even up to and more than 1.5x. Conversely, time ratios for

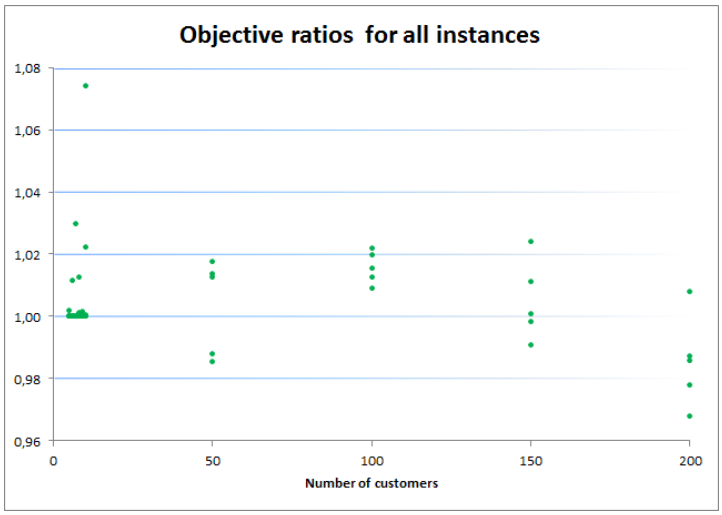


Fig. 5.24: Ratios of average RRT and average SA objective

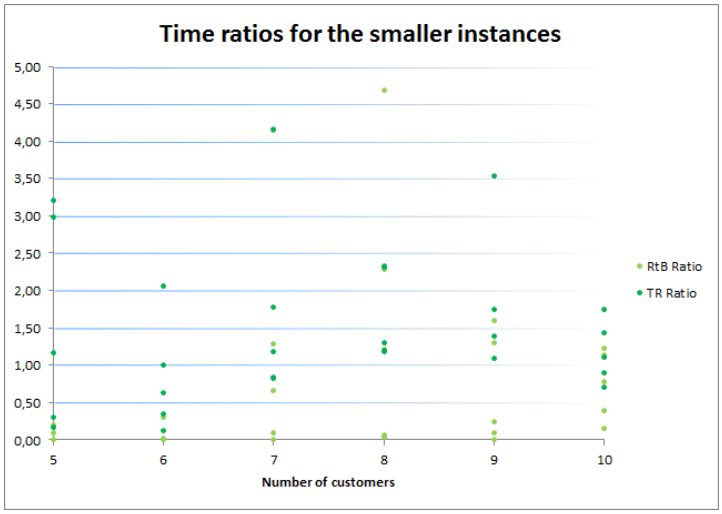


Fig. 5.25: Time ratios (RRT/SA) for the smaller instances

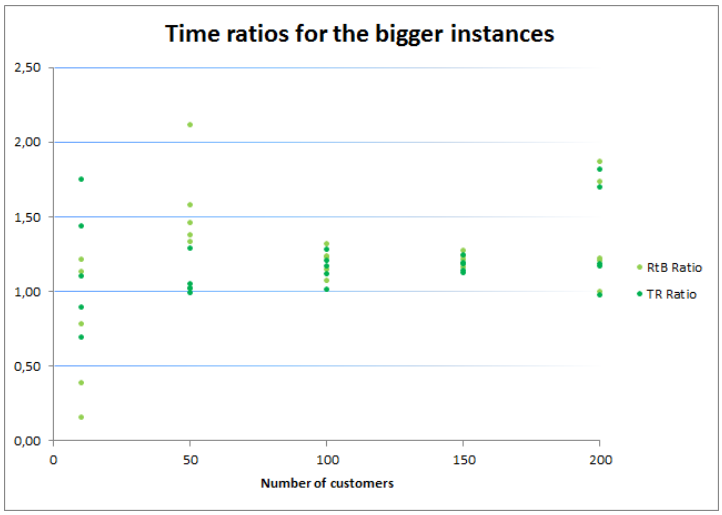


Fig. 5.26: Time ratios (RRT/SA) for the bigger instances

the smaller instances show mostly to be much more in favour of the RRT, at least until 8 customers per instance. Then the situation returns to show that RRT is definitely slower than SA.

All in all, considering that much less time was spent in tuning and working with RRT, results are not to be overlooked as with a longer computation time comes also a seeming improvement in the solution quality output.

Conclusions

In this thesis, Simulated Annealing was shown to be a good way to find solutions for the FSTSP such that it could be practical to use in real-world applications. Moreover, the possible savings of using a truck and drone combination are once again shown among the computational results. Drones seem to be shaping up as an important way to perfrom last-mile delivery in an efficient way.

Chapter 1 briefly introduced the problem to the reader, such that the real-world ties are clear. Chapter 2 explores the existing literature on drone usage in the delivery business, but also with focus on sureillance and military operations. Chapter 3 completely defines the problem with a mathematical model heavily based on the FSTSP by Murray and Chu [Murray and Chu, 2015]. In Chapter 4 different possible solutions to the problem are discussed and this can only lead to using the Simulated Annealing metaheuristic to solve the problem. It is shown thoroughly how it is implemented in its only move and how evaluation of a solution is performed, case by case. Finally, after a complete tuning for the parameters of the SA, Chapter 5 shows numerical analyses on it.

There, it is shown with 6 experiences that the use of drones permits to save a considerable amount of time in touring a set of nodes, and that they should be considered as a feasible way to improve and streamline parcel delivery. The SA is shown to be finding good solutions for the instances provided in linear time. Changes in the operating conditions are shown to be somewhat related in a meaningful way to the objective function, such that if drone endurance or drone top speed or the number of iterations the SA is let run increase, then the time it takes to complete the tour decreases. The objective function is also shown to be somewhat normally distributed by means of descriptive statistics. Finally a comparison with Record-to-Record Travel highlights the pros and cons of using the one or the other method.

First and foremost, future research should finish developing the three moves not implemented in this thesis, and possibly could explore the other approaches described together with the SA for the solution of the problem. It would be feasible also to shave some time off of the current implementation to be able to increase the maximum iterations and give a better chance to the SA to explore a neighborhood. An analysis on possible correlation between the optimal TSP tour for any instance and the optimal truck tour in the FSTSP is surely to be beneficial and insightful on how to build better heuristics for finding better solutions to the problem. A lower bound for the FSTSP could also be an interesting topic of research, as [Agatz et al., 2015] have found one for their TSP-D but it doesn't seem to be directly applicable to the FSTSP since TSP-D has the drone tied to the road network exactly to be able to have that lower bound defined. The important branch of research that is the VRP area of this problem should be explored as it would lead to even more practical and real-world use cases.

While future research on the technical side of it is still underway, from the technological point of view many innovations are quickly coming to the market for drones. Hydrogen fuel cells seems to be a promising way to be able to safely and steadily increase the endurance of a drone from the current standards of 20-30 min to over two hours [Popper, 2015a], greatly improving the already good case in favour of drones. Moreover, refueling would be even faster, and even if in the model it is counted as an independent task (batteries are thought to be available in a steady supply) it would improve things even further. Advancements in obstacle detection and avoidance technology will make it also possible for drone to fly completely autonomously very soon [Nicas and Bensinger, 2015].

All in all, the future seems to be shaping up so that drone deliveries will have a very bright one.

Bibliography

[Onl, 2008] (2008). The future of CEP services.

[Onl, 2012] (2012). Online Retail Sales hit £50bn.

[Giz, 2015] (2015). Amazon to begin testing new delivery drones in the US.

[FPS, 2015] (2015). FPS Takes Flight with First Commercial UK Drone Delivery.

[Hor, 2015] (2015). HorseFly UAV Specifications.

[Cap, 2015] (2015). Volare alto con i droni.

[Agatz et al., 2015] Agatz, N., Bouman, P., and Schmidt, M. (2015). Optimization Approaches for the Traveling Salesman Problem with Drone. Technical Report ERS-2015-011-LIS, Erasmus Research Institute of Management (ERIM).

[Applegate et al., 2006] Applegate, D., Bixby, R., Chvatal, V., and Cook, W. (2006). Concorde TSP solver.

[Applegate et al., 2007] Applegate, D., Bixby, R. E., Chvátal, V., and Cook, W. J. (2007). Concorde TSP solver - Home Page.

[Applegate et al., 2011] Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2011). *The Traveling Salesman Problem: a Computational Study*. Princeton University Press.

[Ball, 2013] Ball, J. (2013). Amazon to deliver by drone? Don't believe the hype.

[Bermingham, 2014] Bermingham, F. (2014). FedEx Researching Drone Delivery But Not For Widespread Use.

[Boone et al., 2015] Boone, N., Sathyan, A., and Cohen, K. (2015). Enhanced Approaches to Solving the Multiple Traveling Salesman Problem. *AIAA Infotech at Aerospace*, SciTech.

[Bortoff, 2000] Bortoff, S. A. (2000). Path Planning for UAVs. *Proceedings of the American Control Conference*, 1(6):364–368 vol.1.

[Busetti, 2003] Busetti, F. (2003). Simulated Annealing Overview. *JP Morgan, Italy*.

[Capt. Tice, 1991] Capt. Tice, B. P. (1991). Unmanned Aerial Vehicles: The Force Multiplier of the 1990s. *Airpower Journal*, V(1).

[Černỳ, 1985] Černỳ, V. (1985). Thermodynamical Approach to the Traveling Salesman Problem: an Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51.

[Cobweb Information Ltd, 2013] Cobweb Information Ltd, . (2013). Uk market synopsis - courier and parcel services.

[Dorigo et al., 2006] Dorigo, M., Birattari, M., and Stützle, T. (2006). Ant Colony Optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

[Drexl, 2012] Drexl, M. (2012). Synchronization in Vehicle Routing – A Survey of VRPs with Multiple Synchronization Constraints. *Transportation Science, Transport Science*, 46(3):297–316.

[Dueck, 1993] Dueck, G. (1993). New Optimization Heuristics - The Great Deluge Algorithm and the Record-to-Record Travel. *Journal of Computational Physics*, 104(1):86–92.

[Duncan, 2015a] Duncan, J. S. (2015a). Exemption No. 11290, Regulatory Docket No. FAA–2014–0474.

[Duncan, 2015b] Duncan, J. S. (2015b). Exemption No. 13564, Regulatory Docket No. FAA–2015–3055.

[Eppstein, 1998] Eppstein, D. (1998). Finding the k Shortest Paths. *Siam Journal on Computing, Siam J. Comput, Siam J Comp, Siam J Comput, Siam J. Computing*, 28(2):652–673.

[Evans, 2015] Evans, O. (2015). Matternet, named Technology Pioneer 2015 by the World Economic Forum, appoints ex-Chief Cargo Officer of Swiss International Air Lines as new Head of Global Business Development.

[French, 2015a] French, S. (2015a). Drone delivery is already here — and it works.

[French, 2015b] French, S. (2015b). The Apple Store now sells drones.

[Ha et al., 2015] Ha, Q. M., Deville, Y., Pham, Q., and Hà, M. H. (2015). Heuristic methods for the Traveling Salesman Problem with Drone. *CoRR*, abs/1509.08764.

[Hern, 2015] Hern, A. (2015). DHL launches first commercial drone 'parcelcopter' delivery service.

[Johnson and Jacobson, 2002] Johnson, A. and Jacobson, S. (2002). On the Convergence of Generalized Hill Climbing Algorithms. *Discrete Applied Mathematics*, 119(Issues 1–2):37 – 57. Special Issue devoted to Foundation of Heuristics in Combinatoria l Optimization.

[Jünger et al., 1995] Jünger, M., Reinelt, G., and Rinaldi, G. (1995). Chapter 4 The traveling salesman problem. *Handbooks in Operations Research and Management Science*, 7:225–330.

[Karp, 1972] Karp, R. M. (1972). *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, chapter Reducibility among Combinatorial Problems, pages 85–103. Springer US, Boston, MA.

[Kashuba et al., 2015] Kashuba, S. V., Lysenko, O. I., Novikov, V. I., and Alekseeva, I. V. (2015). Optimization of UAV Path for Wireless Sensor Network Data Gathering. *Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD), 2015 IEEE International Conference*, pages 280–283.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D. J., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[Koebler, 2013] Koebler, J. (2013). The Next Gun Debate? Armed Drones Could Be Protected By the Second Amendment.

[Li et al., 2007] Li, F., Golden, B., and Wasil, E. (2007). A Record-to-Record Travel Algorithm for solving the Heterogeneous Fleet Vehicle Routing Problem. *Computers and Operations Research*, 34(9):2734–2742.

[Lin, 2011] Lin, K. Y. C. (2011). A vehicle routing problem with pickup and delivery time windows, and coordination of transportable resources. *Computers and Operations Research*, 38(11):1596–1609.

[Lipowski and Lipowska, 2012] Lipowski, A. and Lipowska, D. (2012). Roulette-Wheel Selection via Stochastic Acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193 – 2196.

[Little, 2013] Little, K. (2013). Did Amazon just pull off the best PR stunt ever?

[Madrigal, 2014] Madrigal, A. C. (2014). Inside Google's Secret Drone-Delivery Program.

[Mathew et al., 2015] Mathew, N., Smith, S. L., and Waslander, S. L. (2015). Planning Paths for Package Delivery in Heterogeneous Multirobot Teams. *IEEE Transactions on Automation Science and Engineering*, 12(4):1298–1308.

[Murray and Chu, 2015] Murray, C. C. and Chu, A. G. (2015). The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109.

[Nicas and Bensinger, 2015] Nicas, J. and Bensinger, G. (2015). Delivery Drones Hit Bumps on Path to Doorstep.

[Nilsson, 2003] Nilsson, C. (2003). Heuristics for the traveling salesman problem. Technical report, Tech. Report, Linköping University, Sweden.

[Popper, 2015a] Popper, B. (2015a). Hydrogen fuel cells promise to keep drones flying for hours.

[Popper, 2015b] Popper, B. (2015b). The teenager behind the drone gun now has a drone-mounted flamethrower.

[Quaritsch et al., 2010] Quaritsch, M., Kruggl, K., Wischounig-Strucl, D., Bhattacharya, S., Shah, M., and Rinner, B. (2010). Networked UAVs as Aerial Sensor Network for Disaster Management Applications. *Elektrotechnik Und Informationstechnik, Elektr. Inf. Tech*, 127(3):56–63.

[Richards et al., 2002] Richards, A., Bellingham, J., Tillerson, M., and How, J. (2002). Coordination and Control of Multiple UAVs. In *AIAA guidance, navigation, and control conference, Monterey, CA*. American Institute of Aeronautics and Astronautics.

[Rosenkrantz et al., 2009] Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M. (2009). An analysis of several heuristics for the traveling salesman problem. *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz, Fundamental Probl. in Comp.: Essays in Honor of Prof. Daniel J. Rosenkrantz*, pages 45–69.

[Savuran and Karakaya, 2015] Savuran, H. and Karakaya, M. (2015). Route Optimization Method for Unmanned Air Vehicle Launched from a Carrier. *Lecture Notes on Software Engineering*, 3(4):279.

[Shang et al., 2014] Shang, B., Wu, C., Hu, Y., and Yang, J. (2014). An Algorithm of Visual Reconnaissance Path Planning for UAVs in Complex Spaces. *Journal of Computational Information Systems*, 8(1):1–8.

[Smiley, 2015] Smiley, L. (2015). These drones are already atarting to deliver medicine, books, and pizza.

[Toth and Vigo, 2014] Toth, P. and Vigo, D. (2014). *Vehicle routing: problems, methods, and applications*, volume 18. Siam, second edition.

[Tremonti, 2015] Tremonti, A. M. (2015). Curbing Voyeurism: Peeping Toms go high tech but not undetected.

[Union, 2014] Union, U. P. (2014). *Development Strategies for the Postal Sector: an Economic Perspective*. Eburon Academic.

[Zheng et al., 2005] Zheng, C., Li, L., Xu, F., Sun, F., and Ding, M. (2005). Evolutionary Route Planner for Unmanned Air Vehicles. *IEEE Transactions on Robotics*, 21(4):609–620.

[Zito, 2016] Zito, D. (2016). First Test Delivery of the Workhorse Group HorseFly Drone Featured on NBC-TV Affiliate.