ABSTRACT

Title of dissertation: HYBRID ROUTING MODELS UTILIZING TRUCKS OR SHIPS TO LAUNCH DRONES

Stefan Poikonen
Doctor of Philosophy, 2018

Dissertation directed by: Professor Bruce Golden
R.H. Smith School of Business

Technological advances for unmanned aerial vehicles, commonly referred to as drones, have opened the door to a number of new and interesting applications in areas including military, healthcare, communications, cinematography, emergency response, and logistics. However, limitations due to battery capacity, maximum take-off weight, finite range of wireless communications, and legal regulations have restricted the effective operational range of drones in many practical applications.

Several hybrid operational models involving one or more drones launching from a larger vehicle, which may be a ship, truck, or airplane, have emerged to help mitigate these range limitations. In particular, the drones utilize the larger vehicle as both a mobile depot and a recharging or refueling platform. In this dissertation, we describe routing models that leverage the tandem of one or more drones with a larger vehicle. In these models, there is generally a set of targets that should be visited in an efficient (usually time-minimizing) manner. By using multiple vehicles, these targets may be visited in parallel thereby reducing the total time to visit all targets.

The vehicle routing problem with drones (VRPD) and traveling salesman problem with a drone (TSP-D) consider hybrid truck-and-drone models of delivery, where the goal is to minimize the time required to deliver a set of packages to their respective customers and return the truck(s) and drone(s) to the origin depot. In both problems, the drone can carry one homogeneous package at a time. Theoretical analysis, exact solution methods, heuristic solution methods, and computational results are presented. In the mothership and drone routing problem (MDRP), we consider the case where the larger launch vehicle is free to move in Euclidean space (the open seas) and launch a drone to visit one target location at a time, before returning to the ship to pick up new cargo or refuel. The mothership and high capacity drone routing problem (MDRP-HC) is a generalization of the mothership and drone routing problem, which allows the drone to visit multiple targets consecutively before returning to the ship. MDRP and MDRP-HC contain elements of both combinatorial optimization and continuous optimization. In the multi-visit drone routing problem (MVDRP), a drone can visit multiple targets consecutively before returning to the truck, subject to energy constraints that take into account the weight of packages carried by the drone.

HYBRID ROUTING MODELS UTILIZING TRUCKS OR SHIPS TO LAUNCH DRONES

by

Stefan Poikonen

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Professor Bruce Golden, Chair/Advisor
Professor Edward Wasil
Professor Subramanian Raghavan
Professor Denny Gulick
Professor Tunay Tunca

Dedication

To my family.

Acknowledgments

I owe immeasurable thanks to my advisor, Professor Bruce Golden. His guidance has spanned from granular details of papers to strategic and life advice. He helped me to see the light at the end of the long graduate school tunnel when I was still finding myself as a student. It seems just yesterday I was in BMGT831. I hope for many more papers together and four mile walks.

I also owe special thanks to Professor Edward Wasil who has worked closely with me on multiple projects. His careful eye and advice have been invaluable, and I deeply appreciate all he has done along the way. I hope to continue collaborating over the coming years.

To Rui Zhang, thank you for your sound advice, especially during the job application process. You gave me a more concrete sense of what to expect in the interviewing process, which was extremely helpful. We have worked together on one project so far, but I hope our future proximity in Colorado will lead to others.

To Xingyin Wang, I appreciate the time we had to work on a couple projects together and the chances we have had to meet at conferences. Hopefully we can have a chance to meet and collaborate in the future.

To other friends in the Saturday research group who have been supportive in a number of ways, thank you. I especially wish to mention Debdatta Sinha Roy, who has been a supportive friend and classmate.

To all of you who have helped make this possible through your love, support, and encouragement, thank you. I wish to especially mention my family who have consistently supported me in a variety of ways.

The work of Chapter 2 in this dissertation is derived from joint work with Xingyin Wang and Bruce Golden, some of which appeared in the journal *Optimization Letters* and some of which appeared in the journal *Networks*. The work of Chapter 3 in this dissertation is derived from joint work with Bruce Golden and Edward Wasil, which will appear in the *INFORMS Journal on Computing*. The work of Chapters 4 and 5 in this dissertation are derived from joint work with Bruce Golden.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CETSP	Close-Enough Traveling Salesman Problem
CEVRP	Close-Enough Vehicle Routing Problem
FSTSP	Flying Sidekick Traveling Salesman Problem
GLONASS	Global Navigation Satellite System
GPS	Global Positioning System
MDRP	Mothership and Drone Routing Problem
MDRP-HC	Mothership and High Capacity Drone Routing Problem
MVDRP	Multi-visit Drone Routing Problem
TSP	Traveling Salesman Problem
TSP-D	Traveling Salesman Problem with a Drone
UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aerial System
VRP	Vehicle Routing Problem
VRPD	Vehicle Routing Problem with Drones

Chapter 1: Introduction

1.1 Background on Drones

Unmanned aerial vehicles (UAVs), commonly referred to as drones, come in a variety of shapes and sizes to fit a myriad of applications. Drones gained public notoriety for their use in military contexts, particularly by the United States in Afghanistan and later in Iraq, Libya, Yemen, and Somalia [3, 64].

In recent years, the suggested and actual uses of drones in non-military contexts have rapidly expanded. A study conducted by the Association of Unmanned Vehicle Systems International estimated that drones and related systems will have an economic impact in the United States totaling $82.1 billion from 2015 until 2025 [33]. Business Insider's Intelligence Unit projects the sales of drones to surpass $12 billion in year 2021 alone [45].

A number of companies have capitalized on the cinematographic capabilities of drones. These companies market their drones both to professional filmmakers and hobbyists. DJI, a company based in Shenzhen, China, is the largest consumer drone manufacturer in the world. In 2017, DJI projected sales of $2.7 billion, with 80% of its profits attributable to consumer drone sales [16]. In March 2018, PCMag.com rated the top consumer drones of 2018, where DJI took eight of the top 10 spots,

Figure 1.1: The DJI Phantom 4 is pictured above. The Phantom 4's user manual states the drone is capable of flying up to 28 minutes, using GPS or GLONASS satellite systems, filming at 4K ultra high definition resolution, taking still frame images up to 12 megapixels, traveling at a maximum speed of 20 meters per second, and fixing its focus on a particular target via the use of a gimbal. The total weight of the drone is 1.38kg. Image was retrieved from `https://store.dji.com/product/phantom-4-beginner-kit` in June 2018.

with best seller DJI Phantom 4 taking the top spot [29]. In Figure 1.1, the DJI Phantom 4 is pictured.

The photographic and video capabilities of drones have applications in other areas as well. In agriculture, drones are used to quickly conduct aerial surveillance of crops. The collected imagery may then undergo a spectral analysis to gauge the development, moisture content, and health of crops, which allows for more precise decision making by farmers, including when to water, fertilize, and harvest crops. Additionally, drones such as the MG-1S, pictured in Figure 1.2, may be used to

Figure 1.2: The MG-1S drone is seen spraying liquid into crop fields. Image retrieved as screen capture from `https://www.youtube.com/watch?v=P2YPG8PO9JU` in June 2018.

spread pesticides in fields. In forestry, drones may be used to monitor the growth of flora. They have also been used to spot illegal deforestation activities [52].

Numerous applications for drones exist in security and public safety [8]. In the state of Arkansas, the Fayetteville City Police Department has trained several officers to fly unarmed drones, to aid in searches for missing people, to track suspects fleeing police, to assist in swift water rescue, and potentially to conduct supply drops during natural disasters [17]. Security at the Coachella Music Festival will be using drones to monitor crowd movements, but also to reduce the risk of a Las Vegas style massacre, as occurred at the Route 91 Harvest Music Festival [28]. The US Customs and Border Patrol is exploring increasing the size of its drone fleet to assist in monitoring the borders of the United States [14].

Facebook has tested solar powered drones over the Arizona desert, with the goal of eventually launching Internet-providing drones across the world to facilitate internet access for over one billion people [46].

There are applications for drones in the healthcare sector. In the country of Rwanda, the road infrastructure is limited outside of major cities. The company Zipline delivers blood bags from a centralized refrigerated blood bank to remote hospitals and transfusion centers. A delivery that once took two hours by car may now only take 20 minutes [38]. Zipline has plans to expand service to Tanzania, making up to 2,000 drone flights per day to more than 1,000 healthcare facilities across the country, and is in talks with hospitals in other countries [38,39]. In Switzerland, hospitals have used the services of the company Matternet to deliver medical supplies between hospitals rapidly. Matternet's drones are capable of carrying four pounds of goods up to 12 miles [39].

The use of drones has been documented post-disaster scenarios. Following the April 2015 earthquake in Nepal that killed more than 7,500 people, drones were deployed to survey damage in remote mountain villages, which aided in prioritizing relief efforts [27]. In North Carolina [25] and Texas [26], drones have been used to help identify people affected by flash floods and direct emergency response to them. Adams and Friedland [1] provide a survey of imagery collection via drone in disaster scenarios.

Google's Project Wing [66], the Amazon Prime Air program [6], DHL [22], DPD [23], UPS [61], the Finnish Postal Service [55], and the Russian Postal Service [57] have all considered using drones for parcel delivery. Amazon's efforts have received special attention in the academic literature, following a 2013 television interview of Amazon's CEO Jeff Bezos. During this interview, he stated that half-hour delivery was possible via drone for packages up to five pounds, which represents 86% percent of the company's deliveries [12]. Moreover, the market for rapid delivery of online orders is growing. In a press release, Amazon stated that over five billion items were shipped in 2017 via Amazon Prime, a premium service that offers free two-day shipping on more than 100 million items in the US [5].

The potential applications of drones are vast. Although the benefits of using drones vary depending on the operational context, there are a number of advantages that are frequently seen in drone use cases. In virtually all examples of commercial drone use that we have found, at least some subset of the following advantages apply.

1. Unique line-of-sight capabilities from the sky.
2. Motion not constrained by street networks or street traffic.
3. Cheaper to manufacture relative to traditional ground-based transport.
4. Higher maximum speeds.
5. Energy efficient relative to alternatives.
6. Quieter than combustion engines.
7. Reduces traffic congestion on streets.
8. Operator not required for autonomous drones.
9. Avoids other ground-based dangers or disruptions.

Despite all of these potential advantages, drones present a new array of safety, regulatory, and operational challenges.

In the United States, drone operators must maintain a visual line-of-sight with the unmanned aerial vehicle [40]. Even if visual line-of-sight regulations were lifted, the range of a drone (as constrained by battery life) is finite. A video posted on Amazon's official YouTube channel claims an effective drone range of 15 miles [7]. The FAQ page for Amazon's Prime Air program continues to point to drones capable of delivering packages up to five pounds [6]. However, certain locations may be full of landing obstructions and may not be suited for drone delivery. These limitations on drone use must be considered in most practical operational models.

1.2 Academic Literature Review

1.2.1 Drone Routing

There is a significant body of literature related to micro-level autonomous decision making by drones with respect to optimal control, collision avoidance, obstacle detection, and path finding. Albaker and Rahim [4] survey collision avoidance techniques. Goerzen et al. [34] provide an excellent survey of path finding algorithms. Mori and Scherer [48] and Gageik et al. [30] discuss image processing aspects of obstacle detection by a drone. Though an interesting field, the focus of this dissertation is not on micro-level decision making.

There is an emerging operations research literature related to the use of drones, which considers higher level objectives. These objectives may seek to optimize the number of drones to use, the order of visiting some set of targets, choosing a set of customers to be delivered by drone, etc. Otto et al. [51] provide a detailed survey of papers that consider optimization questions for non-military drone operations. They document consistent growth of the research field, with nearly eight times as many academic manuscripts published in this area in 2017 than 2012.

Several papers, including Avellar et al. [9], Barrientos et al. [10], and Nedjati et al. [50] consider area coverage problems. In area coverage problems, the drone has a sensor (e.g., a camera) with a finite effective range. The drone must travel a path such that the sensor is able to collect a signal from all requisite areas. These problems frequently seek to minimize either energy expenditure, drone flight time, or the number of drones required, and have application to security patrols, agriculture, mapping, and post-disaster assessment.

Another common task for drones is related to search operations. In papers by Raap et al. [56] and Lin and Goodrich [41], the authors seek to either minimize the expected amount of time required for the drone to detect the search object or maximize the probability of detection, given fixed drone flight time. The search object may be stationary or dynamic. If stationary, there may exist a prior probability distribution of the location of the search object.

Several papers consider a fixed set of targets that must be visited in some order by a vehicle or fleet of vehicles in a cost- or time-minimizing manner and may take into account special physical constraints. The work of Dubins [24] from 1957 has given rise to the Dubins Traveling Salesman Problem, where the path of a vehicle is constrained by a minimum turn radius. More recently, a number of papers have considered similar constraints with respect to drones. Manyam et al. [44] use certain motion constraints, including a minimum turn radius, and consider a multi-drone, multi-depot optimization problem. Babel [11] considers a traveling salesman variant with curvature constraints and obstacles. If we do not account for special physical constraints of the drone, then standard vehicle routing and traveling salesman solution techniques may be used. The edited volumes by Golden, Raghavan, and Wasil [35] and Toth and Vigo [60] explore these techniques extensively.

1.2.2 Hybrid Truck-and-Drone Models

The finite battery life of drones along with the ability to lift only relatively small payloads has led to the development of hybrid truck-and-drone models of delivery. The broad idea is that trucks may bring drones close enough to target locations, where range concerns of the drone are alleviated. The drone can launch, visit some set of targets, and return to the truck for recharging or a battery swap. The truck may carry packages that the drone is not actively delivering to targets. Thus, the trucks may be viewed as mobile depots and recharging platforms. These problems inherently involve some form of synchronization constraints between trucks and drones.

The first paper published in the literature concerning hybrid truck-and-drone models of delivery was the Flying Sidekick Traveling Salesman Problem (FSTSP) by Murray and Chu [49]. In the FSTSP, there is one truck, one drone, and a set of customers C. Each customer $c \in C$ has a demand of one homogeneous package. The package may be delivered by a driver-operated truck or by a drone. Some packages may be unsuitable for drone delivery (e.g. they may be too heavy), and thus must be delivered by the truck. The drone is assumed to have a battery that lasts for a fixed duration. The drone may be launched only at the depot or at customer delivery locations and may only carry one package at a time. While the drone is airborne, the truck may visit multiple customers. The objective is to deliver all packages and return the truck and drone to the origin depot in the minimum amount of time. Murray and Chu formulated a mixed integer linear program for the FSTSP, but it could not solve instances with even ten customer package locations in a reasonable amount of time. This motivated a fast heuristic method. The authors developed a heuristic that generates a truck-only delivery path by solving a standard traveling salesman problem (TSP). Then individual packages are reassigned in an iterative, greedy fashion that maximizes time savings. By reassigned, we mean a package may be swapped from truck delivery to drone delivery or vice versa, or the package may be delivered to a new customer location in the route's delivery sequence.

Agatz et al. [2] study the traveling salesman problem with a drone (TSP-D). They formulate the TSP-D as a mixed integer linear program and then develop a family of heuristics, which may be described as "route first, partition second". First, a truck-only TSP solution is formed, either via exact methods or using a minimum spanning tree heuristic. The route is partitioned into customer locations that are delivered by truck and customer locations that are delivered by drone. The route is partitioned using a heuristic method and an exact method based on dynamic programming. Additional details of the work by Agatz et al. will be presented in

Chapter 3.

Campbell et al. [19] use continuous approximation of the transportation network to estimate expected delivery costs for various customer densities and relative operating costs of trucks and drones. A key insight is that maximal savings for a truck-and-drone model of delivery may be found in areas of intermediate customer density, consistent with suburban areas.

Ha et al. [37] study a problem they termed the traveling salesman problem with a drone (TSP-D) with a different objective function than [2]. In their problem, the objective is to minimize the sum of transportation costs and waiting costs for the truck and drone. The authors use the greedy randomized adaptive search procedure (GRASP) to generate solutions.

1.3 Main Contributions

This dissertation seeks to explore emerging operational models dealing with the synchronization of drones with other vehicles, including trucks and ships.

In Chapter 2, we introduce the Vehicle Routing Problem with Drones (VRPD). This model considers a hybrid routing model with multiple trucks and multiple drones per truck. Chapter 2 focuses on theoretical analysis and establishes maximum speed-up ratios by using this model relative to traditional truck-only delivery. It also establishes a relationship between the VRPD and two established problems in the literature: the close-enough vehicle routing problem and the min-max vehicle routing problem.

In Chapter 3, we consider computational approaches to the TSP-D model. In particular, we use a branch-and-bound based approach and were able to find optimal solutions to all 30 instances of Agatz et al. [2]. Heuristic approaches to generate solutions for large instances are also described and implemented.

Chapters 4 and 5 focus on a family of problems that we have name the Mothership and Drone Routing Problems. Unlike other papers in the literature, the launch vehicle in these problems (which may be a naval ship, airplane, or airship) is assumed to move in continuous (Euclidean) space, rather than along a (street) network. We find that second order cone programming is a helpful embedded procedure for determining optimal launch and landing locations for the drone.

Chapter 6 considers a truck-and-drone model where the drone is free to visit multiple customer locations consecutively. The battery life of the drone depends on the collective weight of packages being carried by the drone at a given time. We also decouple the set of feasible launch/landing locations from the set of customer locations. This new model, which we call the multi-visit drone routing problem, provides additional flexibility compared to the TSP-D model.

Chapter 2: The Vehicle Routing Problem with Drones

2.1 Problem Definition

The $VRPD$ model assumes the following:

- m is the number of homogeneous trucks in the fleet.
- k is the number of drones on each truck.
- α is the ratio of drone speed to truck speed. (Without loss of generality, in this paper, we assume drone speed is α and truck speed is 1.)
- The recharge (or battery swap) of a drone's battery is instantaneous.
- We assume (until explicitly noted otherwise) that drones may only launch from or land on the truck, when the truck is located at a customer delivery location or the depot.
- A drone must land on the same truck from which it launched.

In [15], we proved a number of worst-case results comparing the optimal completion time using a fleet of trucks equipped with drones to the optimal completion time using a traditional fleet of only trucks. The results are summarized in Table 2.1. We refer readers who are interested in the proofs to [15]. Denote, by P_t, the

Table 2.1: Some of the problems studied

	P_t	P_{td}	$\sup\{Z(P_t)/Z(P_{td})\}$
1	TSP	$VRPD_{1,\alpha,k}$	$\alpha k + 1$
2	TSP	$VRPD_{m,\alpha,k}$	$m(\alpha k + 1)$
3	VRP^*	$VRPD_{m,\alpha,k}$	$\alpha k + 1$
4	$VRPD_{m,\alpha,k}$	$VRPD_{m,\beta,k}$	β/α

routing problem with the fleet of trucks only, and, by P_{td}, the problem with the fleet of trucks and drones. $Z(P_t)$ and $Z(P_{td})$ are optimal solutions, i.e., the completion times, to P_t and P_{td}, respectively. We found tight upper bounds on the ratios $Z(P_t)/Z(P_{td})$, which indicated the maximum benefit obtained from incorporating drones into the fleet.

In row 1 of Table 2.1, we compare the TSP to $VRPD_{1,\alpha,k}$, i.e., we have a fleet of only one truck carrying k drones. The worst-case ratio is $\alpha k + 1$. The maximum benefit from using drones depends on the number of drones and the drone speed. If the truck carries 2 drones and the drones travel 50% times faster than the truck, the completion may be reduced by 75%, in the best case.

In row 2, we compare the TSP to $VRPD_{m,\alpha,k}$, i.e., we have a fleet of m trucks each carrying k drones. The maximum amount saved depends on the number of trucks, the number of drones, and the speed of the drones.

In row 3, we compare the VRP^* with a fleet of m trucks to $VRPD_{m,\alpha,k}$. Both the VRP^* and $VRPD_{m,\alpha,k}$ have m trucks in the fleet and the worst-case ratio is $\alpha k + 1$, the same as the ratio when we compared the TSP to $VRPD_{1,\alpha,k}$.

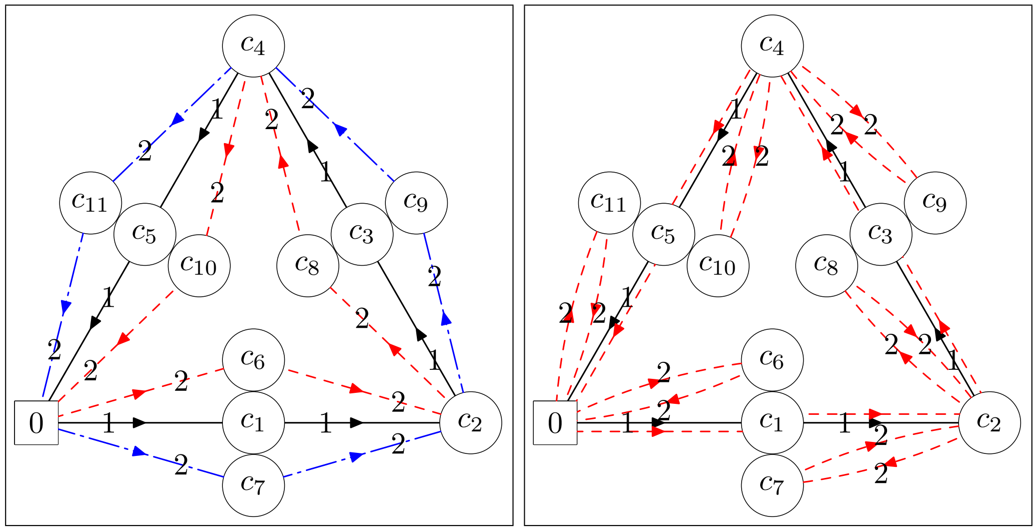
An interesting observation is that the speed of drones, α, and the number of drones per truck, k, play the same role in the worst-case bound. If we have more resources, do we invest in faster drones or in carrying more drones on a truck? In terms of the maximum benefit, doubling the drone speed and doubling the number of drones per truck can produce the same effect, but in a typical case, the problem is not straightforward. A larger number of drones has the advantage of serving more customers in parallel; greater drone speed has the advantage of serving more customers in serial. In our toy examples, we found that if there are times when not all drones are in service (service not fully parallelized), greater drone speed dominates. On the other hand, if drone range or capacity is severely limited, a larger number of drones may dominate. It would be interesting to explore the phenomenon in a simulation study given a computational procedure for the $VRPD$.

It is easy to design instances where a single fast drone is more beneficial than two slow drones. In a trivial case, we can have a single depot and a single package to be delivered to a location d units of distance from the depot. Assume the truck and drone are operating on the same metric. We have the choice of two drones with speed $\alpha_1 = 2$ or one drone with speed $\alpha_2 = 4$. In both cases, the optimal solution is a trivial out-and-back route, launching a single drone directly from the depot. However, in the first case the optimal route duration is $(d+d)/\alpha_1 = d$, whereas the second case has an optimal route duration of $(d+d)/\alpha_2 = \frac{1}{2}d$.

In Figure 2.1, we show an example where two slow drones are more efficient than one fast drone. There are eleven customers. The distances between two nodes (customer or depot) are labeled on the arcs connecting them in Figure 2.1(a). If there is no arc between the two nodes, the distance is the length of the (undirected) shortest path between them. For example, the distance between C_1 and C_6 is the sum of distances between C_1 & C_2 and C_2 & C_6, and thus equals $1 + 2 = 3$. In Figure 2.1(a), we show the optimal solution with slower drones. The fleet has one truck with speed 1 and two drones with speed 2. The solid black line represents the truck path and the red and blue lines represent the two drone paths, respectively. The drones are dispatched at the depot to serve customers C_6 and C_7, respectively, while the truck is dispatched to serve customer C_1, and then C_2. The three vehicles arrive at C_2 at the same time. Then the two drones are sent immediately to serve customers C_8 and C_9. The truck continues to serve C_3, and then C_4. The truck and drones resynchronize at C_4. The drones redeploy to C_{10} and C_{11}, while the truck delivers to C_5. All vehicles regather at the depot. The objective function value of the solution is 6.

In Figure 2.1(b), we show the best solution over the same network with a single faster drone. The fleet has one truck carrying one drone with speed 4 whose path is in red. The drone is dispatched from the depot to serve customer C_6, while the truck is dispatched to serve customer C_1. The truck waits at C_1 for 0.25 time units to pick up the drone, which is immediately sent to serve customer C_7. The truck continues to serve C_2, where it waits for another 0.25 time units to pick up the drone, and so on. The pattern continues, where the truck will eventually serve C_3, C_4, and C_5, waiting at each of those stops for 0.25 time units for the drone to pick up its next package. It can be calculated that the objective function value of the solution is 7.5, which is worse than 6.

In this example, two slower drones are more efficient than one drone that is twice as fast. The limited carrying capacity of a single drone (i.e., one package)



(a) Two slow drones of speed 2 (b) One faster drone of speed 4

Figure 2.1: A larger number of slower drones is better for this network.

forces the single fast drone to resynchronize with the truck on six different occasions (namely C_1, C_2, C_3, C_4, C_5, and the depot), whereas two slower drones only require three resynchronization points at C_2, C_4, and finally back at the depot.

In row 4, we compare two $VRPD$s: $VRPD_{m,\alpha,k}$ and $VRPD_{m,\beta,k}$. The two problems have the same number of trucks each carrying the same number of drones, but the speeds of the drones are different. If we assume $\alpha < \beta$, the the worst-case ratio indicates the maximum savings if a new generation of faster drones is used.

2.2 Extensions: Cost Issues, Other Metrics, and Limited Battery Life

In the previous paper, we ignored cost, assumed that the truck and the drone follow the same distance metric, and ignored the limited battery life of a drone. In this section, we begin to relax these simplifications and provide some initial results for others to build upon.

2.2.1 Limited Battery and Maximum Savings

The following theorem takes into account explicitly the limited battery life (in time units), U, of a drone, which we did not consider in detail in the previous paper. A lower bound on $Z(VRPD_{1,\alpha,k})$ is given by Theorem 1.

Theorem 1. *If the triangle inequality is valid, then*

$$Z(VRPD_{1,\alpha,k}) \geq Z(TSP) - nU\alpha, \tag{2.1}$$

where n is the number of customers served by drones in the optimal $VRPD_{1,\alpha,k}$ solution and U is the battery life of a drone.

Proof of Theorem 1. We construct a feasible TSP solution from the optimal $VRPD_{1,\alpha,k}$ solution. We insert the customers served by drones one by one onto the truck route whose duration was initially equal to $Z(VRPD_{1,\alpha,k})$. Denote the distance between customers i and j by L_{ij}. If a drone is launched at node i to service customer k and is then picked up at node j, the distance covered by the drone is $L_{ik} + L_{kj} \leq \alpha U$. (We assume the truck speed is 1 and the drone speed is α.) If $L_{ik} \leq L_{kj}$, we insert k just after node i on the truck route. If $L_{ik} > L_{kj}$, we insert k just after node j on the truck route. The increase in the distance of the truck route is no more than αU, if the triangle inequality is valid. After all n customers served by the drone are added, the increase in distance (and duration) of the truck route is no more than $n\alpha U$, i.e., the duration of the feasible TSP solution $Z^f(TSP) \leq Z(VRPD_{1,\alpha,k}) + n\alpha U$. Since

$Z(TSP) \leq Z^f(TSP)$, we have

$$Z(VRPD_{1,\alpha,k}) \geq Z(TSP) - nU\alpha$$

after rearranging the terms. $\square$

The result in Theorem 1 is in a different style from those in Table 2.1. In Table 2.1, we consider the ratios of optimal objective function values, that is, the maximum relative benefit from using drones. But in Theorem 1, we consider the difference in objective function values, which indicates the maximum absolute benefit from using drones.

The maximum amount we can save by adding drones to trucks, i.e., $nU\alpha$, is directly proportional to drone battery life and the number of drone deployments. In other words, long range drones and high utilization rates both could help reduce costs. If the operating range in distance ($U\alpha$) is small due to battery constraints and the number of drone deployments n is also small (perhaps due to practical constraints like a small number of available batteries or customer locations that are very spread out), this lower bound may be more restrictive.

The inequality $Z(VRPD_{1,\alpha,k}) \geq \frac{Z(TSP)}{\alpha k+1}$ from Theorem 4 in [15] is still valid if the drones have limited battery life. Considering both theorems, we have $Z(VRPD_{1,\alpha,k}) \geq \max\{\frac{Z(TSP)}{\alpha k+1}, Z(TSP) - nU\alpha\}$.

2.2.2 Truck and Drones Utilizing Different Metrics

In [15], the drones and the trucks follow the same distance metric. In practice, we expect the drones to more or less follow the crow-fly distance and the trucks to be restricted to the street network. Therefore, the worst-case ratios in [15] are conservative in practice. Of course, this dichotomy ignores the reality of high-rise buildings and other aerial obstructions.

We show what happens to the worst-case result if the drone and the truck follow different distance metrics in the following theorem. The distance matrices followed by a truck and a drone are denoted by Q_t and Q_d, respectively. The $(i, j)^{\text{th}}$ entry of Q_t (or Q_d), denoted by $Q_t(i, j)$ (or $Q_d(i, j)$), is the distance traveled by the truck (or drone) from node i to node j. We denote the duration of the optimal TSP solution by $Z(TSP, Q_t)$, and we denote the optimal $VRPD_{m,\alpha,k}$ solution by $Z(VRPD_{m,\alpha,k}, Q_t, Q_d)$. We also make the additional assumption that $Q_d(i, j) \leq Q_t(i, j)$, $\forall i, j$. This implies drones will never travel further between two nodes than a truck.

Theorem 2.

$$\frac{Z(TSP, Q_t)}{Z(VRPD_{m,\alpha,k}, Q_t, Q_d)} \leq \frac{Z(TSP, Q_t)}{Z(TSP, Q_d)} m(\alpha k + 1).$$

Proof of Theorem 2. In our previous paper, we have shown that

$$\frac{Z(TSP, Q_d)}{Z(VRPD_{m,\alpha,k}, Q_d, Q_d)} \leq m(\alpha k + 1).$$

Divide by $Z(TSP, Q_d)$ to get

$$\frac{1}{Z(VRPD_{m,\alpha,k}, Q_d, Q_d)} \leq \frac{1}{Z(TSP, Q_d)} m(\alpha k + 1).$$

Next, multiply both sides by $Z(TSP, Q_t)$ to obtain

$$\frac{Z(TSP, Q_t)}{Z(VRPD_{m,\alpha,k}, Q_d, Q_d)} \leq \frac{Z(TSP, Q_t)}{Z(TSP, Q_d)} m(\alpha k + 1). \tag{2.2}$$

Since $Q_d(i, j) \leq Q_t(i, j)$, it follows that

$$Z(VRPD_{m,\alpha,k}, Q_d, Q_d) \leq Z(VRPD_{m,\alpha,k}, Q_t, Q_d) \leq Z(VRPD_{m,\alpha,k}, Q_t, Q_t) \tag{2.3}$$

because in the worst case, when a vehicle utilizes the Q_d metric, it is possible to use the same set of routes, but inject artificial waiting periods to simulate the Q_t metric. Theorem 2 follows directly from equations (2.2) and (2.3) above. □

This is similar to our bound from the previous paper:

$$\frac{Z(TSP, Q_t)}{Z(VRPD_{m,\alpha,k}, Q_t, Q_t)} \leq m(\alpha k + 1).$$

In Theorem 2, we have an additional factor $B = \frac{Z(TSP,Q_t)}{Z(TSP,Q_d)}$ which compensates for the different metrics. If drones travel as the crow flies, we know that $B \geq 1$.

Theorem 3.

$$\frac{Z(VRP^*, Q_t)}{Z(VRPD_{m,\alpha,k}, Q_t, Q_d)} \leq \frac{Z(VRP^*, Q_t)}{Z(VRP^*, Q_d)}(\alpha k + 1).$$

Proof. We know from Theorem 6 of the previous paper that

$$\frac{Z(VRP^*, Q_d)}{Z(VRPD_{m,\alpha,k}, Q_d, Q_d)} \leq \alpha k + 1.$$

If we divide both sides by $Z(VRP^*, Q_d)$, we obtain

$$\frac{1}{Z(VRPD_{m,\alpha,k}, Q_d, Q_d)} \leq \frac{1}{Z(VRP^*, Q_d)}(\alpha k + 1).$$

Next, multiply both sides by $Z(VRP^*, Q_t)$ and we get

$$\frac{Z(VRP^*, Q_t)}{Z(VRPD_{m,\alpha,k}, Q_d, Q_d)} \leq \frac{Z(VRP^*, Q_t)}{Z(VRP^*, Q_d)}(\alpha k + 1).$$

As with the previous Theorem, we note equation (2.3). Theorem 3 follows directly.

□

The implication of the above theorem is that with the $VRPD$ model, it is not only possible to take advantage of parallelization (with a speed-up factor of up to $\alpha k + 1$ relative to VRP^*), but the use of the crow-fly metric allows for an additional speed-up (up to a factor of $\frac{Z(VRP^*, Q_t)}{Z(VRP^*, Q_d)}$). In Appendix C we display a simple geometric example where the $VRPD$ speed-up ratio actually exceeds $\alpha k + 1$ due to the ability to use the crow-fly metric.

2.2.3 Economic Savings

While minimizing the completion time is the primary objective, a company will want to consider costs, as well. In the next theorem, we combine the completion time and the variable costs of using the truck and drone to form a new objective function, denoted by Y. Therefore, the new objective function for a TSP solution is given by $Y(TSP) = Z(TSP) + \theta X(TSP)$, where $X(TSP)$ denotes the variable cost of truck usage and θ allows us to attach weights to the two components of the objective function. When $\theta = 0$, we are minimizing the completion time. When θ is very large, we are minimizing the sum of the variable costs. The new objective function value of the optimal $VRPD_{1,\alpha,k}$ solution is calculated by $Y(VRPD_{1,\alpha,k}) = Z(VRPD_{1,\alpha,k}) + \theta X(VRPD_{1,\alpha,k})$, where $X(VRPD_{1,\alpha,k}) = X_t + X_d$, the sum of truck and drone usage costs. We assume the cost per unit time of the drone is a times the cost per unit time of the truck. We expect a to be much less than 1 because we assume that drones will fly autonomously once they leave the truck. The drone usage cost is incurred only when the drone is airborne. We ignore the fixed costs for now.

Theorem 4. *If the triangle inequality is valid, then*

$$Y(VRPD_{1,\alpha,k}) \geq Y(TSP) - \left[\frac{\alpha}{a} + \left(\frac{\alpha}{a} - 1\right)\theta\right] X_d,$$

where X_d is the variable cost of k drones in the optimal $VRPD_{1,\alpha,k}$ solution.

The coefficient $\left[\frac{\alpha}{a} + \left(\frac{\alpha}{a} - 1\right)\theta\right]$ is positive if $\alpha > a$. The potential savings from using a drone is large if α, θ, and X_d are large while a is small. We also point out the similar structure of the inequalities in Theorems 1 and 4.

Proof of Theorem 4. As noted earlier, we assume the truck speed is 1 and the drone speed is α. We further assume that the truck usage cost is 1 per unit time and the drone usage cost is a per unit time, so that $X(TSP) = 1 \times Z(TSP)$. If not, we can modify the parameter θ to normalize the usage costs of the vehicles. Note also that $Y(TSP) = Z(TSP)+\theta X(TSP) = (1+\theta)Z(TSP)$. Therefore, a TSP solution that minimizes duration also minimizes the total cost Y.

We want to find a lower bound on $Y(VRPD_{1,\alpha,k})$ in terms of $Y(TSP)$. This is similar to what we did in Theorem 1. From Table 2.1,

$$Z(TSP) \leq (\alpha k + 1)Z(VRPD_{1,\alpha,k}). \tag{2.4}$$

Since the truck usage cost is 1 per unit time, we have $X_t = Z(VRPD_{1,\alpha,k})$, where X_t is the truck usage cost in the optimal $VRPD_{1,\alpha,k}$ solution. Then,

$$Z(TSP) \leq (\alpha k + 1)X_t = X_t + \alpha k X_t. \tag{2.5}$$

Using the same construction process described in the proof of Theorem 1, we can show that an upper bound on the truck usage cost is given by

$$X(TSP) \leq X_t + \frac{X_d}{a}\alpha. \tag{2.6}$$

We construct a feasible TSP solution from the optimal $VRPD_{1,\alpha,k}$ solution. We insert the drone customers one by one onto the truck route whose variable cost was initially equal to X_t. The additional cost due to the drone customers is $\frac{X_d}{a}\alpha$. The factor $\frac{X_d}{a}$ gives the sum of usage time of the k drones. Multiplying it by the drone speed α gives the maximum total distance covered by the k drones. Since the truck has unit speed and unit usage cost, the term $\frac{X_d}{a}\alpha$ also gives the additional usage cost when we convert the optimal $VRPD_{1,\alpha,k}$ to a feasible TSP solution.

The left-hand sides of inequalities (2.5) and (2.6) are equal, i.e., $Z(TSP) = X(TSP)$ because we assume that truck usage cost is 1 per unit time. The two inequalities give two upper bounds on $Z(TSP)$. The tighter upper bound is $X_t + \frac{X_d}{a}\alpha$ given by (2.6), because $\frac{X_d}{ak} \leq \frac{X_t}{1}$, as the average usage time per drone is never greater than the usage time of the truck.

Now,

$$
\begin{aligned}
Y(TSP) &= (1+\theta)Z(TSP) \\
&\leq (1+\theta)\left(X_t + \frac{\alpha}{a}X_d\right) \\
&= X_t + \theta(X_t + X_d) + \left[\frac{\alpha}{a} + \left(\frac{\alpha}{a} - 1\right)\theta\right] X_d \\
&= Y(VRPD_{1,\alpha,k}) + \left[\frac{\alpha}{a} + \left(\frac{\alpha}{a} - 1\right)\theta\right] X_d,
\end{aligned}
$$

which yields the desired result. □

2.3 Extension to $CEVRP^*$

Suppose there exists the following node locations along a street network $P = \{P_1, P_2, ..., P_{|P|}\}$, each requiring a package to be delivered to them from depot D. In the traditional traveling salesman problem (TSP), one may insist that a truck stop at all of these locations, then finally return to D. The min-max close-enough traveling salesman problem ($CETSP^*$) has the same objective as the ordinary traveling salesman problem (i.e., minimize the time required to visit all node locations and return to the depot). However, in the min-max $CETSP$, we assume we need not necessarily visit location P_i itself. We only need to come within distance $R_i \geq 0$ of P_i [8]. Coming within distance R_i of a node P_i is "close enough" for some important applications. For example, utility companies use automated meter reading with RFID to read meters from a distance for billing purposes. Military applications involve surveillance from a distance.

In the min-max vehicle routing problem (VRP^*), we wish for at least one truck out of a set of m homogeneous trucks to visit each customer location $P_i \in P$, then return to the depot. The objective is to minimize the time until all sites are visited and all trucks have returned to the depot. Analogously, we define the min-max close-enough vehicle routing problem ($CEVRP^*$) to be the problem of minimizing the time required for at least one in a set of m trucks to come within some distance R_i of each customer location $P_i \in P$ before returning to the depot.

In this section, we will show that there exists a strong relationship between VRP^*, $VRPD$, and $CEVRP^*$. In future work, we hope to show how this relationship enlightens computational heuristics for finding solutions to the $VRPD$. Moreover, if we have reliable VRP^* and $CEVRP^*$ solvers, this relationship will indicate whether our computational solutions are near-optimal.

2.3.1 VRPD: An Intermediate Problem

Let us define two problems. Firstly, let $VRPD_{ur}$ be the *unrestricted* $VRPD$. This problem has the same characteristics as the $VRPD$, except that drone launch and retrieval locations are not restricted to nodes. This is more consistent with $CEVRP^*$, which does not mandate a covering point of P_i to be a nodal point.

Secondly, let $CEVRP^*_{nodes}$ represent a problem similar to $CEVRP^*$. However, $CEVRP^*_{nodes}$ is stricter. $CEVRP^*_{nodes}$ requires that for each customer P_i, there exists a *nodal* location on some truck route within distance R_i of customer P_i. This is consistent with the $VRPD$ model where launch and retrieval points occur only at node locations.

Theorem 5.

$$Z(CEVRP^*) \leq Z(VRPD_{ur}) \leq Z(VRP^*);$$

$$Z(CEVRP^*_{nodes}) \leq Z(VRPD) \leq Z(VRP^*).$$

These claims are constructed from four inequalities which are proved formally in Appendix B. In less formal terms, we note that the truck routes from the optimal VRP^* solution act as feasible $VRPD$ and $VRPD_{ur}$ routes (that simply do not utilize any drone delivery capabilities). Thus $VRPD$ and $VRPD_{ur}$ can always do at least as well as VRP^*. However, $VRPD$ and $VRPD_{ur}$ may be able to do better by making some drone deliveries.

Similarly, the truck routes from the optimal $VRPD$ (or $VRPD_{ur}$) are feasible solutions to the $CEVRP^*_{nodes}$ (or $CEVRP^*$) problems. Thus, the optimal solution to $CEVRP^*_{nodes}$ (or $CEVRP^*$) is no worse than the optimal solution to $VRPD$ (or $VRPD_{ur}$).

2.3.2 $VRPD$ in the Limit

In this section, we will consider the limit cases of drone speed, namely when α approaches 0 and when α approaches ∞.

Theorem 6.

$$lim_{\alpha \to \infty} Z(VRPD, \alpha) = Z(CEVRP^*_{nodes}).$$

Proof. We establish in Appendix B that every $CEVRP^*_{nodes}$ solution can be converted into a $VRPD$ solution. This is done by starting with the truck route of the $CEVRP^*_{nodes}$ solution. However, the truck waits at the drone release point until the drone delivers its package and returns to the truck. This trivial feasible solution to $VRPD$ is called $VRPD_f$. Let W_j be the sum of all such wait times on the jth truck's $VRPD$ route. Let $W = max_j(W_j)$. By this construction, it is clear that

$$Z(CEVRP^*_{nodes}) + W \geq Z(VRPD_f) \geq Z(VRPD).$$

The upper bound distance on any drone flight, again, is $2R = V$. Thus $2R/\alpha$ is the maximum duration that a truck would wait for any drone to deliver a package. Let M be the maximum number of customers on any route. So $0 \leq W \leq 2MR/\alpha$. Given a finite number of customers,

$$lim_{\alpha \to \infty} W = 0.$$

Furthermore, as $\alpha \to \infty$

$$Z(CEVRP^*_{nodes}) = Z(VRPD_f) \geq Z(VRPD).$$

However, we established in a previous theorem that $Z(CEVRP^*_{nodes}) \leq Z(VRPD)$. Therefore, as $\alpha \to \infty$

$$Z(CEVRP^*_{nodes}) = Z(VRPD_f) = Z(VRPD).$$

□

Theorem 7 (Fast Drone Theorem)**.**

$$lim_{\alpha \to \infty} Z(VRPD_{ur}, \alpha) = Z(CEVRP^*).$$

Proof. The proof is identical in structure to Theorem 6 with one minor exception. Namely, we now may designate any point within distance R of a customer as a launch/retrieval point, rather than being restricted to nodal locations. We then force trucks to wait at these launch points until the drone(s) returns. The total required waiting time of all trucks (again) converges to 0 as $\alpha \to \infty$. $\square$

The two theorems above show that as drone speed goes to ∞, the $VRPD$ is an equivalent problem to $CEVRP^*$. However, it also hints that perhaps a $CEVRP^*$ solution would be a good approximation to the $VRPD$ solution whenever the ratio of drone speed to truck speed is high. In environments with highly congested roadways, but a relatively unobstructed sky, or perhaps when utilizing very high speed drones, it may be worth starting with a $CEVRP^*$ solution, and adapting it into a $VRPD$ solution.

Theorem 8 (Slow Drone Theorem)**.**

$$lim_{\alpha \to 0} Z(VRPD, \alpha) = Z(VRP^*)$$

and

$$lim_{\alpha \to 0} Z(VRPD_{ur}, \alpha) = Z(VRP^*).$$

Proof. If our optimal $VRPD$ or $VRPD_{ur}$ solution has no drone launches, then the solution is the same as the optimal VRP^* solution. In this case, the above equality holds.

Now suppose our $VRPD$ or $VRPD_{ur}$ solution has some drone flight of non-zero length (out and back). Let L be the longest of such routes. Then as $\alpha \to 0$, the time required for such a route is $L/\alpha \to \infty$. This implies that as drone speed goes to

0, any $VRPD$ or $VRPD_{ur}$ solution containing a drone launch (e.g., $VRPD_f$) is such that $lim_{\alpha \to 0} Z(VRPD_f, \alpha) = \infty$. Supposing our truck speed is non-zero, a VRP^* solution would require a finite amount of time. This proves that as $\alpha \to 0$, $VRPD$ and $VRPD_{ur}$ solutions should not contain drone launches to remain optimal. $\square$

In Section 4.1, we showed $VRPD$'s objective value was bounded below by the objective value of $CEVRP^*$ and bounded above by the objective value of VRP^*. Now in Section 4.2, we have shown that $VRPD$ and $CEVRP^*$ are equivalent problems for an arbitrarily fast drone; $VRPD$ and VRP^* are equivalent problems for an arbitrarily slow drone.

Other than the bounds on optimal objective values, we do not yet know the relationship between $VRPD$ optimal solutions and optimal solutions to $CEVRP^*$ and VRP^* for intermediate values of α (i.e., $0 < \alpha < \infty$). Furthermore, we do not know how this relationship is affected by our choice of α, the underlying street network, or the drone network.

2.4 Conclusions and Future Work

This paper extends and strengthens previous results in [15]. $VRPD$ is one model that attempts to complement the carrying capacity and range of a truck with the ability of a drone to help "parallelize" delivery and take advantage of crow-fly distances. This paper has shown the theoretical maximum benefit of this model under ideal circumstances.

A connection between the $CEVRP^*$, $VRPD$, and VRP^* has been made in the form of objective value bounds and asymptotic results. We believe that a number of computational heuristics and benchmark instances could now be developed to find $VRPD$ solutions as close to optimal as possible for practical values of α. Using solution methods for $CEVRP^*$ (such as in [3,5,11]), modifying them to maintain $VRPD$ feasibility, and then applying some local optimization procedures is one new idea for obtaining computational solutions to $VRPD$. An alternative idea is starting with a VRP^* solution and inserting drone deliveries in a smart way. In addition, one may compare computed objective values for $VRPD$ with $CEVRP^*$ and VRP^*, assessing the tightness of these theoretical bounds in practice for varying values of α.

Expanding the model to include limitations on drone package weight (while still allowing trucks to carry heavier packages) could add to the practical worth of this model. The study of other variations, such as allowing a drone to launch on one truck and land on another truck or allowing a drone to carry more than one package at a time, may also be considered.

Chapter 3: Exact and Heuristic Methods for the Traveling Salesman Problem with a Drone

3.1 Introduction

Several technological improvements including better battery life, improved communications systems, advances in stability, and reduction of manufacturing costs have increased the viability of using drones to make deliveries. Drones have been used in healthcare and disaster response contexts particularly in remote regions [9].

Amazon, FedEx, and UPS have explored the use of drones for parcel delivery [61]. In September 2015, the Finnish postal service (Posti Group) experimented with drone delivery of packages to an island near Helsinki [55]. Dynamic Parcel Distribution is the first company to have launched (with all regulatory approvals) a regular route service with a drone in the Provence region of France [23].

In February 2017, UPS released a video of a test of a truck and drone hybrid delivery [62] where the drone road atop the truck. The truck stopped near a customer location and launched the drone with a package to a different customer location to make a delivery. While the drone was in the air, the truck made a delivery and then rendezvoused with the drone. This hybrid model of delivery is the focus of this paper.

3.2 Literature Review

Murray and Chu [49] were the first to introduce a hybrid truck and drone model under the name The Flying Sidekick. In preliminary testing of a mixed integer linear programming model, the authors indicated that routes with up to 10 packages "may require several hours to solve" to optimality. The long solution times motivated a heuristic method, such as the one below:

1. Solve a standard TSP and use this as an initial truck route.

2. In a greedy way, select a package currently on the truck route to be delivered by a drone. This greedy selection preserves feasibility.

Ha et al. [37] solved a mixed integer program that optimized the selection of drone operations according to various objective functions. A drone operation with triplet (i, j, k) launches the drone from the truck at package location i, delivers a package via drone to package location j, and returns the drone to the top of the truck at package location k. A modified TSP routing was then performed based on the selection of drone operations from the mixed integer program.

Wang et al. [65] considered theoretical bounds for the maximum speedup ratio of using a hybrid truck with drone model relative to a truck only model. The authors described optimization problems that arose when using several trucks with one or more drones. Poikonen et al. [54] generalized the bounds of Wang et al. [65] to the case where trucks and drones operated on different metrics. The authors also showed that the close-enough vehicle routing problem may serve as a lower bound to the vehicle routing problem with drones.

Agatz et al. [2] considered a problem similar to Murray and Chu [49] that they named the Traveling Salesman Problem with a Drone (TSP-D). They constructed a family of heuristics and an integer program and found that the best performing heuristic without applying iterative, local improvement procedures was TSP-ep, where ep denotes exact partitioning. First, a standard traveling salesman problem was solved. The goal was then to exactly partition this solution into truck-delivered nodes and drone-delivered nodes. Without loss of generality, the authors relabeled the nodes with indices $0, 1, 2, ..., N$ such that if node a appeared before node b in the standard TSP solution, then $a < b$. Node 0 is the origin depot and node N is the destination depot which may be the same as the origin depot.

Agatz et al. [2] continue by considering the case when $i < k < j$ and that the truck and drone are located together at node i. The drone launches from the truck to node k to deliver a package. While the drone is airborne, the truck delivers to every node $l \neq k$ such that $i < l < j$, and then both the truck and drone rendezvous at node j. Agatz et al. [2] defined $T(i, j, k)$ as the amount of time between the drone launch at node i and the rendezvous at node j. $T(i, j, k) = \infty$ when a triplet is infeasible. They defined $T(i,j) = min_k(T(i,j,k))$. It is beneficial to choose the package location k that minimizes the time until the rendezvous at j. Let $k = -1$ indicate that truck delivery to all nodes l such that $i < l < j$ produces a faster arrival to node j than launching a drone. TSP-ep used the following recursive formula (a special case of the Bellman-Ford Equation [13]):

$V(0) = 0$

For i=1 to N:

$V(i) = min_k(V(k) + T(k,i))$

$Prev(i) = argmin_k(V(k) + T(k,i))$

$V(N)$ is the best TSP-D objective value under the restriction that both the truck delivery order and the drone delivery order are subsequences of the standard TSP solution. Though optimal under this restriction, in general, TSP-ep does not provide the globally optimal solution to the TSP-D. One may retrace the optimal path by iteratively backtracking from node N to $Prev(N)$, then to $Prev(Prev(N))$, then to $Prev(Prev(Prev(N)))$, etc. until reaching the depot where the truck and drone departed. The cost of exactly partitioning a TSP sequence into a TSP-D solution is $O(N^3)$.

Agatz et al. [2] embedded their exact partitioning procedure in several iterative improvement procedures with the following structure.

1. Find the the optimal TSP solution, called $BestTour$.
2. Construct a neighborhood of similar tours around $BestTour$. Call it $Neighbors$.
3. For each $tour$ in $Neighbors$
   (a) Apply the exact partitioning method.
   (b) Compute the associated TSP-D objective value of the partitioned route, called $Obj(tour)$.
   (c) If $Obj(tour) < Obj(BestTour)$, set $NewBestTour = tour$.
4. If $BestTour \neq NewBestTour$, go to step 2.

Agatz et al. [2] tested heuristics including TSP-ep and TSP-ep-all, where all refers to a neighborhood of routes that can be constructed using any local swaps described in their paper. TSP-ep-all considers $O(N^2)$ neighboring TSP routes in each iteration, so it has a total computational complexity of $O(IN^5)$, where I is the number of iterations.

Coutihno et al. [20] considered the Close-Enough Traveling Salesman Problem (CETSP) which is a generalization of the TSP where a city is considered visited if the tour comes within a specified radius of the city. The key feature of Coutihno et al. [20] is their branch-and-bound solution methodology where each node of the branch-and-bound tree is associated with some sequence of visit locations, S. At each node of the tree, a second-order cone program (SOCP) was solved that obeys the visit order dictated by S. Though the visit order is fixed, the SOCP is free to choose the optimal representative point for each visit location (a representative point is within the specified radius of a given city) for each visit location. To put it another way, the branch-and-bound structure served as a mechanism to globally search potential visit sequences. The SOCP that was solved at each node optimized the CETSP solution relative to this sequence. The solution method produced exact solutions to the CETSP.

Our solution method (described in detail in Section 4) borrows certain elements from Coutihno et al. [20] and Agatz et al. [2]. Specifically, the branch-and-bound structure in our solution method is derived from Coutinho et al. [20] and allows us to search various visit sequences. Rather than optimizing an SOCP at each node, we optimally partition the sequence at each node into truck-delivered and drone-delivered nodes. We slightly modify the partitioning procedure from [2] such that the truck may remain stationary while the drone makes a delivery.

3.3 Defining the TSP-D and Notation

3.3.1 TSP-D: Problem Definition

We define the Traveling Salesman Problem with a Drone (TSP-D) as follows. There is one truck and one drone that may ride atop the truck. Let c_t and c_d be matrices of travel times. $c_t(i, j)$ is the value of row i and column j of c_t, and it is set as the time a truck takes to traverse from node i to node j. $c_d(i, j)$ is the value of row i and column j of c_d, and it is set as the time a drone takes to traverse from node i to node j. For all i, j, $c_t(i, j), c_d(i, j) \geq 0$, and the triangle inequality holds for c_t and c_d. Frequently, in our computational experiments, both the truck and the drone operate on the Euclidean metric. In these cases, we define $\alpha = c_t(i, j)/c_d(i, j), \forall i, j$, which is a measure of the relative speed of the drone to the truck. In general, c_t and c_d need not be scalar multiples of one another, in which case α is not defined.

There are N nodes, one depot, and $N - 1$ packages to be delivered, and a set of locations (P) for the packages and the depot. $P_t \subseteq P$ is the set of locations such that the package at that location must be delivered by a truck. Some packages may not be suitable for drone delivery due to complications such as excessive weight or an obstructed landing area. Let $P_d = P \setminus P_t$ be the set of package locations eligible for drone delivery. Each package $P_1, P_2, ..., P_{N-1}$ must be delivered either by truck or by drone. P_0 is the origin depot location, P_N is the destination depot location, and P_0 and P_N may be the same location.

The drone has a battery life of R time units. The drone may launch from atop the truck, carry a single package to a package delivery location, and then must rendezvous with the truck within R time units. The truck may deliver packages while the drone is airborne. Launch and rendezvous points must occur at package locations or the depot. The truck and drone do not need to arrive simultaneously; they can wait for each other to arrive, as long as the rendezvous happens within R time units of the drone's launch. In addition, a drone may be launched and retrieved at the same package location. We assume that after a drone lands on the truck, its battery may be swapped for a fully charged battery instantaneously.

For simplicity, we do not consider drone service times, drone launch overhead times, drone retrieval overhead times, or battery swap times, and we assume each time is negligible. However, it is possible to modify c_d and the computation of $T(i, j, k)$ in a small way to account for all of these times. In Part A of the Online Supplement, we describe how this can be done.

We must construct a simple tour (i.e., a tour that cannot depart a node and revisit that same node) beginning at depot P_0 and ending at depot P_N. If $P_0 = P_N$, the tour is closed. The departure time of the truck from the P_0 is $t_0 = 0$, all packages $P_1, ..., P_{N-1}$ have been delivered and the truck and drone have returned to P_N at time t_f. The objective is to minimize t_f.

3.4 Branch-and-bound Approach

We now describe our branch-and-bound approach (BAB) to the TSP-D. The pseudocode describing BAB is given in Part B of the Online Supplement.

3.4.1 Nodal Structure and Branching Procedure

Each node in our branch-and-bound decision tree is associated with some sequence of package locations, similar to Coutihno et al. [20] If c_t and c_d are symmetric and $P_0 = P_N$, then we assign our root node an arbitrary 3-cycle including the depot which can be done without loss of generality. Suppose we assign the sequence $[0, 1, 2, N]$ to the root node. This corresponds with a route that visits P_0, P_1, P_2, and returns to P_N in that order. If c_t and c_d are symmetric and $P_0 = P_N$, then the routes corresponding to $[0, 1, 2, N]$ and $[0, 2, 1, N]$ have the same objective value. In the case that c_t and c_d are not symmetric or $P_0 \neq P_N$, we assign the root node the sequence $[0, 1, N]$. Although it is possible to assign a symmetric instance $[0, 1, 0]$ as the root node, we choose $[0, 1, 2, 0]$ as the root. We take advantage of known symmetry and avoid the formation of two branches with $[0, 1, 2, 0]$ and $[0, 2, 1, 0]$, which unnecessarily doubles the computation time. For simplicity of notation, assume $P_0 = P_N$ unless specified otherwise.

Our tree begins with only the root node. We then create children of the root node. Find the package location P_i that is farthest (in Euclidean distance) from any package location in the parent's sequence. Suppose that the farthest package location from package locations P_0, P_1, and P_2 is package location P_3. Then the children of the root node $[0, 1, 2, 0]$ are $[0, 3, 1, 2, 0]$, $[0, 1, 3, 2, 0]$, and $[0, 1, 2, 3, 0]$. Our branching procedure inserts the farthest package into various positions of the parent node's sequence. In Figure 3.2, package delivery location P_3 is the farthest from P_0, P_1, and P_2, and package delivery location P_6 is the farthest from P_0, P_1, P_2, and P_3.

We represent a sequence by $S = [0, s_1, ..., s_n, 0]$, where n is the number of package locations visited in the sequence, and s_i is the package location in position i of the visit sequence.

3.4.2 Lower Bound Evaluation for a Node

Suppose $TSPSeq$ is the optimal TSP sequence. Let $aep(TSPSeq)$ be the objective value of TSP-ep from Agatz et al. [2] when we apply the exact partitioning function (aep) of Agatz et al. [2] to the input sequence denoted by $TSPSeq$. Let $aep(S)$ denote the objective value produced by applying the exact partitioning procedure to an input sequence of package locations, S.

In the aep function, drone operations (i, j, k) are considered only if $i < k < j$ (although, elsewhere in [2], this restriction is relaxed). Therefore, launching and retrieving a drone at the same customer node with the truck remaining stationary is impossible in any solution produced by aep, even though this may be characteristic of the optimal solution. Let ep denote an exact partitioning function that incorporates the possibility of the truck remaining stationary throughout the drone's flight into aep. The technical details of ep are given in Part C of the Online Supplement.

Suppose some node has an associated sequence $S = [0, s_1, s_2, ..., s_n, 0]$. At this node, we seek to find a partition of S into an ordered set of packages delivered by the truck (S_t) and an ordered set of packages delivered by the drone (S_d). However, rather than requiring S_t and S_d to be subsequences of a specified TSP solution, we require that S_t and S_d be subsequences of S. Thus, if a node has associated sequence S, it has an associated objective value $ep(S)$. The result is the optimal TSP-D objective value for delivering packages $s_1, s_2, ..., s_n$ subject to the constraint that S_t and S_d are subsequences of S. For a node with sequence S, we say that is has an assumed lower bound of $ALB(S) = ep(S)$.

Suppose some parent node has a sequence S. Suppose its child has a sequence S^+, which is necessarily a supersequence of S. Our algorithm works under the assumption that the insertion of additional package stop locations onto a TSP-D route generally increases the objective value, that is, we assume that $ep(S^+) \geq ep(S) = ALB(S)$. Thus, for a parent node with sequence $S = [0, s_1, s_2, ..., s_n, 0]$, we assume that the objective value of any child node is at least $ALB(S)$. Transitively, any direct descendant node is assumed to have a larger objective value than its ancestor.

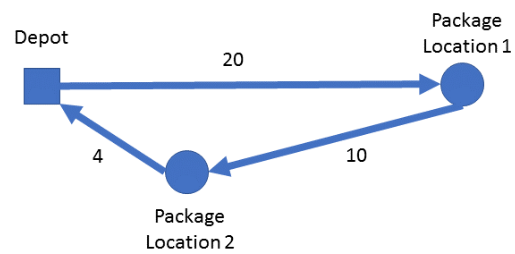
The insertion of additional package locations onto the TSP-D route can actually decrease the objective value (unlike TSP routes, where package insertions can never decrease the objective value), i.e., occasionally $ep(S^+) < ep(S)$. This is directly related to the finite range of the drone. Including additional stops in the route introduces new locations where a drone could potentially launch or land. Thus, a package that would have been delivered by a truck (due to the lack of any launch or landing locations suitable for the range of the drone) could potentially be delivered by the drone, after a new stop location is added to the route. In Figure

3.1, we provide an example where inserting an additional package location onto the route decreases the objective value. In this example $R = 10$, blue edges represent truck movement, red edges represent drone flight, and the number next to each edge is the time required to traverse the edge. Numbers next to blue edges are driving times in minutes; numbers next to red-dashed edges are flying times in minutes. In Figure 3.1(a), the truck drives to package location 1, because the drone only has 10 minutes of battery life. The drone does not have enough battery life to fly to package location 1, make a delivery, and return to the truck. The completion time is $20 + 10 + 4 = 34$ minutes. In Figure 3.1(b), the insertion of package location 3 gives the drone a feasible launching point to deliver to package location 1. The drone launches from package location 3, makes a delivery to package location 1, and lands on the truck at package location 2 with a completion time of $8 + 9 + 4 = 21$ minutes.

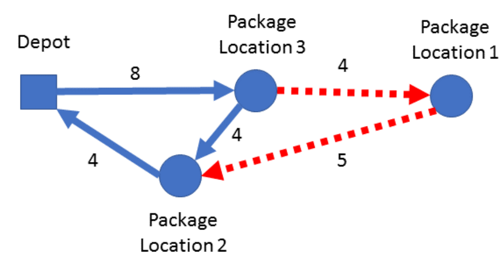
Although the insertion of a new package location onto a TSP-D route occasionally decreases the objective value, in testing, we found that it is more likely that inserting additional package locations increases the objective values.

3.4.3 Exploration, Upper Bounds, and Terminating the Algorithm

If a parent node with sequence $[0, s_1, s_2, ..., s_n, 0]$ has been evaluated and its children have not yet been evaluated, then the lower bound of the parent node is $ALB([0, s_1, s_2, ..., s_n, 0])$. If all children of a parent have been evaluated, then the



(a)



(b)

Figure 3.1: Example where the insertion of an additional package location decreases the objective value.

lower bound of the parent node is equal to the smallest lower bound of the children.

Among all nodes whose children have not yet been evaluated (leaf nodes), we iteratively choose to evaluate the children of the node with the smallest lower bound. If a sequence contains all N package locations, then it is marked as a feasible solution to the overall problem. The assumed lower bound for that node is also an upper bound.

Let LB be the smallest lower bound among nodes that still have unexplored children, and let UB denote the best objective value found for any complete feasible solution (this solution contains all package locations). If no complete feasible solution has yet been found, then $UB = \infty$. We terminate the branch-and-bound algorithm once $LB/UB \geq TER$, where $TER \geq 1$ is our tree exploration ratio. If our assumed lower bound was always a valid lower bound, then setting $TER = 1$ would yield the globally optimal solution. Since our assumed lower bound is sometimes too high, we may compensate for this by setting $TER > 1$.

For example, suppose we set $TER = 1.15$ and find a complete feasible solution with objective value of 100. If we did not find a new complete feasible solution with objective value less than 100, then our algorithm would terminate when all nodes with lower bounds less than 115 had been explored. Thus, our solution is globally optimal, if we did not overestimate the lower bound of any node by more than $0.15(UB)$.

An alternate, but logically equivalent, interpretation is that the assumed lower bound of a node with the associated sequence $[0, s_1, s_2, ..., s_n, 0]$ is given by $ALB([0, s_1, s_2, ..., s_n, 0])/TER$ and we terminate the algorithm when $LB \geq UB$.

After we explore a feasible solution, we set its lower bound to INF. We define INF as any value greater than $N \times maxEdge$, where $maxEdge = max_{i,j} c_t(i, j)$. The value $N \times maxEdge$ serves as an upper bound to the objective value of any feasible sequence partitioned by *ep*. If $TER = \infty$, BAB terminates when the root

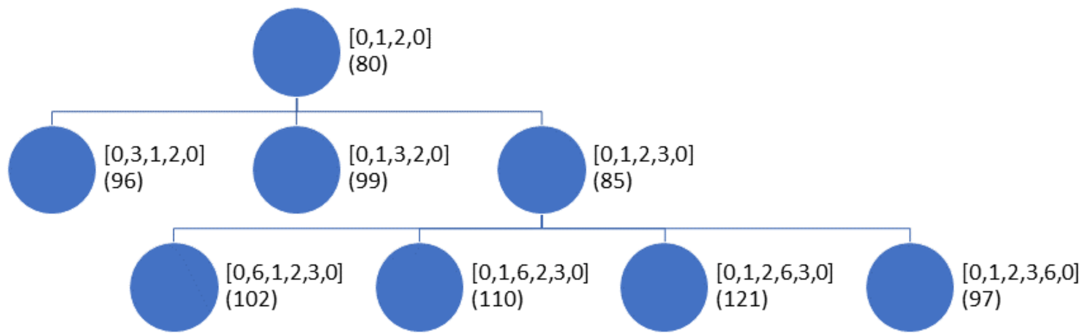


Figure 3.2: Initial exploration of a branch-and-bound tree with associated sequences and objective values in parentheses.

node's lower bound is equal to INF, which occurs only when all feasible solutions have been explored.

3.4.4 Example of the Branch-and-Bound Approach

In Figure 3.2, we begin with evaluating the root node which produces an objective value of 80. We then evaluate all of its children and the child with sequence [0,1,2,3,0] has the lowest objective value of 85, so we then evaluate its children. Among all leaf nodes of the tree, the one with sequence [0,3,1,2,0] has the lowest objective value of 96 and we would explore its children next.

In Figure 3.3, we display an example with four package locations in addition to the depot. The full exploration of the BAB tree when $TER = 1.00$ is shown. If $TER = 1.15$, then we evaluate the children of the red node with objective value of 112, because it has an objective value less than $1.15 \times 100 = 115$.

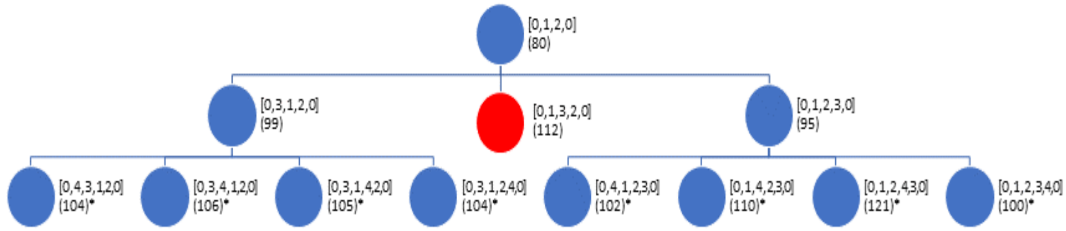


Figure 3.3: Full exploration tree for BAB when $TER = 1.00$. Associated sequences are in brackets, objective values are in parantheses, and an asterisk indicates a feasible solution that visits all package locations. As shown, $LB = 112$ and $UB = 100$, so the heuristic terminates, because $LB/UB = 1.12 > 1.00$. The red node with objective value 112 has unevaluated children.

3.4.5 Reduction to O(Cn^2)

When computing $T(i, j, k)$ for each $i < k < j$, we have an $O(n^3)$ computation. Since this computation occurs for each node visited in the branch-and-bound tree, this computation becomes very costly. Therefore, before starting the branch-and-bound approach, we compute a constant C associated with c_t, the truck network metric. C is the largest integer such that a truck may visit C distinct nodes on the street network within R time units.

Now, we need only compute $T(i, j, k)$ for each $i < k < j \leq i + C$. We need not compute any potential drone deliveries for $j \geq i + C + 1$. Suppose $j \geq i + C + 1$. There are at least $C + 2$ nodes between i and j, inclusive of the endpoints. If some node k is serviced by the drone, then at least $C + 1$ nodes must be visited by the truck within time R before the drone battery loses its charge. However, C is the maximum number of nodes visited by the truck in time R. Computing $T(i, j, k)$ is unnecessary whenever $j \geq i + C + 1$ due to infeasibility associated with the battery life of the drone.

Therefore, we have reduced the computational complexity of each nodal evaluation from $O(n^3)$ to $O(Cn^2)$ at the cost of computing C once before starting the BAB procedure. C may be computed exactly by solving the integer program given in Part D of the Online Supplement. In areas of low density and little clustering of delivery locations, we expect C to be small. Alternatively, we could compute an upper bound on C, denoted C^+, by summing the smallest elements of c_t until the sum exceeds R. The number of distinct elements that may be summed before exceeding R is C^+. Then we only compute $T(i, j, k)$ for each $i < k < j \leq i + C^+$.

3.5 Additional Heuristics for the TSP-D

3.5.1 Boosted Lower Bound Heuristic

In practice, the computation time required to perform the branch-and-bound algorithm is heavily dependent on the ability to prune large portions of the decision tree. BAB was built on the assumption that as more package locations are inserted into a sequence, the objective values associated with that sequence strictly increase. Thus, among two sequences with the same associated objective value, we prefer the longer sequence, because it has fewer packages that need to be inserted to form a feasible solution.

Consider the sequence $[0, s_1, s_2, ..., s_n, 0]$ that does not visit $N-1-n$ package locations that are required for a full global solution. We define our heuristic lower bound (HLB) as

$$HLB([0, s_1, s_2, ..., s_n, 0]) = ALB([0, s_1, s_2, ..., s_n, 0]) + f(N-1-n)$$

where f is an increasing function and $f(0) = 0$.

Our boosted lower bound heuristic uses the original branch-and-bound structure and branching process. However, we replace the objective value of each node and, thus, the lower bound on all descendant nodes ALB with HLB. When we find a feasible solution at some node, $N-1 = n$, $f(N-1-n) = f(0) = 0$. Thus, for any feasible sequence S with all package locations, $HLB = ALB$.

3.5.1.1 Linear Boost Heuristic

Let $f(N-1-n) = \gamma(N-1-n)$, where $\gamma > 0$ is a specified constant. Our linear boost heuristic assumes that, for any sequence, the insertion of additional packages into that sequence will cost at least an additional γ time units per package on average.

Consider the example in Figure 3.2. The next step in BAB evaluates the children of the node with sequence [0,3,1,2,0] and objective value 96. Suppose that $N = 11$. The linear boost heuristic (denoted by BAB+L for the branch-and-bound method plus an additional linear term) with $\gamma = 2$ has a lower bound of $96 + 2(7) = 110$ for the sequence [0,3,1,2,0], because there are seven package locations that have not yet been inserted into the sequence. The node with sequence [0,1,2,3,6,0] has a lower bound of $97 + 2(6) = 109$, because there are six remaining package locations to be inserted into the sequence. BAB+L would evaluate the children of the node with sequence [0,1,2,3,6,0] next.

3.5.1.2 Quadratic Boost Heuristic

Our branching procedure is similar to farthest insertion. Those package locations farthest away are inserted before those package locations that are nearest to the existing subtour. The second variant of our heuristic assumes that the marginal cost per package insertion is larger for short sequences. As the subtour grows and iteratively inserts the package that is farthest away, the marginal insertion cost is assumed to decrease.

Rather than using $f(N-n) = (N-n)\gamma$, which is linear in the number of package locations not yet inserted (i.e., constant marginal insertion cost of γ), we use $f(N-n) = (N-n)^2\gamma$ which is consistent with decreasing marginal insertion costs as n increases. We denote this method by BAB+Q.

3.5.2 Divide-and-Conquer Heuristic

For BAB, BAB+L, and BAL+Q, computation times increase superlinearly (see Section 6). For large problem sizes, we consider a different heuristic method that we call the divide-and-conquer heuristic (denoted by DCH).

Let $CT(N)$ be the average computation time for BAB for instances of size N. Since $CT(N)$ increases superlinearly, $mCT(N/m) < CT(N)$. Thus, we expect the computation time of solving m problems of size N/m to be less than the computation time of solving a single larger problem of size N.

DCH begins by solving the standard TSP on the truck metric. We relabel nodes according to their order of appearance in the standard TSP solution. The first node visited in the standard TSP solution is relabeled node 1; the second node visited in the standard TSP solution is relabeled node 2; generally, the ith node visited in the standard TSP solution is relabeled node i. Node 0 and node N may be identical and serve as the origin and destination depots.

Next, we split the relabeled nodes from the TSP solution into m groups. The first group has nodes $0, 1, 2, ..., \lfloor N/m \rfloor$. The second group has nodes $\lfloor N/m \rfloor, \lfloor N/m \rfloor + 1, ..., \lfloor 2N/m \rfloor$. Generally, group i has nodes $\lfloor (i-1)N/m \rfloor, \lfloor (i-1)N/m \rfloor + 1, ..., \lfloor iN/m \rfloor$. Thus, the node set is divided into m groups where each group has a size of $\lfloor N/m \rfloor$ or $\lfloor N/m \rfloor + 1$.

For each of the m groups, we solve a subproblem. In particular, we solve a TSP-D on the set of nodes in each group with a condition. For group i, we set the root node sequence to $[\lfloor(i-1)N/m\rfloor, \lfloor iN/m\rfloor]$. Node $\lfloor(i-1)N/m\rfloor$ acts as the origin depot for this subproblem; node $\lfloor iN/m\rfloor$ acts as the destination depot for this subproblem. Then each subproblem is solved using BAB. Any node j such that $\lfloor(i-1)N/m\rfloor < j < \lfloor iN/m\rfloor$ is inserted between the origin and depot nodes on subproblem i. The full problem solution is the union of the solutions of all subproblems in order.

In Figure 3.4, we give an example with $N = 30$ nodes and $m = 3$. A standard TSP route has already been specified and nodes have been relabeled accordingly. Subproblem 1 requires that the truck and drone start at node 0 and service nodes 1 through 9 in some order. After servicing nodes 1 through 9, the truck and drone must rendezvous at node 10. Subproblem 2 requires that the truck and drone start at node 10, service nodes 11 through 19 in some order, then rendezvous at node 20. Subproblem 3 requires that the truck and drone start at node 20, service nodes 21 through 29 in some order, then rendezvous at node 30. Combining the solution to all subproblems produces a solution to the full problem with $N = 30$.

The intuition behind DCH is that the truck route in a good TSP-D solution may have a similar broad shape to the optimal TSP solution. By solving each subproblem, we optimize the local structure. In Figure 3.4, we have the flexibility to

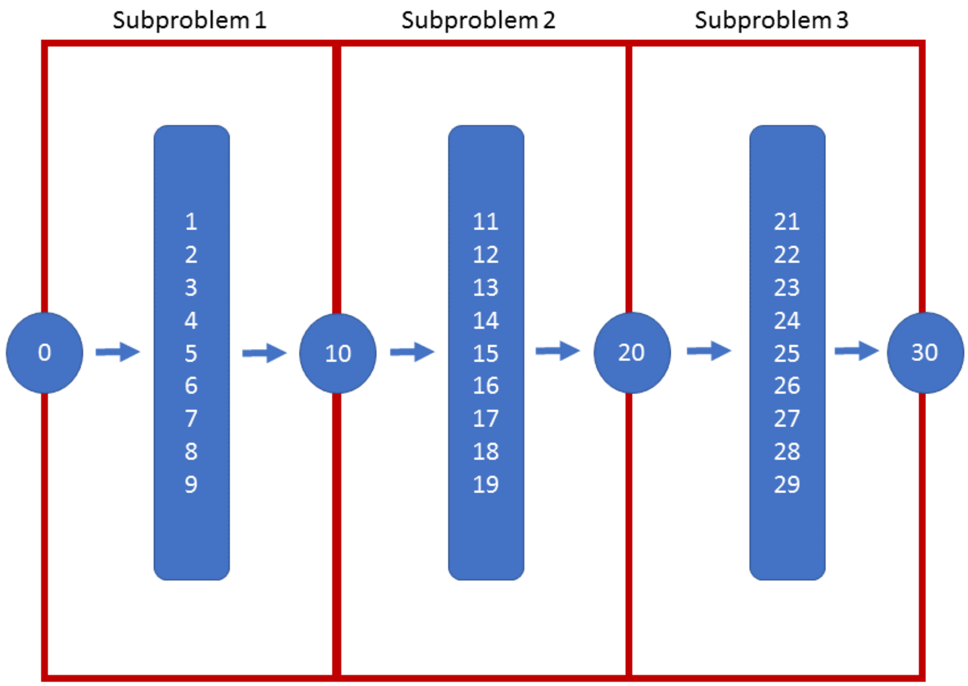


Figure 3.4: Divide and conquer heuristic with $N = 30$ and $m = 3$.

rearrange the order of nodes 1 through 9, 11 through 19, and 21 through 29. By solving m subproblems, we reduce computation times significantly.

3.6 Computational Results

All instances were created by randomly generating N locations on a 50 by 50 grid where the coordinates were distributed uniformly in each of the two dimensions. One of the N locations was randomly designated as the depot. All computations were performed on a computer with an i7-6700 processor operating at 3.4 GHz, 16 GB of RAM, and no parallelization. All computation times are reported in seconds. For DCH and the TSP-ep [2] method, an optimal standard (truck-only) TSP solution was used as input. Computation times for DCH and the TSP-ep method do not include the time required to compute the solution to the standard TSP. Instance data may be found online at `http://stefan-poikonen.net/tspd_instance_data.zip`.

In Tables 3.1 and 3.2, we see the apparent convergence of objective values when we increase TER. In Table 3.3, we compare BAB to TSP-ep and TSP-ep-all on the benchmark instances of Agatz et al. [2]. In Table 4.1, the objective values and computation times of five solution methods are reported. In Tables 3.5 and 3.6, we vary γ in the BAB+L and BAB+Q heuristics. In Table 3.7, we display a tradeoff of computation time vs. solution quality for DCH by changing the number of subproblems. In Table 3.8, we study the effect of changing R and α. In Table 3.9, we consider the choice of alternative truck and drone metrics.

3.6.1 Branch-and-Bound Results for Different Tree Exploration Ratio Values

In Table 3.1, we generated 100 random instances with $N = 10$, and solved each instance with BAB using various values of TER. By setting $TER = \infty$, we enumerate the entire branch-and-bound tree and obtain the optimal solution to each instance. In the column Obj, the average objective value over the 100 instances for a specified value of TER is given. The column Gap (Opt) is (Obj-Opt)/Opt where Opt is the average objective value of the 100 optimal solutions. The column Time gives the computation time (in seconds) required on average for a specified value of TER. In the column Optimal, we show the number of instances where the value of TER produced the optimal solution. The number of instances where TSP-ep produced a better objective value than BAB for a specified value of TER is shown in the column TSP-ep Better. In the bottom two rows, we show the average objective value, average gap from the optimal TSP-D solution, and average computation times for TSP-ep and the standard TSP. As the value of TER increases, the objective value of BAB converges. When TER reaches a value of 1.250, the objective value of BAB is less than or equal to the objective value of TSP-ep in all 100 instances. In eight of the 100 instances, TSP-ep produced the optimal solution.

In Table 3.2, we randomly generated 50 instances with $N = 15$. Computation times were intractably large for $TER = \infty$ (i.e., not a single instance was solved after several hours of testing). In the column Best Solution, we show the number of instances where the value of TER produced the lowest objective value among all solution methods that were tested. Since we do not know the optimal solution, the column Gap (Best) shows the average value of (Obj-Best)/Best where Best is the the lowest objective produced by any method tested. BAB had an objective value less than or equal to the objective value of TSP-ep in all instances with $TER > 1.025$. The objective values appear to be converging as TER increases.

In Figure 3.5, we show an example where TSP-ep fails to find the optimal solution and BAB finds the optimal solution. In this example, $\alpha = 2$ and distances between package locations are 10, except along the diagonals where the distance is $10\sqrt{2}$. The TSP-ep solution begins by launching a drone to package location 1, while the truck drives to package location 2. By the time the truck arrives at package location 2, the drone has already delivered a package at location 1 and is ready to land on the truck. The drone is launched again to package location 3, while the truck returns to package location 0. The drone will rendezvous with the truck at package location 0. The BAB solution initially launches a drone to package location 2 and sends the truck to package location 1. The truck waits for $(10+10\sqrt{2})/2-10$ time units at package location 1 for the drone to arrive. The drone is then launched to package location 3 to make a delivery and will eventually rendezvous with the truck at package location 0. The solution produced by BAB could not occur with TSP-ep, because the statement $0 < 2 < 1$ is not valid. In TSP-ep, only drone operations for triplets (i, j, k) where $i < k < j$ are considered. Thus, a drone operation beginning at package location 0 and ending at package location 1 could not launch a drone to package location 2.

In Table 3.3, we report the results for BAB, TSP-ep, and TSP-ep-all on the instances with $N = 10$ and $\alpha = 2$ that were solved in Agatz et al. [2]. There were 10 random instances of three types: uniform, 1-center, and 2-center. Uniform instances distributed package locations uniformly over a square grid. In 1-center instances, the distance of a package location from the center was distributed normally with standard deviation 50 and the angle relative to the grid was distributed uniformly over $[0, 2\pi]$. The 2-center instances were generated in the same way as 1-center instances, except that package locations were shifted horizontally by 200 with probability 0.5. In the columns labeled Opt, we report the number of instances (out of

Table 3.1: Computational results for BAB with $N = 10$.

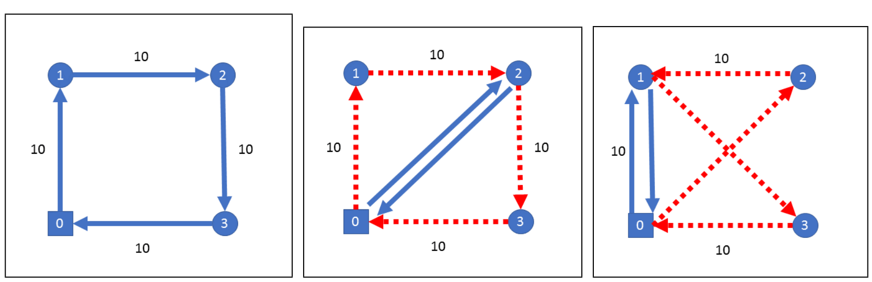
BAB					
TER	Obj	Gap (Opt)	Optimal	TSP-ep Better	Time (s)
1.000	153.243	0.032	36	11	0.031
1.025	152.400	0.027	44	9	0.041
1.050	151.254	0.019	55	6	0.064
1.075	151.048	0.017	58	5	0.104
1.100	150.614	0.014	64	5	0.164
1.125	150.067	0.011	75	3	0.239
1.150	149.757	0.009	81	3	0.331
1.175	149.335	0.006	87	2	0.449
1.200	148.914	0.003	90	1	0.594
1.225	148.823	0.002	93	1	0.794
1.250	148.673	0.001	94	0	1.013
1.275	148.661	0.001	95	0	1.281
1.300	148.518	0.000	97	0	1.568
1.325	148.517	0.000	98	0	1.989
1.350	148.517	0.000	98	0	2.443
∞	148.462	0.000	100	0	77.835
TSP-ep	159.21	0.103			0.003
TSP	186.24	0.210			0.001

10) that were solved optimally. In the columns labeled Gap, we report how much the objective value exceeded the optimal solution on average. We found that BAB performed best on 1-center instances and worst on uniform instances. One possible reason for relatively bad performance on uniform instances may be related to the fact that these are the least-clustered instances. For a specific delivery location, the set of potential launch points for the drone may be especially limited in these instances. The insertion of additional stop locations into a sequence may more frequently decrease the objective value, relative to 1-center or 2-center instances. We note that, in Section 4.2, we described how an objective value can decrease by adding stop locations.

Table 3.2: Computational results for BAB with $N = 15$.

BAB					
TER	Obj	Gap (Best)	Best Solution	TSP-ep Better	Time (s)
1.000	164.446	0.055	8	1	0.300
1.025	163.820	0.051	9	1	0.468
1.050	161.708	0.038	11	0	1.096
1.075	160.486	0.030	15	0	2.904
1.100	159.987	0.027	19	0	6.057
1.125	159.319	0.023	23	0	12.294
1.150	158.375	0.017	31	0	23.024
1.175	157.258	0.009	37	0	36.851
1.200	156.284	0.003	44	0	62.069
1.225	156.115	0.002	47	0	107.165
1.250	155.804	0.000	50	0	175.542
TSP-ep	183.821	0.180			0.003
TSP	214.309	0.376			0.001

Figure 3.5: In (a), the TSP solution has an objective value of 40. In (b), the TSP-ep solution has an objective value of $20\sqrt{2} \approx 28.28$. In (c), the BAB solution has an objective value of $10 + 10\sqrt{2} \approx 24.14$.



(a) TSP Solution (b) TSP-ep Solution (c) BAB Solution

BAB						
	Uniform Instances		1-Center Instances		2-Center Instances	
TER	Opt	Gap	Opt	Gap	Opt	Gap
1.000	2	0.055	3	0.009	5	0.013
1.025	3	0.037	5	0.007	6	0.009
1.050	5	0.024	6	0.002	7	0.006
1.075	6	0.021	9	0.001	9	0.002
1.100	6	0.016	9	0.001	9	0.002
1.125	7	0.013	10	0.000	9	0.002
1.150	8	0.010	10	0.000	9	0.002
1.175	9	0.009	10	0.000	9	0.002
1.200	9	0.003	10	0.000	9	0.002
1.225	9	0.003	10	0.000	10	0.000
1.250	9	0.003	10	0.000	10	0.000
1.275	10	0.000	10	0.000	10	0.000
1.300	10	0.000	10	0.000	10	0.000
∞	10	0.000	10	0.000	10	0.000
TSP-ep	0	0.160	0	0.152	0	0.127
TSP-ep-all	6	0.004	5	0.011	5	0.013

Table 3.3: Computational results on instances with $N = 10$ and $\alpha = 2$ from [2].

3.6.2 Solution Quality and Computation Time Results for Five TSP-D Solution Methods

In Table 4.1, we give the results for five methods and the optimal TSP solution with $R = 20$, $\alpha = 2$, and $N = 10, 20, ..., 90, 100, 200$. The truck follows the taxicab metric while the drone follows the Euclidean distance metric and travels at a speed twice as fast as the truck. Each method used the same set of 25 instances for each value of N. TER is set at 1.05 for BAB, BAB+L, BAB+Q, and all subproblems in DCH. Each row gives the average results for 25 randomly generated instances for a value of N. Obj gives the average objective value and Time (s) gives the average solution time in seconds.

	BAB		BAB+L		BAB+Q		DCH		TSP-ep		TSP
N	Obj	Time (s)	Obj	Time (s)	Obj	Time (s)	Obj	Time (s)	Obj	Time (s)	Obj
10	149.532	0.066	154.287	0.022	152.340	0.036	149.532	0.068	159.760	0.002	176.662
20	171.642	58.726	185.495	2.145	180.675	11.969	182.230	0.180	197.785	0.003	237.681
30	-	-	209.112	8.095	-	-	200.945	0.308	215.904	0.005	277.929
40	-	-	239.345	28.906	-	-	226.153	0.908	250.627	0.007	319.502
50	-	-	-	-	-	-	241.360	1.818	276.284	0.011	352.407
60	-	-	-	-	-	-	267.539	2.973	301.867	0.018	382.892
70	-	-	-	-	-	-	283.304	3.080	316.564	0.026	407.699
80	-	-	-	-	-	-	299.092	3.685	339.768	0.036	438.725
90	-	-	-	-	-	-	322.370	5.269	362.058	0.050	464.001
100	-	-	-	-	-	-	337.906	5.797	377.677	0.066	486.096
200	-	-	-	-	-	-	465.627	14.000	523.734	0.486	666.792

Table 3.4: Computation time and objective value averages for five methods and the objective value for the optimal TSP solution. A dash (-) indicates that the 25 instances could not be solved within five hours.

In BAB+L and BAB+Q, we set the parameter $\gamma = 5.0$ and $\gamma = 5.0/N$, respectively. In DCH, the number of subproblems is defined by $m = N/10$, so that the subproblem size remains constant at 10 regardless of N. BAB, BAB+L, and BAB+Q have computation times that grow quickly. In DCH, by keeping subproblem size constant at 10, computation time grows linearly with N. TSP-ep is an $O(N^3)$ method. BAB produced the best objective values, but it was the slowest method. DCH had objective values that, on average, were smaller than TSP-ep for every value of N. TSP-ep was the fastest method for every instance.

3.6.3 Linear and Quadratic Boost Heuristics Tradeoff

BAB+L and BAB+Q use the input parameter γ. In Tables 3.5 and 3.6, we show a tradeoff between objective value and computation time. We generated 25 random instances with size $N = 20$ and constant parameter values $R = 20$, $\alpha = 2$, and $TER = 1.00$. The objective values and computation times were averaged over

γ	Obj	Gap	Time (s)
0	173.56	0.000	9.493
2	177.80	0.024	1.590
4	181.60	0.046	0.571
6	184.81	0.065	0.399
8	186.74	0.076	0.271

Table 3.5: Tradeoff between solution quality and computation time for BAB+L for $N = 20$.

Nγ	Obj	Gap	Time (s)
0	173.56	0.000	9.493
2	176.67	0.017	4.770
4	178.19	0.027	3.023
6	178.74	0.030	1.426
8	179.55	0.035	0.986

Table 3.6: Tradeoff between solution quality and computation time for BAB+Q for $N = 20$.

all 25 instances. In Tables 3.5 and 3.6, larger values of γ had smaller computation times and produced worse objective values, on average. We point out that, when $\gamma = 0$ and $N\gamma = 0$, we have the same results as BAB. The column Gap in Tables 3.5 and 3.6 is computed by (Obj-BAB)/BAB, where BAB represents the objective value found by setting $\gamma = 0$.

3.6.4 Divide-and-Conquer Heuristic Tradeoff

When $m = 1$, DCH is equivalent to BAB and when $m = N$, DCH produces the truck-only TSP solution. In Table 3.7, an intermediate number of subproblems is considered where $N = 48$. We set $R = 20$, $\alpha = 2$, and $TER = 1.00$, and average the results from 25 instances. In Table 3.7, m is the number of subproblems, N/m is the average size of each subproblem, Obj is the average objective, and Time is the average computation time in seconds. There is a clear tradeoff — solving many

N/m	m	Obj	Time (s)
4	12	280.85	0.011
6	8	261.94	0.039
8	6	250.49	0.132
12	4	245.74	2.031
16	3	240.15	45.366
24	2	237.28	512.213

Table 3.7: Tradeoff between solution quality and computation time for DCH where $N = 48$.

small subproblems is computationally faster, but objective values are worse. Large values of m create a more constrained problem that is anchored at $m + 1$ points of the initial TSP solution. Anchor points are package locations that occur as either the first node or last node visited in one of the subproblems. In Figure 3.4, package locations 0, 10, 20, and 30 are anchor points. Furthermore, all nodes of group i must be visited before any nodes of group $i + 1$. In Figure 3.4, this means package locations 1 through 9 are serviced before package locations 11 through 19; package locations 11 through 19 are serviced before package locations 21 through 29. Small values of m provide more solution flexibility but suffer from slower computation times.

3.6.5 Effect of Drone Battery Duration and Speed on the TSP-D Solutions

We consider the effects of drone battery life and drone speed on the solution to the TSP-D. In Table 3.8, 25 instances were generated with $N = 48$. Each instance was solved by DCH with $R = 10, 20, 30$, $\alpha = 0.5, 1.0, 2.0, 3.0$, $TER = 1.00$, and $m = N/10$. The average TSP objective value over the 25 instances is 348.06.

Larger values of drone range and faster speeds produced smaller objective values for the TSP-D. By adding a very low performance drone, the improvement in objective value is typically very small. For example when $R = 10$ and $\alpha = 0.5$, the performance improvement was only 0.02% compared to the TSP solution. In contrast, a high performance drone ($R = 30$ and $\alpha = 3$) produced TSP-D solutions with objective values 36.89% lower than the TSP solution.

$R\alpha$ is the range of the drone in units of distance. If we compare two sets of parameters with equal values of $R\alpha$, such as $R = 30$ and $\alpha = 1.0$ versus $R = 10$ and $\alpha = 3.0$, the set of parameters with a larger value of α produced a smaller objective value in each case. This indicates that for two drones with equal range (in distance units), the drone with larger speed is usually more valuable than the drone that is capable of hovering for a long period of time to preserve feasibility of certain operations.

3.6.6 The Effect of Distance Metrics on the TSP-D Solutions

In Table 3.9, we consider the effect of different distance metrics on objective values. For each size N, 25 instances were generated and the average objective values over the 25 instances are reported.

R	α	Obj
10	0.5	347.99
10	1.0	333.57
10	2.0	286.22
10	3.0	256.84
20	0.5	345.33
20	1.0	295.69
20	2.0	240.90
20	3.0	224.51
30	0.5	337.82
30	1.0	279.44
30	2.0	232.33
30	3.0	219.67
TSP		348.06

Table 3.8: Drone battery and speed vs. TSP-D objective value for $N = 48$.

In the column TSP-D Taxi/Euc, we give the TSP-D objective value with c_t defined by the taxicab distance and c_d defined by the Euclidean distance divided by two. In the column TSP-D Euc/Euc, we give the TSP-D objective value with c_t defined by the Euclidean distance and c_d defined by the Euclidean distance divided by two (i.e., $\alpha = 2$). TSP Taxi gives the optimal objective value of the standard TSP using the taxicab distance. DCH was used for TSP-D Taxi/Euc and TSP-D Euc/Euc with $m = N/12$ and $R = 20$. Improve gives the average reduction in objective value in relative terms compared to TSP Taxi. We see that, for all instance sizes except $N = 12$, TSP-D Taxi/Euc has an average objective value that is more than 30% less than TSP Taxi. If the truck is free to move in Euclidean space, the average completion time reduction exceeds 40% except when $N = 12$.

N	TSP-D Taxi/Euc		TSP-D Euc/Euc		TSP Taxi
	Obj	Improve	Obj	Improve	Obj
12	165.02	-0.216	132.60	-0.370	210.38
24	198.76	-0.304	161.36	-0.435	285.47
36	232.93	-0.310	190.87	-0.434	337.51
48	263.17	-0.312	218.64	-0.428	382.30
60	290.71	-0.316	241.23	-0.432	425.30

Table 3.9: Comparison of TSP and TSP-D results for three different metrics.

3.7 Conclusions and Future Work

In this paper, we presented four heuristics for the TSP-D based on the branch-and-bound algorithm. For smaller instances, we showed that increasing the value of TER with BAB leads to the convergence of objective values. This suggests that BAB may generate solutions that are very close to the optimal solution when TER is sufficiently large. For larger instances, DCH produced objective values that compared favorably to TSP-ep. Although TSP-ep produced the smallest computation time in all instances, DCH had an average completion time of less than 15 seconds for the largest instances ($N = 200$). Because DCH can be solved in a reasonable amount of time on problems of practical size, DCH might be useful to drone delivery services. Additional computational experiments analyzed the effect of input parameters. We showed that when the truck was constrained to the taxicab metric, a single drone with battery life of 20 minutes and double the speed of the truck produced very significant savings, often in excess of 30%.

In future work, we hope to consider variants of the TSP-D including allowing more than one drone per truck and allowing drones to launch or land along an edge in addition to package stop locations. We want to model the overhead time required for each drone launch or landing and want to add an extra cost factor to the objective for each drone launch. We also want to consider embedding the TSP-D in a vehicle routing problem with multiple trucks. Since TSP-D produced objective values nearer to optimal on 1-center instances than on uniformly distributed instances, we want to consider the impact of customer distribution on the TSP-D and related solution methods.

Chapter 4: The Mothership and Drone Routing Problem

4.1 Introduction

The use of one or more unmanned aerial vehicles ("UAVs") in coordination with other types of vehicles has applications in private industry, military, and other government domains. [51] Amazon, Google, UPS, and DHL [5, 22, 61, 66] have all invested in programs to research the operational capabilities of drones for use in the private sector, which may include delivery of online purchases to customers. Military uses of drones range from kinetic strikes, surveillance, signal collection, transport of goods, and disaster relief. Use of drones by other government agencies may be applied to tracking criminals, monitoring traffic, emergency search-and-rescue, and monitoring wild fires.

While several previous papers have focused on truck-and-drone tandems for routing, including [2, 37, 49], this paper considers coordination of a different pair. The mothership and drone routing problem involves two vehicles:

1. The *mothership* is a large vehicle (a large ship or airplane), which is capable of moving in Euclidean space.

2. The *drone* is a smaller vehicle which is carried by the mothership, launched to

some location, then returns to the mothership for refueling or to pick-up new cargo before being launched again. The drone may be a small boat or UAV. We will generically refer to movement of the drone as flying/flight.

This mothership and drone model is fundamentally distinct from others in the literature, as the mothership operates in continuous, Euclidean space with the ability to launch or retrieve the drone at any location, rather than only at certain nodes in a graph. Potential applications of this specific model range from delivery of goods to island locations, oceanic search-and-rescue, signals collections, and military operations.

In Section 2, we present a literature review. In Section 3, we formally define the mothership and drone routing problem. In Section 4, we describe our exact solution method to the problem. In Section 5, we present computational heuristics. Section 6 contains computational results for the MDRP. Section 7 describes a model where a drone is allowed to visit multiple targets consecutively without returning to the mothership, called MDRP-HC, and associated solution methods. Section 8 provides computational results for MDRP-HC. Section 9 discusses the flexibility of our solution methods, future research, and conclusions.

4.2 Literature Review

In 2015, Murray and Chu [49] introduced the Flying Sidekick Traveling Salesman Problem (FSTSP). In FSTSP, a single drone is capable of launching from the truck with a single package, making a delivery, and returning to the truck at a rendezvous location. The truck is still capable of making deliveries while the drone is airborne, however, truck and drone must rendezvous within a fixed time limit, before the battery of the drone is depleted.

Agatz et al. [2] consider a similar problem titled the Traveling Salesman Problem with a Drone (TSP-D). A mixed integer programming formulation is given, in addition to a family of heuristics. These heuristics begin by forming a delivery sequence (either via heuristic or by solving a TSP over the customer locations), then partitioning the route into locations delivered by the truck and locations delivered by the drone. Poikonen et al. [53] adapt the partitioning procedure of Agatz et al. [2], and use it as an embedded procedure within each node of a branch-and-bound tree to produce optimal solutions. In [53], a divide-and-conquer technique is applied to break a larger master problem into a sequence of smaller subproblems to increase computational speed. Campbell et al. [19] use continuous approximation to help compute expected delivery costs. Ha et al. [37] introduce a greedy randomized adaptive search procedure (GRASP) to generate solutions to TSP-D.

In Wang et al. [65] and Poikonen et al. [54], a multi-truck, multi-drone problem titled VRPD is considered. In particular, bounds are given for the maximum possible speed-up ratio of a truck-and-drone versus truck-only model.

In Coutinho et al. [20], a different problem is considered, the close-enough traveling salesman problem (CETSP). The CETSP is a generalization of the TSP, where it is not necessary to exactly visit each customer location. Rather, it is sufficient to come “close-enough” (i.e., within a predefined radius) for each customer location. The use of second order cone programming to grade prospective sequences of visit orders is an idea we borrow from [20]. For the curious reader, the work of Lobo et al. [43] provides a brief introduction to second order cone programming, a primal-dual interior point solution method, and a list of applications where second order cone programming may be used. The authors of [43] note that second order cone programs can be solved particularly efficiently, even more efficiently than the more general class of semidefinite programs. A formal proof related to the polynomial convergence rate of primal-dual interior point methods for second order cone programs is found in the work of Monteiro and Tsuchiya [47]. The key takeaway from [43] and [47], for our paper, is that it is possible to solve many moderately large second order cone programs in a tractable amount of time.

In a paper by Savuran and Karakaya [58], a ship-and-drone routing problem is considered. In particular, an aircraft carrier is used as a mobile depot. A drone with range constraints is tasked with visiting as many targets as possible before returning to the carrier. Unlike in our work, in [58], the route of the carrier is already fixed and there is the option to not visit some targets. The primary solution method used was a genetic algorithm.

4.3 Defining the Problem

In the mothership and drone routing problem (MDRP), there exists one mothership and one drone. Both vehicles are capable of moving freely in the Euclidean plane, $\mathbb{R}^2$. We assume that there exist no obstructions to prevent mothership and drone travel from moving in straight line segments.

The mothership and the drone begin at a starting location, denoted $orig$. There exists a set of target locations T. For each $t_i \in T$, we require that the drone launch from the mothership, fly to t_i, then return to the mothership. After all targets have been visited, the mothership and drone return to a final location, denoted $dest$. In this problem, we will assume $orig$ and $dest$ are the same location. However, all results in this paper are easily extendable to the case that $orig$ and $dest$ are different locations.

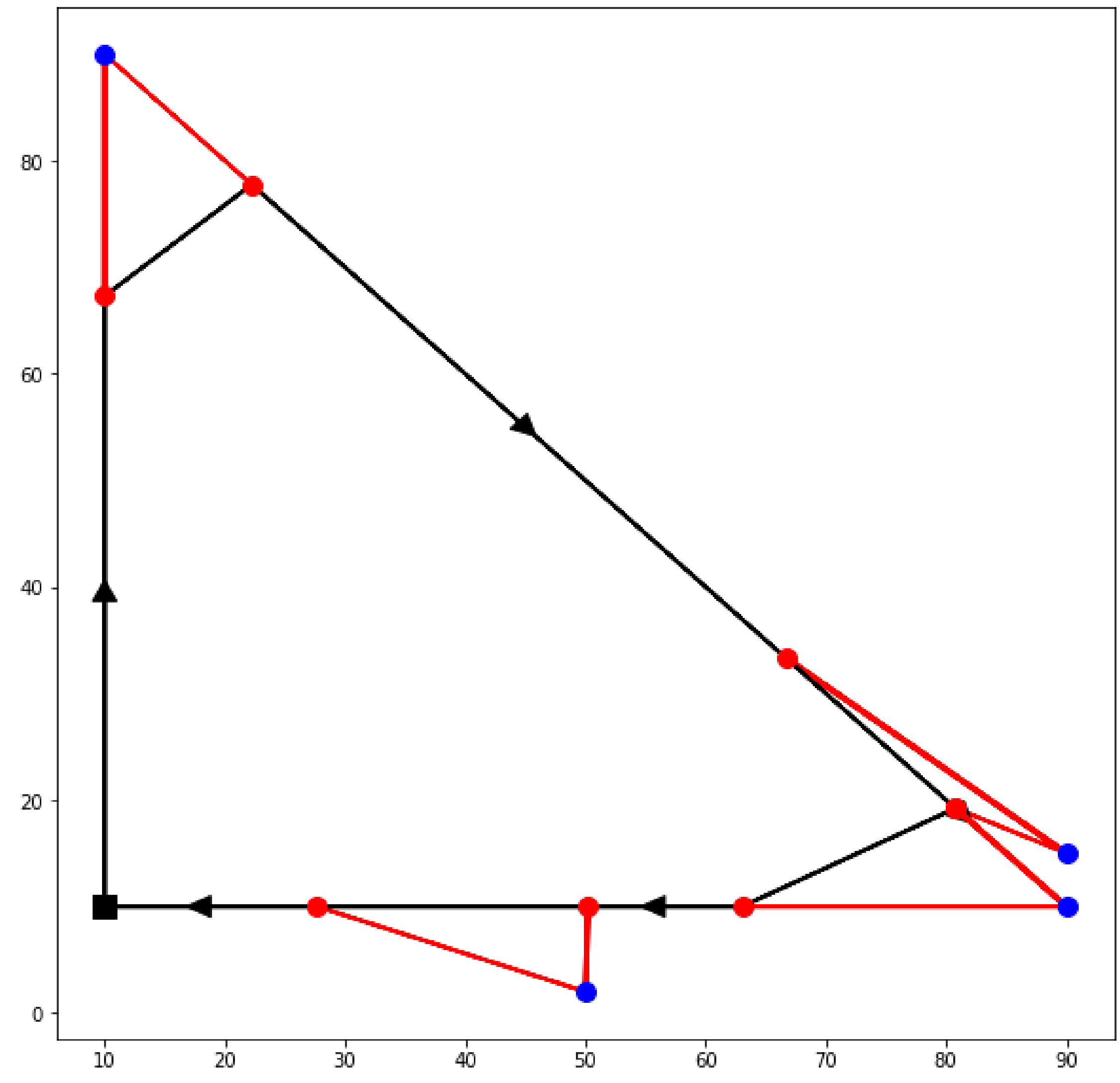
The drone may not be separated from the mothership for more than R consecutive time units. The mothership has unit maximum speed; the drone has a maximum speed of $\alpha > 1$. The drone may not visit multiple targets consecutively; it must return to the mothership after visiting a target.

The goal is to find a path of minimum duration that begins at $orig$, ends at $dest$, and where every $t_i \in T$ is visited by the drone. The MDRP is a generalization of the Euclidean Traveling Salesman Problem. In Figure 4.1, we display an example solution path for the MDRP with four targets. We point out the following result.

Theorem 9. *Let T be a set of target locations and $\{orig\}$ be the starting and terminal location. Let obj(TSP) and obj$(MDRP)$ denote the optimal objective value for the TSP and MDRP, respectively, for the set of locations $T \cup \{orig\}$. Then, obj$(TSP)/\alpha \leq$ obj$(MDRP) \leq$ obj(TSP).*

The lower bound of Theorem 9 can be shown by noting that the drone, at minimum, must travel the distance of the Euclidean TSP among the locations $T \cup \{orig\}$ at maximum speed α. The upper bound of Theorem 9 is valid, because, at

Figure 4.1: An example solution path for the MDRP with $R = 20$ and $\alpha = 2$. Black line segments trace the path of the mothership. Red line segments trace the flight path of the drone. Blue circles are target locations. Red circles are locations where the drone launches from or returns to the ship.



worst, the ship may travel to each target location $T \cup \{orig\}$ and launch the drone at negligible distance from the target.

4.4 Exact Solution Method

We may view MDRP as simultaneously answering the following two questions.

1. What is the optimal order to visit each $t_i \in T$?

2. For each $t_i \in T$, what is the optimal location to launch the drone and what is the optimal location to retrieve the drone?

Notably, the first question concerns discrete optimization, whereas, the second question concerns continuous optimization.

4.4.1 Second Order Cone Program for a Fixed Sequence

Suppose we have a fixed sequence of locations $S = [orig, s_1, s_2, ..., s_n, dest]$ with $s_1, s_2, ..., s_n \in T$. We wish to solve a subproblem that seeks to find the minimum duration closed tour, under the restrictions that: (1) the tour begins and ends at $orig = dest$, (2) each of $s_1, s_2, .., s_n$ is visited by the drone, (3) that if $i < j$, s_i is visited by the drone before s_j, (4) that the maximum speeds (1 and α) of the vehicles are not surpassed, and (5) that drone and mothership are not separated for more than R time units. Our formulation of this subproblem is labeled $\texttt{LENSEQ}(S)$.

`LENSEQ`(S):

$$minimize(\sum_{k=0}^{n+1}(cTime(k) + sTime(k)) \tag{L0}$$

`Subject to:`

`For k=0 to n:`

$$\|lPoint(k+1) - rPoint(k)\| \leq cTime(k) \tag{L1}$$

$$\|lPoint(k) - rPoint(k)\| \leq sTime(k) \tag{L2}$$

$$\|s_k - lPoint(k)\| \leq outFlightDist(k) \tag{L3}$$

$$\|s_k - rPoint(k)\| \leq inFlightDist(k) \tag{L4}$$

$$(outFlightDist(k) + inFlightDist(k))/\alpha \leq sTime(k) \tag{L5}$$

$$sTime(k) \leq R \tag{L6}$$

`End For`

$$lPoint(0) = orig \tag{L7}$$

$$rPoint(0) = orig \tag{L8}$$

$$lPoint(n+1) = dest \tag{L9}$$

$$rPoint(n+1) = dest \tag{L10}$$

In `LENSEQ`(S), for integer i such that $1 \leq i \leq n$, we use $lPoint(i)$ to represent the location at which the drone launches from the mothership before visiting target location s_i. Similarly, we use $rPoint(i)$ to represent the location where the drone is retrieved, after flying to s_i. We use $cTime(i)$ to represent the duration of time the drone rides on the mothership after returning from s_i, but before launching to s_{i+1}. We use $sTime(i)$ to represent the time elapsed starting from the launch of the drone to s_i, until the drone is retrieved by the mothership after returning from s_i.

Objective (L0) sets the duration of the tour as the sum of the times during which the mothership and drone are combined ($cTime$) and separated ($sTime$). Constraint (L1) ensures that $cTime(k)$ is at least as large as the mothership travel time from $rPoint(k)$ to $lPoint(k+1)$. Constraint (L2) ensures that $sTime(k)$ is at least as large as the travel time of the mothership from $lPoint(k)$ to $rPoint(k)$. Together, constraints (L3), (L4), and (L5) ensure that $sTime(k)$ is at least as large as the sum of the drone's flight duration from $lPoint(k)$ to s_k and the drone's flight duration from s_k to $rPoint(k)$. Constraint (L6) ensures the drone is retrieved before its maximum flight time has elapsed. Constraints (L7) through (L10) set the origin and destination of the path.

The above second order cone program may quickly solve for the optimal set of launch and landing points, relative to a fixed sequence S. We will use $lenSeq(S)$ to denote the objective value that results from applying `LENSEQ` to an input sequence S. If we consider Figure 4.1 as an example, `LENSEQ` does not choose which order the blue targets are visited; that is already fixed. However, `LENSEQ` does find optimal locations for the red circles (i.e., the launch and landing locations for the drone) and returns the objective value associated with this optimal choice of launch and landing locations.

We return to the question of finding the best sequence S.

4.4.2 Branch-and-Bound: Finding the Best Sequence

Our solution method is predicated upon the following theorem.

Theorem 10. *If S_1 is a subsequence of S_2, then $lenSeq(S_1) \leq lenSeq(S_2)$.*

Theorem 10 can be shown by observing that any feasible solution to `LENSEQ`(S_2) must also be a feasible solution to `LENSEQ`(S_1), thus $lenSeq(S_1)$ is at most $lenSeq(S_2)$.

Naïvely, we could enumerate every sequence S that begins at $orig$, visits each $t_i \in T$ (in various permutations), then returns to $dest$, and then apply the $lenSeq$ procedure to each sequence. Yet, this scales factorially and applying the $lenSeq$ procedure to each is intractable for all but the smallest of problems.

Instead, we will leverage Theorem 10. If S_1 is a subsequence of S_2, and if subsequence S_1 is not promising (i.e. $lenSeq(S_1)$ is large), then S_2 should not be highly prioritized in our search, because we know $lenSeq(S_2)$ is at least as large as $lenSeq(S_1)$.

In `ALGBAB`, we display the pseudocode for an algorithm (BAB) that searches the space of all potential visit orders to visit subsets of T with the drone. In this algorithm, we construct a branch-and bound tree, where each node is assigned a subsequence of targets and a second order cone program is solved at each node with respect to that subsequence.

In (L1) to (L7) of `ALGBAB`, we begin at the root node and associate it with a sequence $[orig, t_1, dest]$. We then set the lower bound of the root node to $lenSeq([orig, t_1, dest])$

and the upper bound of the root node to ∞. While the lower bound of the root node is less than the upper bound of the root node (L8), we iterate the following steps.

1. Find the leaf node of the branch-and-bound tree that has the smallest lower bound and call it `currNode` (L9).

2. Select the target location $t_i \in T$ that is furthest from any target that is in the sequence associated with `currNode`, i.e., `currNode.sequence`, and call it `newTarget` (L10).

3. Construct children nodes of `currNode`. The sequences associated with the children are constructed by taking the sequence of `currNode` and inserting `newTarget` into various positions (L12). For example, if `currNode.sequence` is $[orig, t_7, t_1, t_6, dest]$ and `newTarget` $= t_4$, then the sequences for the children of `currNode` are $[orig, t_4, t_7, t_1, t_6, dest]$, $[orig, t_7, t_4, t_1, t_6, dest]$, $[orig, t_7, t_1, t_4, t_6, dest]$, and $[orig, t_7, t_1, t_6, t_4, dest]$.

4. For each child node `child` with associated sequence `child.sequence`, set `child.lowerBound` $= lenSeq($`child.sequence`$)$ (L13).

5. For each child node `child` with associated sequence `child.sequence`, if each $t_i \in T$ is contained within `child.sequence` (i.e. the sequence visits all targets), then set `child.upperBound` = `child.lowerBound` (L14,L15), because this represents a feasible solution to the overall problem that visits each $t_i \in T$. Otherwise, set `child.upperBound` $= \infty$ (L16, L17), because there exists some

target $t_i \in T$ that is not visited.

6. Properly update the tree with the newly constructed children nodes and their relationship with `currNode` (L18, L19, L20, L21).

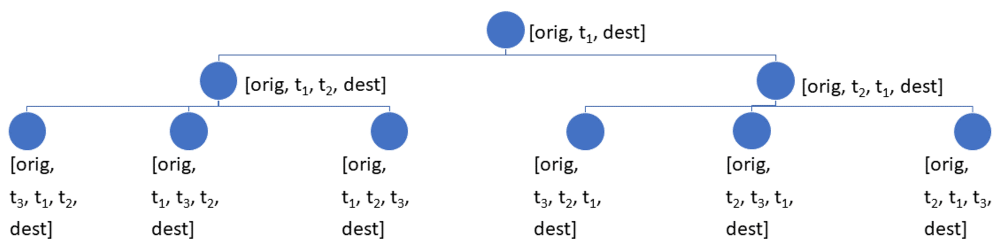
7. Mark `currNode` as no longer being a leaf node and update upper bounds and lower bounds for the ancestors of `currNode` in the tree (L22, L23, L24, L25, L26, L27, L28).

Corollary 1. *The algorithm BAB produces an optimal solution to MDRP.*

This branch-and-bound approach (BAB) is an exact approach, because Theorem 10 implies that each lower bound constructed in the branch-and-bound tree is valid and the search space of the branch-and-bound tree contains all valid visit sequences.

In Figure 4.2, we display the branch-and-bound tree for a small instance with three targets. Next to each node in Figure 4.2 is its associated sequence. The lower bound of a node with associated sequence S is initially computed as $lenTour(S)$.

Figure 4.2: A branch-and-bound tree that explores all sequences for visiting targets t_1, t_2, and t_3.



ALGBAB:

tree $= \emptyset$

rootNode.sequence $\leftarrow [orig, t_1, dest]$

rootNode.lowerBound $\leftarrow lenSeq($rootNode.sequence$)$

rootNode.upperBound $\leftarrow \infty$

rootNode.parent $\leftarrow$ none

rootNode.isLeaf $\leftarrow$ true

tree.add(rootNode)

while(rootNode.upperBound > rootNode.lowerBound):

    currNode $\leftarrow min_{node \in \texttt{tree} | \texttt{node.isLeaf=true}}$(node.lowerBound)

    newTarget $\leftarrow max_{t \in T}(min_{s \in \texttt{currNode.sequence}}(distance(s,t))$

    For position from 1 to length(currNode.sequence):

        newNode.sequence $\leftarrow insert$(currNode.sequence,newTarget,position)

        newNode.lowerBound $\leftarrow lenSeq$(newNode.sequence)

        If ($\forall t \in T, t \in$ newNode.sequence):

            newNode.upperBound $\leftarrow$ newNode.lowerBound

        Else:

            newNode.upperBound $\leftarrow \infty$

        newNode.isLeaf $\leftarrow$ true

        newNode.parent $\leftarrow$ currNode

        currNode.children.add(newNode)

4.5 Heuristics for MDRP

Although BAB is an exact solution method, for larger instances it may be intractably slow. (The computational experiments of Section 6 will confirm this.) We, therefore, propose a number of heuristic methods that are significantly faster.

4.5.1 Greedy Sequence

In the Greedy Sequence (GS) heuristic, we begin by solving the Euclidean traveling salesman problem (TSP) on the set of locations $\{orig\} \cup T$. We denote the optimal TSP path $TSPSeq = [orig, s_1, s_2, ..., s_n, dest]$. We then apply the `LENSEQ` second order cone program to input $TSPSeq$. The corresponding objective value is $lenSeq(TSPSeq)$.

4.5.2 Greedy Sequence with Local Search

In the Greedy Sequence with Local Search (GSLS) heuristic, we begin by finding $TSPSeq$ in the same way as in Greedy Sequence heuristic. Let us denote the neighborhood of an arbitrary sequence $S = [orig, s_1, s_2, .., s_n, dest]$ as $neighborhood(S)$. $neighborhood(S)$ consists of the following sequences.

1. Any sequence formed by swapping s_i and s_j, with $i \neq j$. This is called a 2-point swap.

2. Any sequence formed by selecting s_i and moving it elsewhere in the sequence (though not before $orig$ or after $dest$). This is called a 1-point swap.

3. Any sequence that results from removing a consecutive string within the sequence, $s_i, s_{i+1}, ..., s_j$ with $i < j$, and reinserting the string in reverse order. This is a 2-opt.

We then perform an iterative downhill local search. Pseudocode for this local search algorithm is labeled `ALGGSLS`.

```
ALGGSLS:

currSeq ← BestTSPSeq
contLocalSearch ← true
While(contLocalSearch=true)
    contLocalSearch ← false
    bestObjVal ← lenSeq(currSeq)
    For each neighbor in neighborhood(currSeq):
        objVal ← lenSeq(neighbor)
        If(objVal<bestObjVal):
            bestObjVal ← objVal
            bestSeq ← neighbor
    If(bestSeq ≠ currSeq):
        currSeq ← bestSeq
        contLocalSearch ← true
```

The size of a neighborhood when $|T| = n$ is $O(n^2)$ sequences. If I is the number of downhill steps in ALGGSLS, then the computational cost is $O(I * n^2) * cost(\texttt{LENSEQ})$, where $cost(\texttt{LENSEQ})$ is the computational effort required to solve LENSEQ for a single input sequence.

4.5.3 Partial Solve with Greedy Insert

In the Partial Solve with Greedy Insert (PSGI) heuristic, we let $T_p \subset T$ be a smaller subset of target locations. In Phase 1 of PSGI, we apply a slightly modified version of ALGBAB, where any references to T are replaced by T_p. We are effectively solving MDRP using BAB, but only on the subset T_p instead of T. The solution path from Phase 1 is then labeled $bestPartialSeq$.

In Phase 2, we begin with $bestPartialSeq$ and then greedily apply a form of cheapest insertion. The pseudocode for the cheapest insertion of Phase 2 is labeled

ALGPSGIPHASE2.

ALGPSGIPHASE2:

$\texttt{currSeq} \leftarrow bestPartialSeq$ $(L1)$

For each $t_i \in T \setminus T_p$: $(L2)$

    $\texttt{bestObjVal} \leftarrow \infty$ $(L3)$

    For each position = 2 to $|bestPartialSeq|$-1: $(L4)$

        $\texttt{trialSeq} \leftarrow insert(\texttt{currSeq}, t_i, \texttt{position})$ $(L5)$

        $\texttt{objVal} \leftarrow lenSeq(\texttt{trialSeq})$ $(L6)$

        If objval < bestObjVal: $(L7)$

            $\texttt{bestObjVal} \leftarrow \texttt{objVal}$ $(L8)$

            $\texttt{nextSeq} \leftarrow \texttt{trialSeq}$ $(L9)$

    $\texttt{currSeq} \leftarrow \texttt{nextSeq}$ $(L10)$

4.6 MDRP Computational Results

Code, instances, and solution data can be found at `http://stefan-poikonen.net/projects/MDRP/index.html`. All computational results were performed on a computer with an Intel i7-6700 CPU operating at 3.40GHz with 16GB of available RAM. Code was executed in Python 2.7 and Gurobi 7.5.1 was called as a solver for any second order cone programs or traveling salesman problem formulations. Any computation times reported are measured in seconds.

In the Greedy Sequence and Greedy Sequence with Local Search heuristics, finding a TSP solution is required at the beginning of the algorithm. To do so, we used a lazy constraint integer program formulation derived from [31], where violated sub-tour constraints were added as needed.

In the Partial Solve with Greedy Insert heuristic, we set $|T_p| = \lfloor 0.5 * |T| \rfloor$. That is, we initially apply `ALGBAB` on half of the targets T_p, and we greedily insert the remaining half of the targets.

4.6.1 Comparing Solution Methods for MDRP

In Table 4.1 and Table 4.2, each row displays results for the mean objective values (Obj) and computational time (Time) of 25 randomly generated instances using various solution methods. The drone flight time is fixed as $R = 20$. The ratio of drone speed to mothership speed is $\alpha = 2$.

In Table 4.1, we use a uniform distribution over a 100 by 100 grid to randomly generate the location of $orig$ and each $t_i \in T$. We refer to the instances from Table 4.1 as the uniform instances. In Table 4.2, we generate instances where $orig = (0, 0)$, and target locations are restricted to two clusters: the circle centered at $(25, 75)$ with radius 20 and the circle centered at $(75, 25)$ with radius 20. Within these two circles, the location probability density is uniform. We refer to the instances of Table 4.2

as the clustered instances.

The column $|T|$ indicates the number of targets used in each of 25 random instances for the row. For each method used to solve MPD, the column Save is calculated by (TSPObj-Obj)/TSPObj, where TSPObj is the objective value of the Euclidean TSP on $T \cup orig$.

The column under TSP corresponds to the objective value of the Euclidean TSP on the set $orig \cup T$. The columns under BAB report results for the exact solution method from Section 4; the columns under GS report results for the Greedy Sequence heuristic of Section 5.1; the columns under GSLS report results for the Greedy Sequence with Local Search heuristic of Section 5.2; the columns under PSGI report results for the Partial Solve with Greedy Insert heuristic of Section 5.3.

For BAB, we report (in column Nodes) the average number of nodes constructed in the branch-and-bound tree for each set of 25 instances.

Each dash (-) indicates that for the given instance size and solution method, the average solve time among 25 instances exceeded the timeout limit of 900 seconds.

	TSP	BAB				GS			GSLS			PSGI		
$\lvert T \rvert$	Obj	Obj	Time	Nodes	Save	Obj	Time	Save	Obj	Time	Save	Obj	Time	Save
10	289.129	213.744	1.543	266.520	0.261	214.538	0.009	0.258	214.403	2.152	0.258	215.143	0.303	0.256
15	346.392	240.736	18.802	2246.520	0.305	245.200	0.014	0.292	243.603	6.813	0.297	248.744	1.017	0.282
20	377.677	252.232	700.220	61 640.280	0.332	260.784	0.024	0.310	258.523	17.280	0.315	263.646	3.330	0.302
30	455.334	-	-	-	-	302.818	0.048	0.335	301.279	55.342	0.338	299.914	35.999	0.341
50	567.317	-	-	-	-	371.101	0.157	0.346	369.223	293.981	0.349	-	-	-
100	779.824	-	-	-	-	511.596	1.331	0.344	-	-	-	-	-	-
200	1072.641	-	-	-	-	698.989	16.798	0.348	-	-	-	-	-	-

Table 4.1: Computational results for the MDRP on uniformly distributed instances.

	TSP	BAB				GS			GSLS			PSGI		
$\lvert T \rvert$	Obj	Obj	Time	Nodes	Save	Obj	Time	Save	Obj	Time	Save	Obj	Time	Save
10	278.753	242.106	12.585	1906.080	0.131	245.518	0.026	0.119	244.982	2.335	0.121	243.187	0.365	0.127
15	295.646	252.732	559.557	63 334.320	0.145	258.422	0.037	0.126	257.782	7.115	0.128	254.932	1.530	0.137
20	308.035	-	-	-	-	265.013	0.065	0.139	264.707	16.690	0.140	264.356	14.495	0.141
30	328.606	-	-	-	-	277.630	0.163	0.155	301.279	55.342	0.157	-	-	-
50	369.201	-	-	-	-	298.854	0.341	0.190	298.142	229.996	0.192	-	-	-
100	779.824	-	-	-	-	511.596	1.331	0.344	-	-	-	-	-	-
200	436.448	-	-	-	-	342.172	4.061	0.216	-	-	-	-	-	-

Table 4.2: Computational results for the MDRP on clustered instances.

4.6.2 Analysis of MDRP Computational Results

In Table 4.1 and Table 4.2, the computational time of BAB rapidly increases with instance size. This correlates strongly with the average number of nodes explored in the branch-and-bound tree. Moreover, the clustered instances of Table 4.2 were computationally more costly than the uniform instances of Table 4.1. In the clustered instances, swapping the order of two targets within the same cluster usually produces similar objective values. This symmetry causes slower convergence of the branch-and-bound algorithm.

The GS heuristic is the fastest heuristic tested. For instances where the optimal MDRP solutions were found, the worst performance was on the uniform instances of size $|T| = 20$. In this row of instances, GS cut 31.0% percent from the optimal TSP solution, whereas the optimal MDRP solution cut 33.2% from the optimal TSP solution. The vast majority of computation time for GS on moderate and large size instances was spent solving for an optimal TSP. Using a faster TSP procedure (for example the Lin-Kernighan Heuristic [42]) could reduce this significantly. In a randomly generated set of 25 uniform instances of size $|T| = 200$, we found that the computation time of the GS algorithm, aside from computing the TSP, averaged only 0.360 seconds. For $|T| = 20$, we found that the computation time of the GS algorithm, aside from computing the optimal TSP, averaged only 0.020 seconds.

The GSLS heuristic showed marginal impact in reducing the objective value compared to the GS heuristic. In the best case (uniform instances of size $|T| = 20$), GSLS reduced the objective value (relative to the GS algorithm) only by 0.9%. The Euclidean distances used by both mothership and drone may imply that local searches are unlikely to produce significant improvements.

For all sets of clustered instances, PSGI was the best performing heuristic. However, for uniform instances, PSGI was sometimes outperformed by GS and GSLS. The computation time of PSGI quickly grows as $|T_p|$ grows. This is similar to the computation time growth of BAB as $|T|$ increases.

4.7 The Mothership and High Capacity Drone Routing Problem

A fundamental assumption of the mothership and drone routing problem is that the drone must return to the mothership following each target visit. However, in some applications, it may be possible for the drone to launch from the mothership, visit one or more targets consecutively, then return to the mothership. We define the mothership and high capacity drone routing problem (MDRP-HC) in the same way as MDRP, except we now allow the drone to visit multiple targets consecutively before returning to the mothership. We continue to require that the drone must not be separated from the mothership for more than R consecutive time units.

Theorem 11. *If $R = \infty$, the solution of the MDRP-HC will have the drone visit* *all targets consecutively before returning to the mothership at dest. Moreover, the solution is equivalent to the Euclidean TSP, where the speed of travel is* α*.*

4.7.1 Concepts: Drone Subtours and Compositions

We define a *drone subtour*, $ST_i = (st_{i_1}, st_{i_2}, ..., st_{i_z})$, as an ordered set of target locations, with $st_{i_1}, st_{i_2}, ..., st_{i_z} \in T$, that are visited consecutively by the drone without the drone returning to the mothership in between. If $j < k$, then st_{i_j} is visited by the drone before st_{i_k}.

In Figure 4.3, we display an example solution for the MDRP-HC, which contains three drones subtours. The subtours contain two, three, and two targets and are indicated by red line segments.

Let $S = [orig, s_1, s_2, ..., s_n, dest]$ be a potential order for visiting targets $s_1, s_2, .., s_n \in T$. Let $compositions(S)$ be the set of ways that we can group $[s_1, s_2, ..., s_n]$ into separate drone subtours, while preserving the feature that if $i < j$, then s_i is visited by drone before s_2. For example, suppose $S = [orig, t_4, t_2, t_3, t_1, dest]$. Then $compositions(S) =$

$$\{[orig, (t_4, t_2, t_3, t_1), dest], [orig, (t_4, t_2, t_3), (t_1), dest], [orig, (t_4, t_2), (t_3, t_1), dest],$$
$$[orig, (t_4), (t_2, t_3, t_1), dest], [orig, (t_4, t_2), (t_3), (t_1), dest], [orig, (t_4), (t_2, t_3), (t_1), dest],$$
$$[orig, (t_4), (t_2), (t_3, t_1), dest], [orig, (t_4), (t_2), (t_3), (t_1), dest]\}.$$

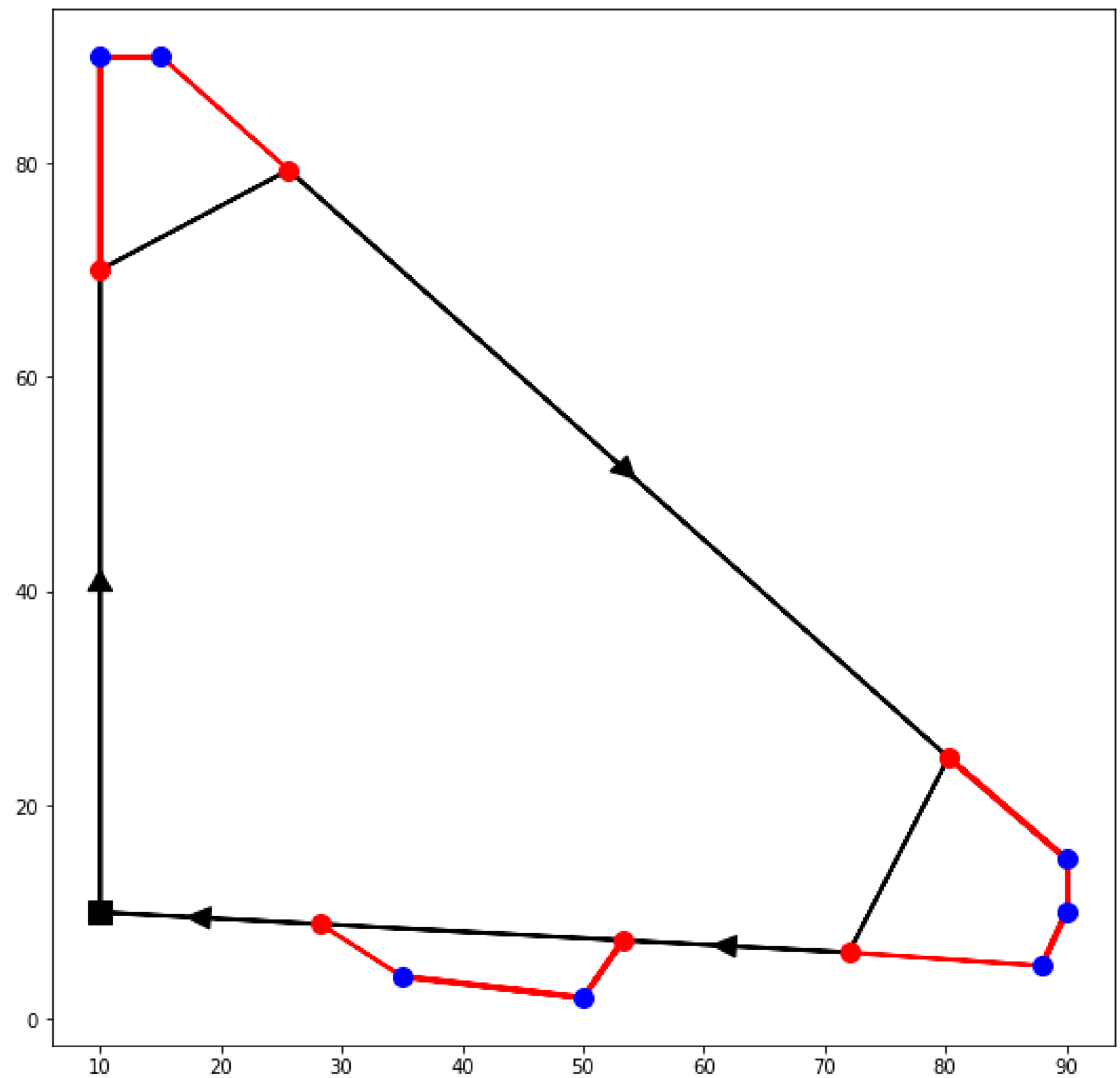


Figure 4.3: An example solution path for the MDRP-HC with $R = 20$ and $\alpha = 2$. Black line segments trace the path of the mothership. Red line segments trace the flight path of the drone. The black square is location $orig = dest$. Blue circles are target locations. Red circles are locations where the drone launches from or returns to the ship. By applying the `LENCOMP` function, we are optimally choosing locations for the red circles.

The first composition contains (t_4, t_2, t_3, t_1). This means that the drone would launch from the mothership, visit t_4, t_2, t_3, t_1 consecutively, then return to the mothership. In the second composition, the drone would launch to visit t_4, t_2, t_3 consecutively before returning to the mothership. Afterwards, the drone launches to visit t_1, as a second separate subtour.

4.7.2 Second Order Cone Program for a Fixed Composition

Suppose a composition $C = [orig, ST_1, ST_2, ..., ST_m, dest]$ is fixed, where m is the number of distinct drone subtours within the composition.

If $ST_i = (st_{i_1}, st_{i_2}, ..., st_{i_z})$, then define $len(ST_i) = \sum_{j=1}^{z-1} \|st_{i_{j+1}} - st_{i_j}\|$, which represents the flight distance of the drone within the drone subtour. In Figure 4.3, for example, $len(ST_2)$ is the sum of the distance from the third target location to the fourth target location and the distance from the fourth target location target to the fifth target location.

Let $launch(ST_i)$ denote the location where the drone launches from the mothership immediately prior to visiting the first target of ST_i. Likewise, let $land(ST_i)$ denote the location where the drone will land on the mothership, after visiting the last target of ST_i. These are represented by red circles in Figure 4.3. Let $first(ST_i) = st_{i_1}$ denote the first target location within ST_i. Let $last(ST_i) = st_{i_z}$ denote the last target location within ST_i.

For composition C with drone subtours $ST_1, ST_2, ..., ST_m$, we would like to optimally choose $launch(ST_i)$ and $land(ST_i)$ for $i = 1, 2, ...m$ to minimize completion time. To do so, we apply the pseudocode labeled `LENCOMP`. To call `LENCOMP` for a composition C, we denote this $lenComp(C)$.

`LENCOMP(C):`

`Set` $len(ST_0) = 0, first(ST_0) = depot, last(ST_0) = depot$

`For k=1 to m:`

`Precompute constant` $len(ST_k) = \sum_{j=1}^{z-1} \|st_{k_{j+1}} - st_{k_j}\|$

$$minimize(\sum_{k=0}^{n+1}(cTime(k) + sTime(k)) \quad (L0)$$

`Subject to:`

`For k=0 to m:`

$$\|lPoint(k+1) - rPoint(k)\| \leq cTime(k) \quad (L1)$$

$$\|lPoint(k) - rPoint(k)\| \leq sTime(k) \quad (L2)$$

$$\|first(ST_k) - lPoint(k)\| \leq outFlightDist(k) \quad (L3)$$

$$\|last(ST_k) - rPoint(k)\| \leq inFlightDist(k) \quad (L4)$$

$$len(ST_k) \leq intraFlightDist(k) \quad (L5)$$

$$(outFlightDist(k) + intraFlightDist(k)+$$

$$inFlightDist(k))/\alpha \leq sTime(k) \quad (L6)$$

$$sTime(k) \leq R \quad (L7)$$

`End For`

$$lPoint(0) = orig \quad (L8)$$

$$rPoint(0) = orig \quad (L9)$$

$$lPoint(m+1) = dest \tag{L10}$$

$$rPoint(m+1) = dest \tag{L11}$$

We note that if $len(ST_i) > R\alpha$ for any $i = 1, 2, ..., m$, then the composition C is infeasible, as it is impossible to satisfy constraints (L5), (L6), and (L7) simultaneously. This aligns with the constraint that the drone must return to the mothership within R time units. We also note that the number of decision variables in `LENCOMP` is no more than the number of decision variables in `LENSEQ`, because $m \leq n$ (i.e., the number of drone subtours is no more than the number of targets).

4.7.3 Finding the Best Composition

Section 7.2 describes describes how to optimize MDRP-HC for a fixed composition C. However, we must address the question: "Which composition C is best?" We propose a number of methods to select high quality compositions.

To find a high quality composition, there are two steps. First, determine a sequence that describes which order the targets will be visited. Second, find a composition that efficiently groups consecutive targets of the sequence into drone subtours.

4.7.3.1 Branch-and-bound: An Exact Approach

We may use a branch-and-bound scheme that has a broad structure similar to the BAB method for MDRP in Section 4.2. This method, denoted BAB-C, uses a tree similar to Figure 4.2 to search the space of potential sequences. Each node of this branch-and-bound tree corresponds with some sequence S.

The difference compared to BAB, however, is that for a node associated with the sequence S, the lower bound of the node is not set to $lenSeq(S)$. Instead, the lower bound of a node associated with sequence S is $min_{C \in compositions(S)}(lenComp(C))$.

Brute forcing all compositions C of a sequence S is costly: $O(2^{n-1})$, where n is the number of targets visited in sequence S. Therefore, we describe a more efficient procedure in Appendix A for finding the best composition C with respect to a sequence S.

We then apply branch-and-bound until convergence of the upper bound lower bound of the root node. We then return the best composition of the leaf node with the lowest lower bound as our solution.

4.7.3.2 Greedy Sequence Exact Composition Heuristic

In the Greedy Sequence Exact Composition heuristic (GSEC), we choose a sequence, S, as the solution of the Euclidean TSP on $\{orig\} \cup T$. This sequence S

determines which order we will visit each of the targets.

The next questions is, what is the best composition of S? We find the best composition C of the sequence S, using the method described in Appendix A.

We call this method Greedy Sequence, Exact Composition because the sequence S is not necessarily the best visit order. However, the composition C is the best composition with respect to delivery order S.

4.7.3.3 Greedy Sequence and Greedy Composition

In the Greedy Sequence and Greedy Composition heuristic (GSGC), we greedily fix a sequence S as the solution of the Euclidean TSP on $\{orig\} \cup T$. Let us write $S = [orig, s_1, s_2, ..., s_{|T|}, dest]$.

Next, we use a greedy procedure to determine a composition C for S. For the first drone subtour, we set $ST_1 = (s_1, s_2, ..., s_{y_1})$, where y_1 is the maximum integer such that $len(ST_1) \leq R\alpha$ and $\|s_1 - s_{y_1}\| < R$. Then for the second drone subtour, we set $ST_2 = (s_{y_1+1}, s_{y_1+2}, ..., s_{y_2})$, where y_2 is the maximum integer such that $len(ST_2) \leq R\alpha$ and $\|s_{y_1+1} - s_{y_2}\| < R$. In general, we set $ST_{j+1} = (s_{y_j+1}, s_{y_j+2}, ..., s_{y_{j+1}})$, where y_{j+1} is the maximum integer such that $len(ST_{j+1}) \leq R\alpha$, $y_{j+1} \leq |T|$, and $\|s_{y_j+1} - s_{y_{j+1}}\| < R$. We then define our compositions by $C = [orig, ST_1, ST_2, ..., ST_m, dest]$. We then compute $lenComp(C)$.

To put it another way, we pack as many targets as possible into the first drone subtour without violating the range constraints of the drone. We then pack as many targets as possible into the second drone subtour without violating the range constraint of the drone and so on.

4.7.3.4 Greedy Sequence and Greedy Composition with Slack

In the GSGC heuristic, if we maximally fill a drone subtour with targets, this may leave the drone with very little slack range to fly to the first target of the drone subtour and to return to the ship after the last target of the drone subtour. This, at times, has the effect of strictly constraining the feasible launch and landing locations for each drone subtour.

The Greedy Sequence and Greedy Composition with Slack heuristic (GSGC+S) is nearly identical as GSGC. However, we set $ST_{j+1} = (s_{y_j+1}, s_{y_j+2}, ..., s_{y_{j+1}})$, where y_{j+1} is the maximum integer such that $len(ST_{j+1}) \leq (1 - slackFactor) * R\alpha$, $y_{j+1} \leq |T|$, and $\|s_{y_j+1} - s_{y_{j+1}}\| < (1 - slackFactor) * R$, where $0 < slackFactor < 1$. The idea is that $slackFactor$ ensures that we do not maximally fill each drone subtour, which guarantees more freedom in choosing the launch and landing locations for each drone subtour.

4.8 MDRP-HC Computational Results

Computational results for algorithms related to MDRP-HC are found in Table 4.3 and Table 4.4. In Table 4.3, instances are generated by randomly selecting $orig$ and the target set over a uniform distribution on grid of size 100 by 100. In Table 4.4, for each instance size, $|T|$, we generated 25 random instances, where the $orig = (0, 0)$ and target locations were randomly distributed among the circles with radius 20 centered at (25,75) and (75,25).

In both Table 4.3 and Table 4.4, $R = 20$ and $\alpha = 2$ are fixed. The columns under BAB-C correspond with the BAB-C solution method; the columns under GSEC correspond with the GSEC solution method columns under GSGC correspond with the GSGC solution method; and columns under GSGC+S correspond with the GSGC+S solution method. In the GSGC+S heuristic, we fixed $slackFactor = 0.2$ based on preliminary testing. Columns titled Obj, Time, Nodes, and Save correspond to the average objective value, computational time (seconds), nodes explored in the branch-and-bound tree, and savings relative to the Euclidean solution, respectively. Dashes indicate an average solve time exceeding 900 seconds.

	TSP	BAB-C				GSEC			GSGC			GSGC+S		
$\|T\|$	Obj	Obj	Time	Nodes	Save	Obj	Time	Save	Obj	Time	Save	Obj	Time	Save
6	247.206	180.212	1.619	30.040	0.271	180.621	0.112	0.269	195.918	0.010	0.207	186.590	0.009	0.245
8	276.015	199.690	17.362	121.160	0.277	200.222	0.250	0.275	217.640	0.010	0.215	208.126	0.009	0.246
10	292.055	201.632	35.987	175.960	0.310	203.125	0.053	0.304	229.211	0.011	0.215	214.035	0.011	0.267
15	342.824	-	-	-	-	230.748	16.268	0.327	265.390	0.015	0.226	246.953	0.014	0.280
20	396.569	-	-	-	-	254.273	234.042	0.359	296.117	0.018	0.253	273.921	0.018	0.309
30	462.943	-	-	-	-	-	-	-	339.226	0.041	0.267	306.920	0.042	0.337
50	573.894	-	-	-	-	-	-	-	402.533	0.111	0.299	360.185	0.113	0.372
100	785.445	-	-	-	-	-	-	-	508.073	3.034	0.353	454.771	3.048	0.421
200	1065.807	-	-	-	-	-	-	-	649.942	35.210	0.390	582.593	36.127	0.453

Table 4.3: Computational results for MDRP-HC on uniformly distributed instances.

	TSP	BAB-C				GSEC			GSGC			GSGC+S		
$\|T\|$	Obj	Obj	Time	Nodes	Save	Obj	Time	Save	Obj	Time	Save	Obj	Time	Save
6	260.745	216.809	2.462	39.160	0.169	217.177	0.106	0.167	222.682	0.006	0.146	222.199	0.006	0.148
8	270.946	223.620	22.477	129.000	0.175	224.023	0.374	0.173	233.241	0.007	0.139	231.105	0.007	0.147
10	283.369	233.331	208.214	497.440	0.177	233.866	1.033	0.175	244.009	0.009	0.139	242.034	0.008	0.146
15	296.968	-	-	-	-	238.400	12.690	0.197	253.606	0.121	0.146	248.574	0.119	0.163
20	311.461	-	-	-	-	245.267	291.369	0.213	261.503	0.019	0.160	255.025	0.018	0.182
30	331.784	-	-	-	-	-	-	-	271.077	0.051	0.183	262.370	0.051	0.209
50	370.613	-	-	-	-	-	-	-	286.639	0.210	0.227	276.490	0.210	0.254
100	441.263	-	-	-	-	-	-	-	319.290	5.133	0.276	303.628	5.125	0.312
200	537.363	-	-	-	-	-	-	-	364.963	59.861	0.321	349.491	60.115	0.350

Table 4.4: Computational results for MDRP-HC on clustered instances.

4.8.1 Analysis of MDRP-HC Computational Results

The exact BAB-C algorithm exhibits large computational time growth similar to BAB. The GSEC heuristic produces objective values that are very near optimal. In the worst row of instances, BAB-C saves 31.0% relative to the TSP, whereas GSEC saves 30.4%. This indicates that the Euclidean TSP initialization is very reasonable for MDRP-HC. Nonetheless, computational tractability for GSEC becomes an issue for larger instances.

The GSGC and GSGC+S heuristics were very fast. On the slowest set of instances ($|T| = 200$, clustered), the average time spent by these heuristics, aside from solving the TSP as an initialization, was only 0.21 seconds. This is even faster than the GS method for MDRP, because the second order cone program `LENCOMP` only needs to solve for one launch and one landing location for each drone subtour, instead of solving for one launch and one landing location for each target location, thus reducing the number of decision variables. On average, the GSGC+S heuristic produced higher quality solutions than GSGC, at similar computational cost. More finely tuning $slackFactor$ may produce better results.

4.9 Variants, Conclusions, and Future Work

4.9.1 Variants

One of the key features of our proposed solution methods is the flexibility to accommodate different objectives and/or constraints. We point the reader to Appendix B and Appendix C for variant problems to MDRP and MDRP-HC that can be solved by minorly modifying our proposed solution method. These modifications generally involve altering only a few lines of a second order cone program. Variants presented include a *close-enough* version of the problem with application to signal collection, weight constraints, energy constraints, and minimizing the sum of waiting times.

4.9.2 Conclusions

We introduced the mothership and drone routing problem (MDRP). The problem is distinct from other papers in the literature, as the launching vehicle (i.e. "the mothership") is capable of moving in continuous space. This allows second order cone programs to be used throughout as subroutines in solution methods.

Our BAB method is an exact approach to solve MDRP that works well for small instances. However, scalability is an issue, so we introduced heuristic methods. Aside from the time required to solve a single TSP to initialize the algorithm, the computational time for the GS heuristic was small, averaging only 0.360 seconds for instances with 200 target locations. For instance sets for which we have the exact solution, the worst performance of the GS heuristic on any row of instances produced objective values that averaged 3.39% greater than optimal; on the best row of instances, GS was only 0.37% suboptimal. We believe the GS heuristic is a promising solution method for large instances. Two other heuristics provided marginal improvement in objective quality relative to GS, but require more computational time.

We also introduced the Mothership and High Capacity Drone Problem (MDRP-HC), where a drone may visit multiple targets consecutively without returning to the ship. We proposed both exact and heuristic methods to MDRP-HC. The exact solution method was slow. However, the GSEC solution method, the GSGC heuristic, and the GSGC+S methods provide faster solutions methods. GSEC objective values quite near the optimal solution, however, it also runs into computational tractability issues on larger instances. GSGC+S produced better results than GSGC, indicating that by not filling every drone subtour to capacity, we not only expand the set of feasible launch and landing locations, but this expanded choice produces better objective values. Further tuning of the parameter $slackFactor$ and adding some form of local search to GSGC+S may bring objective values even closer to optimal.

4.9.3 Future Work

There are a number of future directions that we believe merit consideration. In this paper, we assumed the mothership is capable of traveling by the Euclidean metric. If the mothership is an airplane or a ship in the open seas with little to no dry land, this may be a reasonable assumption. However, in an operational context where the mothership is a sea vessel that is operating in a region with significant areas of dry land, shallow waters, hostile actors, or political boundaries, the mothership may not be able to always traverse straight line segments without accounting for these obstructions. In a subsequent paper, we will describe how to account for this. These obstructions inject non-convexity into the problem, which requires a significant restructuring of our solution methods.

We are also interested to explore whether some ideas from this paper may carry over to a truck-and-drone context. Another natural question to consider is this: how could we best route a mothership that may launch more than one drone to visit targets?

4.10 Insert A: Computing the best composition for a given input sequence

For any sequence S containing n targets, there are 2^{n-1} compositions of S. This is equal to the number of order-dependent integer partitions possible for a positive integer n. Thus, computing $lenComp(C)$ for each $C \in compositions(S)$ is costly and should be avoided.

To compute the best composition C for a given input sequence S, we construct a binary branch-and-bound tree. The root node is associated with the composition $[orig, (s_1), dest]$. Each time we we descend a level in the tree, we add the next target of the sequence into the child nodes. The left branch merges the target into the preceding drone subtour. The right branch adds the target as a new drone subtour. For a node that is associated with composition C, the lower bound is computed as $lenComp(C)$. The upper bound of a node with associated composition C is ∞, unless C contains all targets that are in sequence S, in which case the upper bound of the node is the same as the lower bound.

The branch-and-bound tree for an example sequence $S = [dest, s_1, s_2, s_3, orig]$ is shown in Figure 4.4.

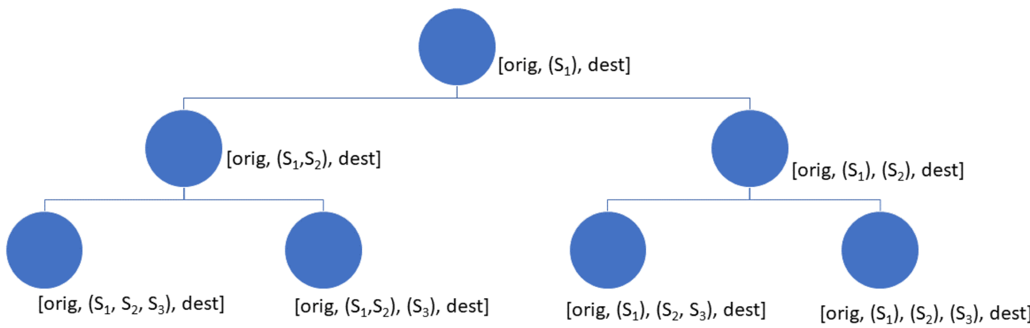


Figure 4.4: A binary branch-and-bound tree that explores all compositions for the fixed sequence $S = [orig, s_1, s_2, s_3, dest]$. Each left branch appends the new target into the preceding drone subtour. Each right branch appends the new target as a new drone subtour. Next to each node in the figure is the associated composition.

4.11 Insert B: Variants of MDRP

A key feature of the solution methods that we have proposed is that they are extendable to variant problems of MDRP. In particular, we are able to modify the constraints and/or objective of the second order cone program `LENSEQ` to fit the specifications of variant problems, so long as we preserve the form of a second order cone program, or more broadly, a semidefinite program. Additionally, if we modify `LENSEQ` for a variant problem and Theorem 10 continues to hold, then applying the solution method `ALGBAB` is optimal for the variant problem. We give a few examples of how MDRP may be modified to fit variant problems.

Penalize Flight Time

In `LENSEQ`, we minimize the total duration of the solution path. Suppose that, instead, we are interested in minimizing the sum of the total duration of the solution path and a scalar multiple ($\gamma > 0$) of drone flight time to penalize fuel expenditure of the drone. We may accomplish this by replacing (L0) of `LENSEQ` with the following objective:

$$minimize(\sum_{k=0}^{n+1}(cTime(k) + (1+\gamma) * sTime(k))$$

Minimizing the Sum of Waiting Times

Suppose we wish to minimize the sum of waiting times of all targets $t_i \in T$, where the waiting time of a target t_i is defined as the time elapsed starting from the departure of the mothership and/or drone from *orig* until the drone arrives at t_i. To do so we make two modifications to `LENSEQ`. First, we add the following set of constraints.

```
For k=1 to n:
```

$$\sum_{i=0}^{k-1}(sTime(i) + cTime(i)) + outDroneDist(k)/\alpha \leq arrivalTime(k)$$

Second, we change the objective (L0) to the following.

$$minimize(\sum_{k=1}^{n}(arrivalTime(k))$$

The *Close-Enough* Variant

Suppose that for each target $t_i \in T$ it is sufficient that a drone pass within distance $rad_i \geq 0$, rather than needing to visit the exact location of t_i. This may be relevant for an application where me must collect a signal or establish a line-of-sight with each target. To model this problem, we replace `LENSEQ` with `LENSIGNALTOUR`.

`LENSIGNALTOUR:`

$$minimize(\sum_{k=0}^{n+1}(cTime(k) + sTime(k)) \tag{L0}$$

`Subject to:`

`For k=0 to n:`

$$\|lPoint(k+1) - rPoint(k)\| \leq cTime(k) \tag{L1}$$

$$\|lPoint(k) - rPoint(k)\| \leq sTime(k) \tag{L2}$$

$$(outFlightDist(k) + inFlightDist(k))/\alpha \leq sTime(k) \tag{L3}$$

$$sTime(k) \leq R \tag{L4}$$

`End For`

$$lPoint(0) = orig \tag{L5}$$

$$rPoint(0) = orig \tag{L6}$$

$$lPoint(n+1) = dest \tag{L7}$$

$$rPoint(n+1) = dest \tag{L8}$$

`For k=1 to n:`

$$\|readPoint_k - lPoint(k)\| \leq outFlightDist(k) \tag{L9}$$

$$\|readPoint_k - rPoint(k)\| \leq inFlightDist(k) \tag{L10}$$

$$\|s_k - readPoint_k\| \leq rad_i \tag{L11}$$

`End For`

The decision variable $readPoint_k$ represents a location within a distance of rad_k of s_k that the drone will visit. We may think of this as the designated signal reading location for target s_k.

Enforced Minimum Refuel Time

There may exist a minimum waiting period after a drone returns to the mothership from one target before it is ready to be redeployed. In the simplistic case where this minimum waiting is a fixed constant $minWait$, we can model this by adding the following constraints to `LENSEQ`.

```
For k=1 to n-1:
```

$$minWait \leq cTime(k)$$

Alternatively, the minimum waiting period before relaunching may scale linearly with the battery or fuel drained from the preceding flight (i.e. recharging or refueling may occur at a linear rate). Suppose for each unit of drone flight time, we must recharge for δ time units before launching to the next target, in order to replace expended fuel. In such a case, we could add the following set of constraints.

```
For k=1 to n-1:
```

$$\delta * sTime(k) \leq cTime(k)$$

Enforcing Maximum Energy Expenditure

In some contexts, a drone may be tasked to deliver a payload to a target. The weight of the payload to be delivered to target t_i is w_i. Let $e(w)$ be the rate of energy drain per unit distance when the drone is carrying a payload of weight w. Let E be the maximum energy a drone may expend before returning to the ship. This scenario may be modeled by adding the following set of constraints to the second order cone program `LENSEQ`.

```
For k=1 to n:
```

$$e(w_k) * outFlightDist(k) + e(0) * inFlightDist(k) \leq E$$

4.12 Insert C: Variants of MDRP-HC

We may incorporate additional constraints or features into the MDRP-HC model by altering the second order cone program `LENCOMP`. After modifying `LENCOMP`, we can otherwise apply BAB-C or GSEC as normal. We give a few examples of additional constraints or features that may be added to MDRP-HC.

Constraining Maximum Delivery Weight in Drone Subtour

In the context of delivery, we may wish to set a maximum weight W for the sum of package weights carried by the drone at any one time. It is fairly straightforward to extend the solution methods of MDRP-HC to this case. If the weight of the package delivered to t_i is w_i, then we simply disallow any composition that contains a drone subtour ST_x such that $\sum_{t_i \in ST_x} w_i > W$. To do so, we may simply add the following constraints to the second order cone program of `LENCOMP`.

$$\texttt{For each } ST_x \in C :$$

$$\sum_{t_i \in ST_x} (w_i) \leq W$$

If any drone subtour in the composition violates the maximum weight requirement, the second order cone program will be infeasible, due to the above constraint, and return ∞.

Constraining Maximum Delivery Energy in Drone Subtour

Similar adaptations can be made to constrain the maximum energy expenditure of a drone in a single drone subtour. In practice, the battery life of UAVs is frequently a pressing constraint that should be considered.

Suppose E is the maximum energy expenditure for a single drone subtour. Define $e(w)$ as the rate of energy expenditure per unit distance, whenever the sum of all package weights carried by the drone is w. Also, we use w_{i_j} to denote the weight of the package delivered to st_{i_j}.

To incorporate the maximum energy expenditure E for a drone subtour, we do the following. For each drone subtour $ST_k \in C$, we precompute the constant $weightTotal(ST_k) = \sum_{t_i \in ST_k} w_i$. Next, for each drone subtour $ST_k \in C$, we precompute the constant:

$$intraTourEnergyUsed(ST_k) = \sum_{j=2}^{|ST_k|} (\|st_{k_j} - st_{k_{j-1}}\| * e(weightTotal - \sum_{l=1}^{j-1}(w_{k_l}))),$$

which is the amount of energy expended by the drone from the arrival at the $first(ST_k)$ until arrival at $last(ST_k)$.

Next, we add the following set of constraints to the second order cone program of `LENCOMP`.

$$\begin{aligned}
&\texttt{For each } ST_k \in C: \\
&\quad e(weightTotal(ST_k)) * outFlightDist(k) + \\
&\quad intraTourEnergyUsed(ST_k) + \\
&\quad e(0) * inFlightDist(k) \qquad \leq E
\end{aligned}$$

The first term of the sum on the left hand side of the inequality is the energy expenditure from the launch point until the first target of the drone subtour; the second term is the energy expended in the middle of the drone subtour; the third term is the energy expended by the drone returning to the land point, carrying zero package weight.

Service Time

For each target t_i that is visited by a drone, there may be a fixed service time β_i. However, we continue to require the drone to return to the mothership within R time units of launch, inclusive of total service time at targets. For each drone subtour $ST_k \in C$, we first compute $serviceTime(k) = \sum_{t_i \in ST_k} \beta_i$. We then modify (L6) of `LENCOMP` from:

$$(outFlightDist(k) + intraFlightDist(k) + inFlightDist(k))/\alpha \leq sTime(k)$$

to:

$$(outFlightDist(k) + intraFlightDist(k) + inFlightDist(k))/\alpha + serviceTime(k) \leq sTime(k).$$

Chapter 5: The Mothership and Drone Problem: Dealing with Obstacles and Non-Convexities

5.1 Introduction

5.1.1 Limitations of the MDRP Model

Previously, we introduced the Mothership and Drone Routing Problem. A fundamental assumption of the model was that the launching vehicle (i.e., *the mothership*) was capable of moving according to the Euclidean metric. This assumption may be reasonable in some circumstances, particularly if the mothership is itself an airplane operating in unconstrained airspace, or if the mothership is operating in open seas, where there are relatively few obstructions (i.e., land, political/military boundaries, etc.).

However, in many circumstances, it is not reasonable to assume that the mothership may operate according to the Euclidean metric without accounting for obstacles. Dry land, shallow waters, political boundaries, military threats, piracy, bad weather conditions, and other circumstances may force the mothership to take a non-direct route.

Moreover, if we approximate the boundaries of these obstacles by polygons, the feasible domain of launch and landing locations (i.e., $\mathbb{R}^2$ minus the union of the interiors of the polygons) is non-convex. This non-convexity prohibits the use of the methods described in Chapter 4.

5.1.2 Application Background

A video released by Boeing [15] in January 2018 showcased a prototype of an autonomous drone that has been developed. The drone is an octocopter with vertical take-off and landing capabilities and is described as an unmanned cargo aerial vehicle (CAV). Boeing's CAV drone is capable of launching with much larger payloads than drones that have been showcased by Amazon, Google, UPS, or DPD. In the video, a Boeing engineer speaks of delivering 250 to 500 pounds of cargo at a range of 10 to 20 miles.

The United States Navy frequently engages in disaster relief efforts around the world. [36] After the 2010 earthquake in Haiti, which measured 7.0 on the Richter Scale, and the 2004 Indian Ocean Earthquake and Tsunami, the United States Navy launched relief efforts. These relief efforts involved large naval vessels bringing supplies and medical doctors to ports. However, bringing relief supplies inland to remote villages remains a challenge.

We envision a similar disaster relief scenario. However, instead of the ship visiting ports, a cargo drone rides atop a ship. Disaster relief supplies are loaded onto the cargo drone, the drone is launched to an isolated village, supplies are offloaded from the drone to the village, and the drone returns to the mothership, where its batteries are replaced and cargo replenished. By utilizing drones, we avoid many problems related to poor or damaged road infrastructure, which limit the inland distribution of supplies. Moreover, the views offered by the drone while flying into these villages may provide valuable information to prioritize future operations. Nonetheless, the ship must take a path that avoids any dry land.

5.2 Problem Definition

In the mothership and drone routing problem with obstacles (MDRP+O), there exists one mothership and one drone. The drone is capable of moving freely in the Euclidean plane, $\mathbb{R}^2$. The mothership also moves according to the Euclidean plane, except that its path may not intersect with any predetermined forbidden regions. These forbidden regions are called obstacles. We define $Obst$ as the set of obstacles that the ship must avoid. We assume that any coastline may be approximated by the edges of a polygon. Thus, each $o \in Obst$ is a region corresponding to the interior of a polygon. There is no requirement that these polygons be convex or regular.

The mothership and the drone begin at a starting location, denoted $orig$. There exists a set of target locations T. For each $t_i \in T$, we require that the drone launch from the mothership, fly to t_i, then return to the mothership. After all targets have been visited, the mothership and drone return to a final location, denoted $dest$. In this problem, we will assume $orig$ and $dest$ are the same location. However, all results in this paper are easily extendable to the case that $orig$ and *dest* are different locations.

The drone may not be separated from the mothership for more than R consecutive time units. The mothership has unit maximum speed; the drone has a maximum speed of α. The drone may not visit multiple targets consecutively; it must return to the mothership after visiting a target.

The goal is to find a path of minimum duration that begins at *orig*, ends at *dest*, and where every $t_i \in T$ is visited by the drone.

5.3 Solution Method Overview

Our solution method contains four major steps. They are the following.

1. Pre-compute the "wet route distance" between each pair of vertices for any obstacle polygon. This saves computational effort in later steps.

2. Form a discretization of potential launch/landing locations around each target location.

3. Solve a Generalized Traveling Salesman Problem. The solution will serve as the path of the mothership in an initial feasible solution for the MDRP+O.

4. Apply a sequential second order cone program that iteratively improves the existing solution until a termination criterion is reached.

We provide details of these steps in the following sections.

5.4 Step 1: Compute Pairwise Wet Route Distances

Each obstacle $o \in Obst$ is the shape of a polygon. Let $V(o)$ denote the set of vertices of the polygon defining obstacle o. Similarly, let $V(Obst) = \cup_{o \in Obst} V(o)$ be the union of all polygon vertices among all obstacles.

For each $v_i, v_j \in V(Obst)$, we wish to compute the shortest path possible by a mothership from v_i to v_j without the mothership moving through an obstacle polygon. The method we use to compute these *wet route distances* is founded upon the work of [18].

For each $v_i, v_j \in V(Obst)$, we check if there exists a direct line-of-sight between v_i and v_j. A *line-of-sight* exists between v_i and v_j if the line segment connecting v_i and v_j does not pass through the interior of any polygon $o \in Obst$.

We construct a graph $G = (V(Obst), E)$, where an edge $(v_i, v_j) \in E$ with corresponding edge cost of $\|v_i - v_j\|$ exists if and only if v_i and v_j have a direct line-of-sight with one another.

We next compute all pairs of shortest paths over graph G, for any pair of vertices $v_i, v_j \in V(Obst)$. If $|V(Obst)| = m$, then this can be done in by applying Dijkstra's Algorithm m times, once for each origin $v_i \in V(Obst)$, at a total worst-case computational cost of $O(m^3)$, or $O(m^2 log(m))$ if the graph is non-dense. [21]

We use $wrd(v_i, v_j)$ to denote the wet route distance between two vertices v_i and v_j.

5.5 Step 2: Discretize Potential Launch/Landing Locations

A drone with maximum flight time of R and speed α has a maximum flight distance of $R\alpha$. Suppose a drone launches from the mothership at a location $launch_i$, flies to target location t_i, and returns to the mothership at location $land_i$.

Suppose that $\|launch_i - t_i\| > R\alpha/2$ and $\|land_i - t_i\| > R\alpha/2$. Then a drone operation flying from $launch_i$ to t_i and t_i to $land_i$ has a combined flight distance greater than $R\alpha$, which exceeds the maximum range of the drone. Thus, for all targets t_i, any feasible solution requires that at least one of $launch_i$ and $land_i$ to be within distance $R\alpha/2$ of t_i. Moreover, whenever there exists at least one location accessible to the ship within distance $R\alpha/2$, p_i, for each $t_i \in T$, a feasible solution exists for the MDRP+O, where $p_i = land_i = launch_i$.

Our goal, at this point, is to form a feasible solution. This requires forming a closed tour with a launch and/or landing point within radius $R\alpha/2$ for each $t_i \in T$. For each $t_i \in T$, we construct a circle of radius $shrinkFactor * R\alpha/2$, where $0 < shrinkFactor \leq 1$. We then discretize the perimeter of that circle into $discretizationResolution$ equally spaced points. For a target location t_i represented by the coordinate pair (x_i, y_i), the set of discretized points around that target location is defined by $discretization(i)$. The computation of $discretization(i)$ is described in the below pseudocode.

`For each` $t_i \in T$:

$discretization(i) = \emptyset$

`For` $j = 0, 1, 2, ..., discretizationResolution - 1$:

$angle = 2\pi j / discretizationResolution$

$xOffset = shrinkFactor * R\alpha/2 * cos(angle)$

$yOffset = shrinkFactor * R\alpha/2 * sin(angle)$

$point = (x_i + xOffset, y_i + yOffset)$

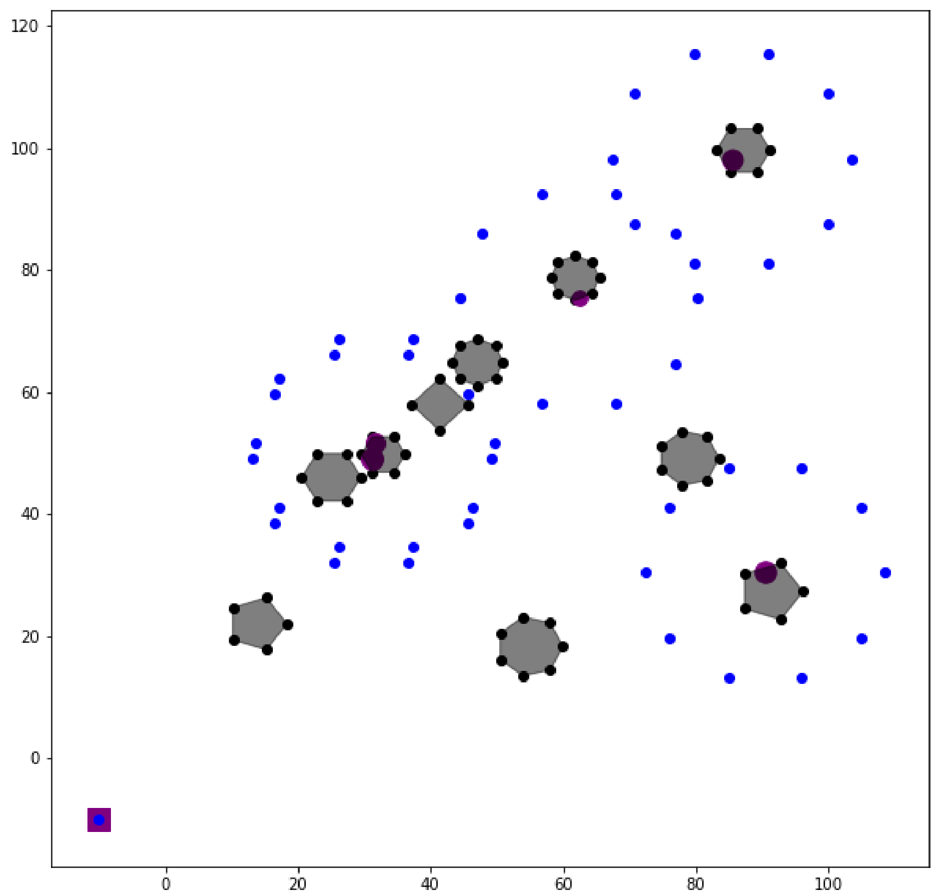
`If` $point$ `not contained in any` $o \in Obst$

$discretization(i).add(point)$

We only retain those points on the perimeter of a circle if the point does not lie in the interior of an obstacle polygon. An example of this discretization process with five target locations, ten obstacle polygons, $shrinkFactor = 0.9$ and $discretizationResolution = 10$ is shown in Figure 5.1. We set $shrinkFactor = 0.9$ because this tended to result in a better initialization than $shrinkFactor = 1.0$ and $shrinkFactor = 0.8$ in preliminary testing. The intuition is that we do might not wish to initialize with a launch/landing location on the absolute edge of the drone's range. Doing so may get our solution stuck in a highly suboptimal local optimum.

We define $dPoints = \cup_{t_i \in T} discretization(i)$ as the union of all discretized points constructed around all target locations.

Figure 5.1: There are ten obstacle polygons (grey polygonal regions). There are five target locations (purple circles) located within the obstacle locations. These represent targets on land. Around each target location, there is a ring of ten blue circles, because $discretizationResolution = 10$. The depot is indicated by purple square in the bottom left.



As a problem input for the Generalized Traveling Salesman Problem (which will be solved in Step 3), we must define the cost of traversing an arc (p_i, p_j) with $p_i, p_j \in dPoints$. In Figure 5.1, this corresponds to finding the shortest wet route distance between each pair of blue points. To compute the cost between two discretized points, we will compute the shortest path that does not pass through the interior of any obstacle polygon. For a pair of points p_i and p_j, if there exists a direct line-of-sight, then the shortest path between them is a linear segment with the simple Euclidean distance $\|p_i - p_j\|$. If there does not exist a direct line-of-sight between p_i and p_j, the shortest path between them that avoids all obstacles has a distance $d(p_i, p_j)$ which may be computed as follows.

$$d(p_i, p_j) = min_{v_a \in LOS(p_i), v_b \in LOS(p_j)}(\|p_i - v_a\| + wrd(v_a, v_b) + \|p_j - v_b\|)$$

In the above, $LOS(p)$ refers to the set of vertices $v \in V(Obst)$ such that there exists a direct line-of-sight between p and v.

Proof that this computation leads to the shortest path is found in [18]. The general idea is that is that if a direct line-of-sight does not exist between p_i and p_j, then the shortest path between p_i and p_j necessarily makes turns at one or more obstacle vertices. The term v_a corresponds to the first turn point on the path between p_i and p_j. The term v_b corresponds with the last turn point on the shortest path between p_i and p_j. The total path distance can be written as the sum of the Euclidean distance from p_i to the first turn point, the wet route distance from the first turn point to the last turn point, and the Euclidean distance from the last turn point to p_j.

5.6 Step 3: Solve a Generalized TSP

The Generalized Traveling Salesman Problem (GTSP) is a generalization of the traveling salesman problem. [32] In the ordinary traveling salesman problem, a solution tour must visit each target location, and begin and end at some predefined depot. In the GTSP, however, a set of locations is divided into clusters and it is only necessary that the solution path visit at least one location in each cluster. The objective of the GTSP is to minimize the cost of the closed tour that satisfies all visit requirements.

In our case, we require that the tour begin and end at a predefined location $orig = dest$. We also require that the mothership visit at least one point within $discretization(i)$ for each $t_i \in T$. Such a solution ensures that the mothership pass within distance $R\alpha/2$ of each target t_i.

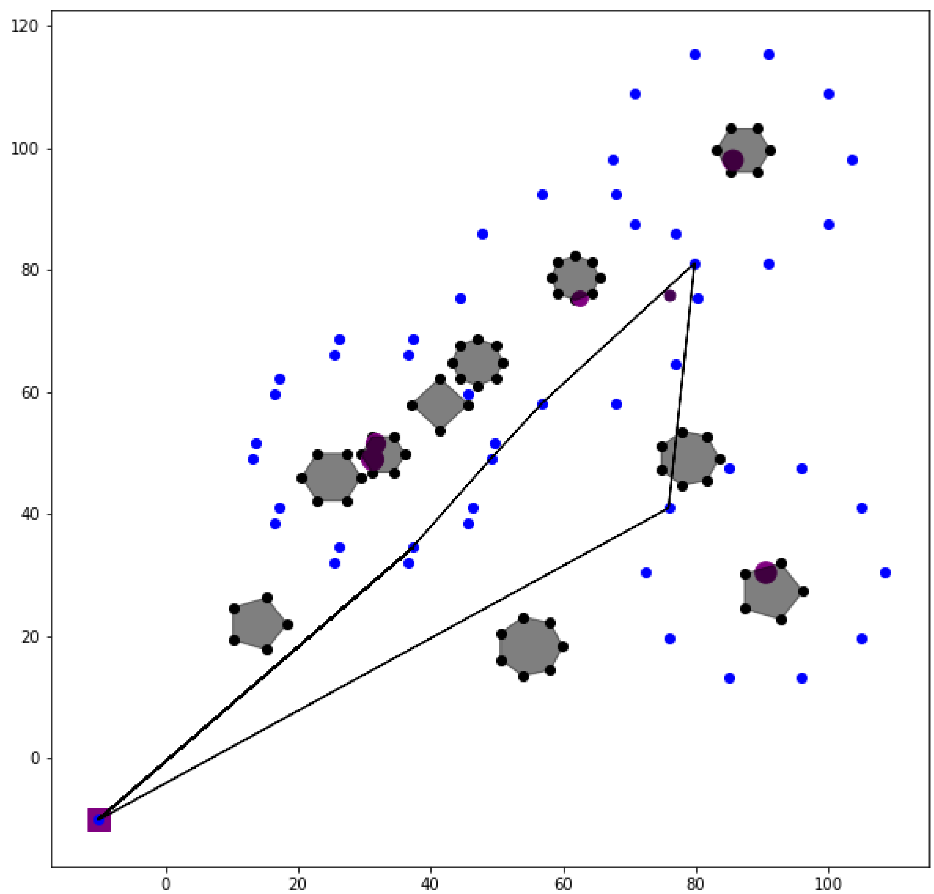
To solve this Generalized Traveling Salesman Problem, we used the formulation below, which was solved using Gurobi 7.5.1. The subtour elimination constraints were added in a lazy fashion. That is, we solved a relaxed version of the problem without subtour elimination constraints. If an optimal solution is found for a relaxed problem that contains any subtours with less than $|T| + 1$ arcs, we add a clique constraint that disallows each subtour found in the solution and we solve again. We continue solving and adding clique constraints until an optimal solution is found that contains only a single closed tour that visits all $|T|$ clusters and $orig = dest$.

$$\begin{aligned}
&\texttt{Minimize} && \sum_{p_i, p_j \in dPoints} d(p_i, p_j) x(p_i, p_j) \\
&\texttt{Subject to:} && \\
& && \sum_{p_i \in discretization(k)} x(p_i, p_j) = 1, \forall t_k \in T \\
& && \sum_{p_j \in discretization(k)} x(p_i, p_j) = 1, \forall t_k \in T \\
& && \sum_{p_j \in dPoints} x(orig, p_j) = 1 \\
& && \sum_{p_i \in dPoints} x(p_i, dest) = 1 \\
& && \texttt{number of arcs in any subtour } \geq |T| + 1 \\
& && x(p_i, p_j) \in \{0, 1\}, \forall p_i, p_j \in dPoints
\end{aligned}$$

After obtaining a solution, if $x(p_i, p_j) = 1$ in the optimal solution and if $p_i \in discretization(k)$, then we set $launch_k = p_i$ and $land_k = p_i$. Our initial feasible solution is characterized by the mothership traveling on the shortest wet route path between each pair of locations (p_i, p_j) wherever $x(p_i, p_j) = 1$. The drone flights in our initial solution fly from $launch_i$ to t_i to $land_i$ for each $t_i \in T$.

An example GTSP solution, which defines the mothership path in an initial solution to the MDRP+O, is seen in Figure 5.2.

Figure 5.2: There are ten obstacle polygons (grey polygonal regions). There are five target locations (purple circles) located within the obstacle locations. These may reprsents five targets on land that must be visited. Around each target location, there is a ring of ten blue circles, because $discretizationResolution = 10$. The depot is indicated by purple square in the bottom left. The black line segments connect consecutive visit locations in the optimal Generalized TSP solution. Notably, for each target location, at least one blue point in the circular ring surrounding the target location is visited in the GTSP solution.



5.7 Step 4: Solve a Sequential Second Order Cone Program

Our sequential second order cone program has the following broad structure.

Iterate for *maxIter* iterations:

- Precompute obstacle-free regions around the launch and landing locations of the incumbent solution.
- Solve a second order cone program.
- Let the solution of the second order cone program become the incumbent solution.

5.7.1 Precomputed Values

If we consider the Euclidean plane with one or more polygons removed, the resulting region is non-convex. The set of allowable launch or landing locations (i.e., obstacle-free regions) is thus a non-convex set, because it consists of the Euclidean plane minus a union of closed polygons.

We wish to apply the constraint that each launch and landing location must be in an obstacle-free location. However, we seek to write this constraint in convex form.

In our sequential second order cone program, we assume there exists an incumbent feasible solution. The initial feasible solution for the first iteration of the sequential second order cone program comes from the Generalized TSP solution. On subsequent iterations of the sequential second order cone program, the incumbent solution is the optimal solution from the previous iteration of the second order cone program.

The initial solution may be fully represented by a set of launch locations and a set of landing locations. In particular, we use $initLaunch_i$ to denote the location where the drone launches from the mothership towards $t_i \in T$ in the incumbent solution. Likewise, $initLand_i$ denotes the location where the drone lands on the mothership after visiting $t_i \in T$.

For each $t_i \in T$, we will consider a circular region around $initLaunch_i$ of maximum radius such that the circular region does not intersect with any of the obstacle polygons. The radius of this circle is denoted $launchFreedom_i$. Similarly, we computed $landFreedom_i$ as the radius of the largest circle around $initLand_i$ that does not intersect any obstacles. The computation of $launchFreedom_i$ and $landFreedom_i$ can be achieved using basic geometry.

The idea is that we know these circular regions around the incumbent solution's launch and landing points are free of obstructions. Moreover, by considering a circular region, we are able to write an optimization problem in the form of a convex program.

In addition to computing these obstacle-free radii around each incumbent launch and landing location, we will also pre-compute what we call *waypoints*.

If we compute the shortest wet route path from $initLand_i$ to $initLaunch_{i+1}$, then either the path is direct (a direct line-of-sight exists) or there are one or more turning points along the way. If a direct line-of-sight does not exist between $initLand_i$ and $initLaunch_{i+1}$, then $firstObstC_i$ is the location of the first turning point along the wet route path between them and $lastObstC_i$ is the location of the last turning point along the wet route path between them.

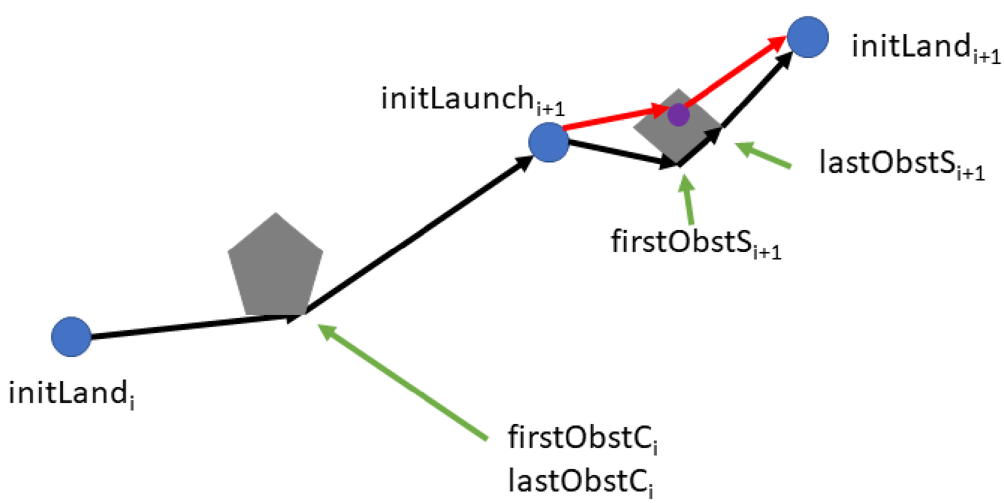
If we compute the shortest wet route path from $initLaunch_i$ to $initLand_i$, then either the path is direct (a direct line-of-sight exists) or there are one or more turning points along the way. If a direct line-of-sight does not exist between $initLaunch_i$ and $initLand_i$, then $firstObstS_i$ is the location of the first turning point along the wet route path between them and $lastObstS_i$ is the location of the last turning point along the wet route path between them.

All of $firstObstC_i, lastObstC_i, firstObstS_i, lastObstS_i$ correspond with the location of a vertex of an obstacle polygon. In Figure 5.3, we display a portion of the route of an incumbent solution to illustrate the meaning of these variables.

5.7.2 Solve a Second Order Cone Program

We then solve the second order cone program presented below. After solving, we will save the decision variables $launch_i$ and $land_i$ for $i = 1, 2, ..., |T|$, as these

Figure 5.3: A partial incumbent route showcasing terminology related to turning points. The black lines represent the path of the ship in a portion of the incumbent solution. Red lines represent the flight path of the drone. Grey polygonal regions are obstacles. The purple cirlce is target location t_{i+1}. Blue circles are either landing or launching points on the incumbent solution. While mothership and drone are together, the ship must only make one turn. Thus, $firstObstC_i$ and $lastObstC_i$ are the same location. While ship and drone are separated during the flight of the drone to target t_{i+1}, the ship must turn twice: first at $firstObstS_{i+1}$ and last at $lastObstS_{i+1}$.



allow us to fully determine the solution.

`Minimize:` $\sum_i cTime_i + sTime_i$

`Subject to:`

$$\|launch_i - initLaunch_i\| \leq launchFreedom_i$$

$$\|land_i - initLand_i\| \leq landFreedom_i$$

$$\|launch_i - t_i\| \leq droneOutboundDist_i$$

$$\|t_i - land_i\| \leq droneInboundDist_i$$

$$(outboundDist_i + inboundDist_i)/\alpha \leq sTime_i$$

`If not exist direct line of sight from` $initLaunch_i$ `to` $initLand_i$:

$$\|launch_i - firstObstS_i\| \leq distToFirstObstS_i$$

$$\|lastObstS_i - land_i\| \leq distToLastObstS_i$$

$$distToFirstObstS_i + wrd(firstObstS_i, lastObstS_i) + distToLastObstS_i \leq sTime_i$$

`Else:`

$$\|launch_i - land_i\| \leq sTime_i$$

`If not exist direct line of sight from` $initLand_i$ `to` $initLaunch_{i+1}$:

$$\|land_i - firstObstC_i\| \leq distToFirstObstC_i$$

$$\|lastObstC_i - launch_{i+1}\| \leq distToLastObstC_i$$

$$distToFirstObstC_i + wrd(firstObstC_i, lastObstC_i) + distToLastObstC_i \leq cTime_i$$

`Else:`

$$\|land_i - launch_{i+1}\| \leq cTime_i$$

$sTime_i \leq R$

$cTime_i \leq R$

$launch_0 = orig$

$land_0 = orig$

$launch_{|T|+1} = dest$

$land_{|T|+1} = dest$

5.7.3 Update the Solution

We now set $initLaunch_i \leftarrow launch_i$ and $initLand_i \leftarrow land_i$ for each of $i = 1, 2, ..., |T|$. The solution of the second order cone program of the current iteration will be the incumbent solution for the next iteration.

5.8 Illustration of First Iterations of the Sequential Second Order Cone Program on Example Instance

In Figures 5.4, 5.5, 5.6, 5.7, 5.8, and 5.9, we display the initial solution and the solution after each of the first five iterations of the second order cone program. In each of these images, obstacle regions are shown as gray polygons. Purple circles represent target locations. Green line segments display the path of the mothership. (Note: Although the images seem to show the path of the mothership passing through obstacle polygons, the actual path does not. In the images, we simply connect consecutive destinations by a linear segment, although the mothership will actually use a wet route path.) Blue line segments display outbound drone flight segments. By outbound, we mean a segment that begins at the mothership and ends at a target location. Red line segments display inbound drone flight segments. By inbound, we mean a segment that begins at a target location and ends back at the mothership. Blue circles show a circular region of maximum radius that is obstacle-free around each launch point. Red circles show a circular region of maximum radius that is obstacle-free around each land point. The radii of these circles are related to the precomputed constants $launchFreedom_i$ and $landFreedom_i$ for $i = 1, 2, ..., |T|$.

We only display solutions after the first five iterations of the sequential second order cone program, however, we note that after running 25 iterations of the sequential second order cone program, the objective value appears to converge to an objective of 287.3100.

5.9 Computational Experiments

In all computational results, we set the location of the depot as $orig = dest = [-10, -10]$. We set the maximum flight time of the drone to 20 units and the relative speed of the drone to the mothership, $\alpha = 2$. The location of the centroids of all obstacle polygons were generated uniformly, where the x-coordinate and y-coordinate are randomly selected from $\mathbb{U}[0, 100]$. A regular polygon was constructed around the randomly selected centroid. The number of sides for the randomly

Figure 5.4: Iteration 0: adapted Generalized TSP solution.

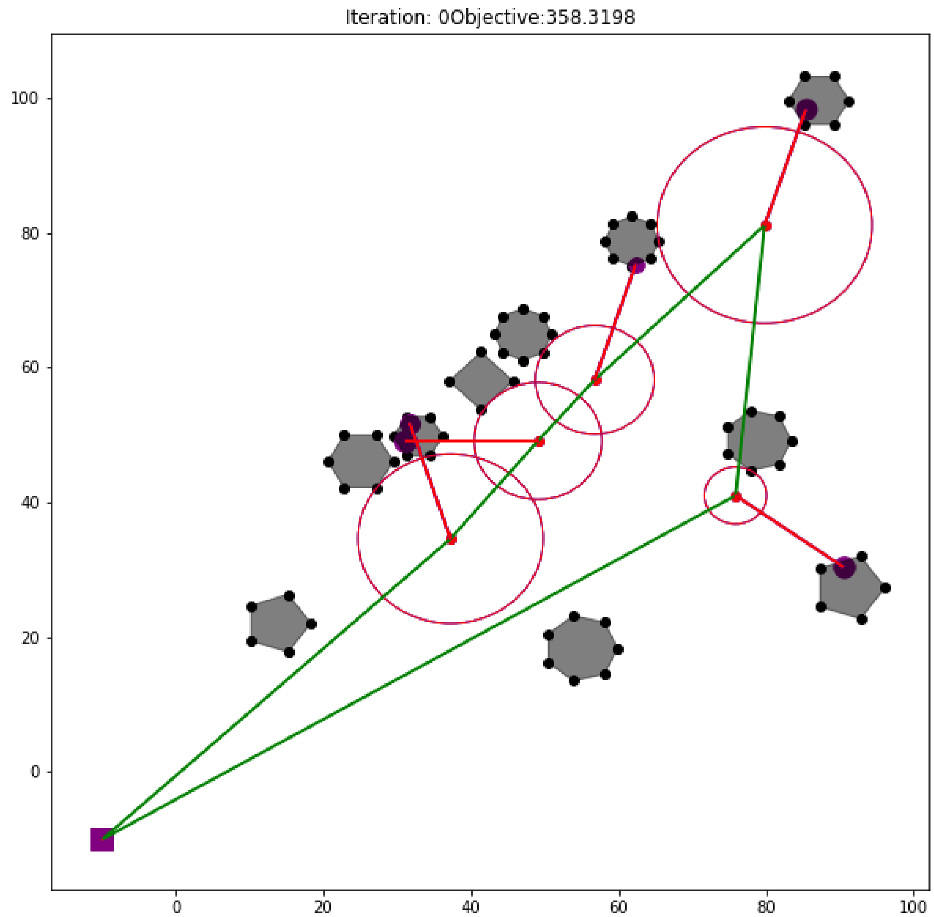


Figure 5.5: Iteration 1: Solution after 1 iteration completed of sequential second order cone program.

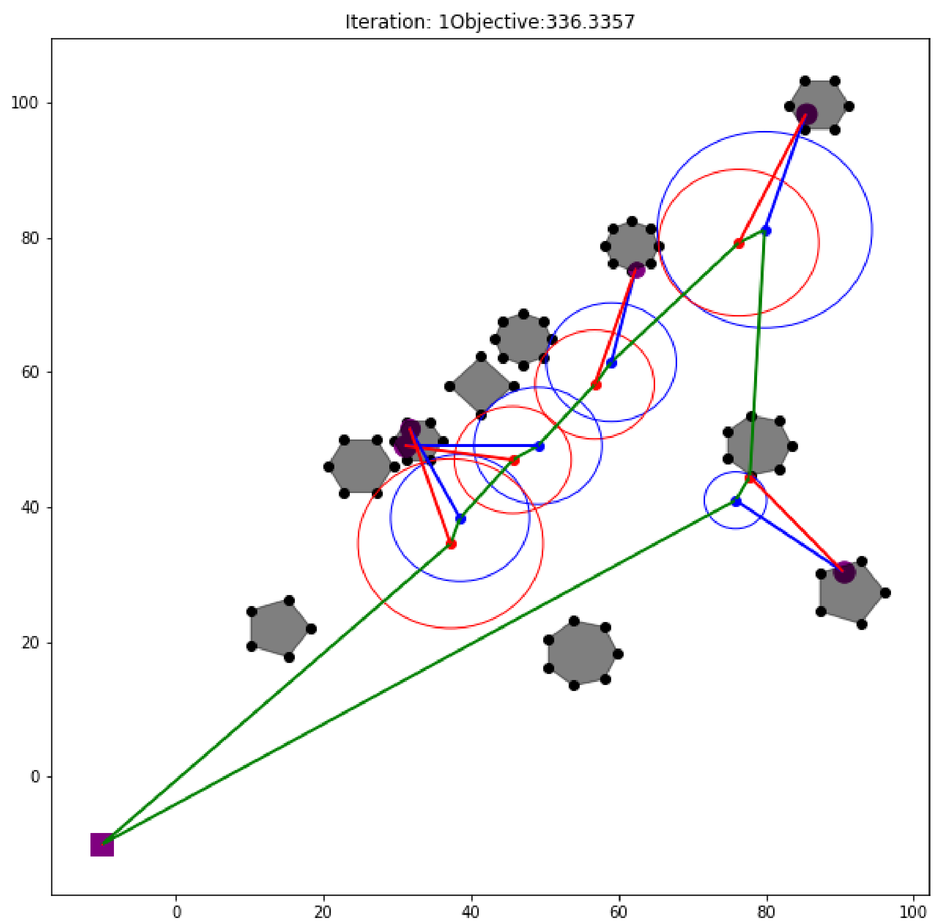


Figure 5.6: Iteration 2: Solution after 2 iterations completed of sequential second order cone program.

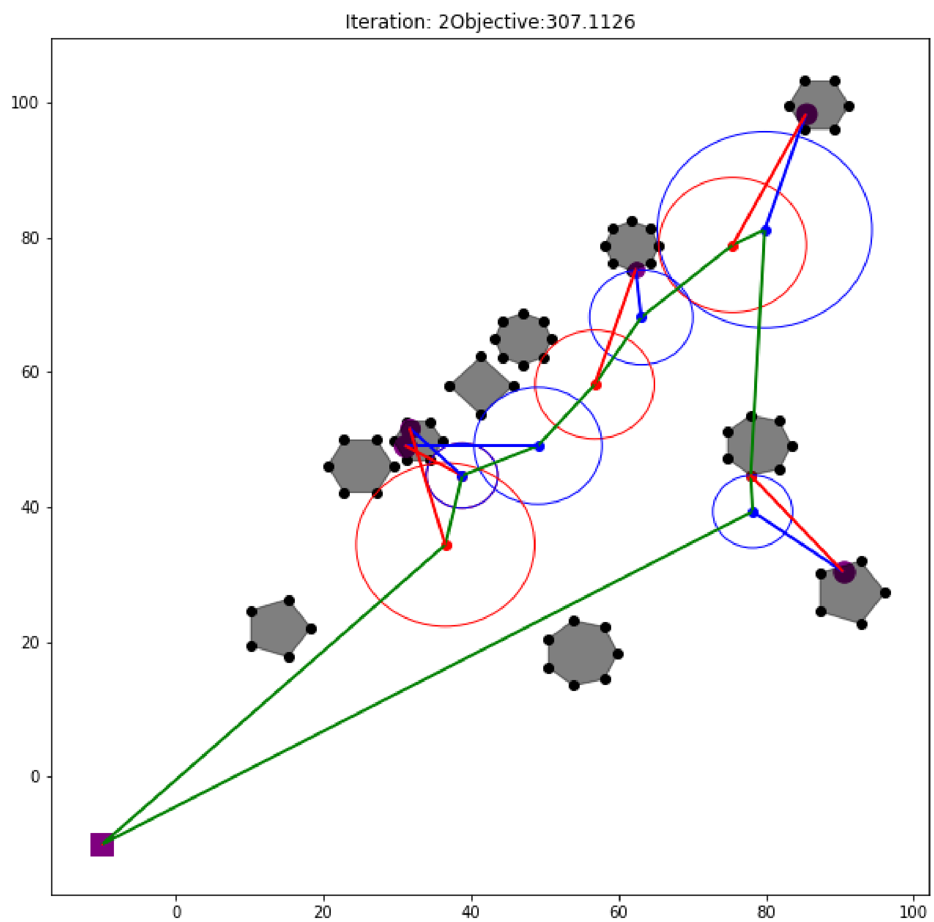


Figure 5.7: Iteration 3: Solution after 3 iterations completed of sequential second order cone program.

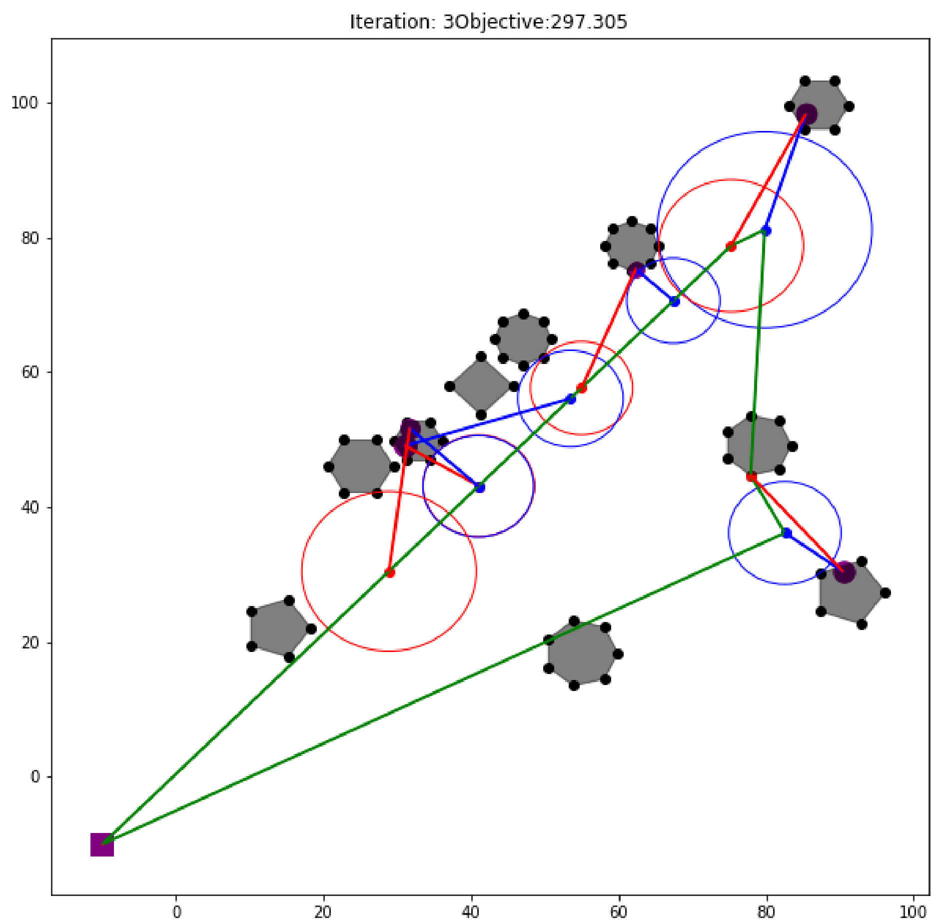


Figure 5.8: Iteration 4: Solution after 4 iterations completed of sequential second order cone program.

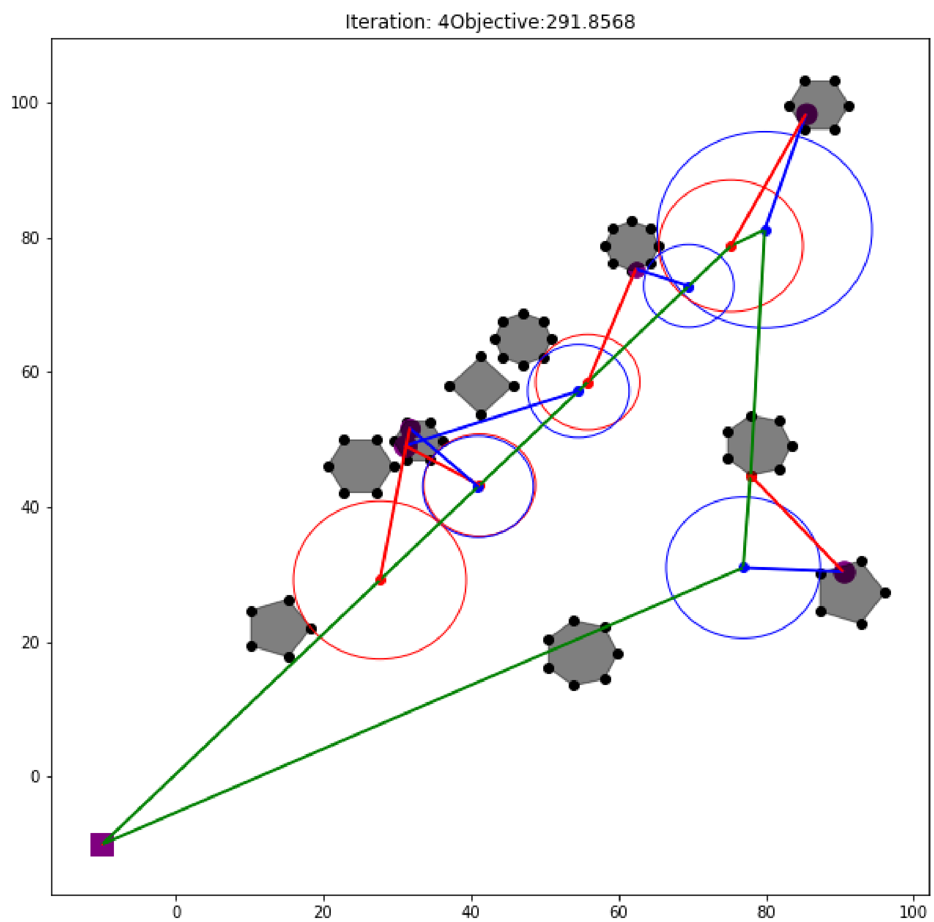
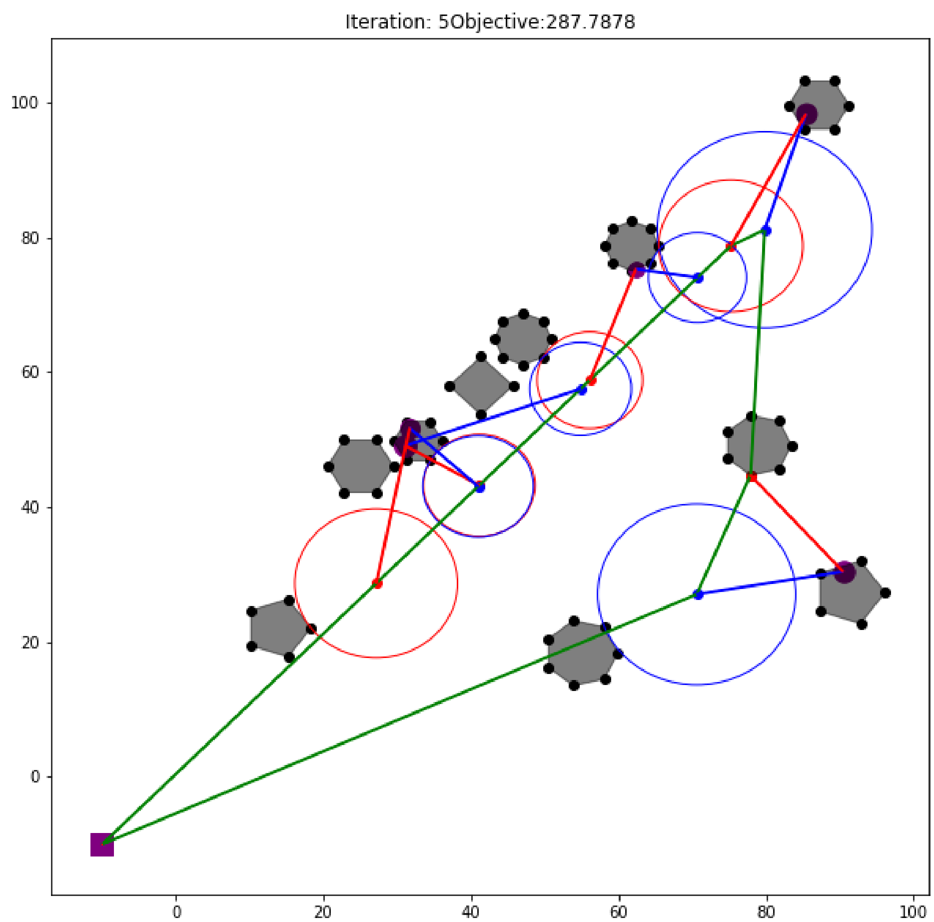


Figure 5.9: Iteration 5: Solution after 5 iterations completed of sequential second order cone program.



generated regular polygon varied from three to eight, each with a probability of 1/6. The polygon radius was selected from a the uniform distribution $\mathbb{U}[3, 5]$. By the polygon radius, we mean the distance from the centroid of the polygon to any vertex. Throughout, we set the maximum number of second order cone program iterations to $maxIter = 25$.

Target locations were selected uniformly among the area bounded by obstacle polygons. That is, all target locations were uniformly distributed among the "dry land" area of the instance.

In the table of results, we additionally have the following instance parameters, which varied depending on the set of instances.

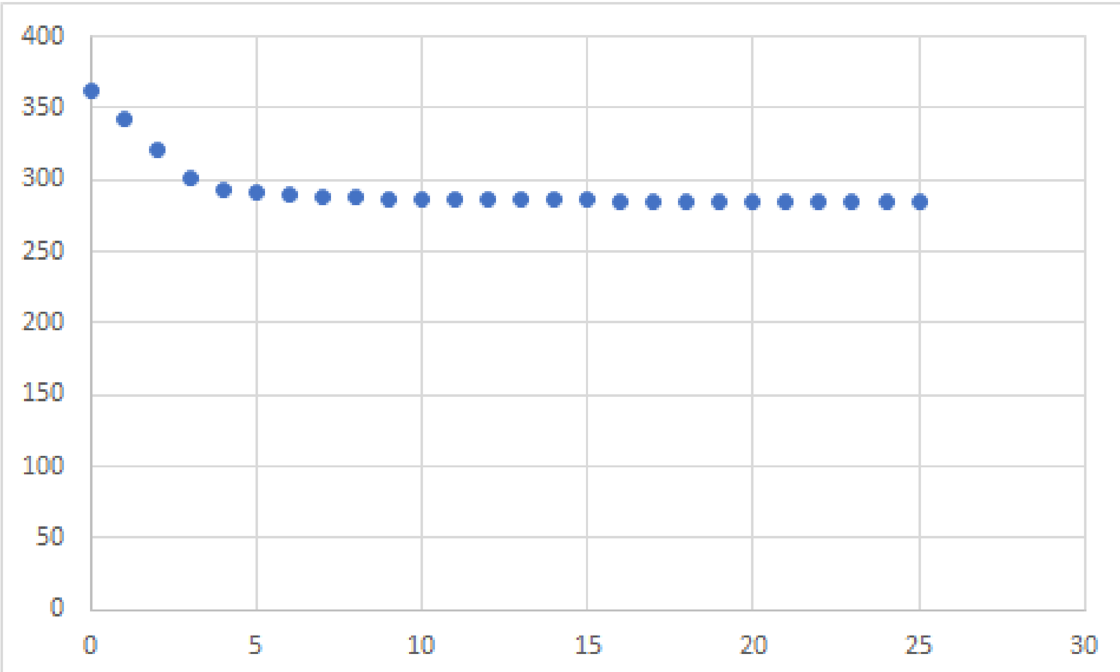
- $|Obst|$: the number of obstacle polygons randomly
- $|T|$: the number of target locations.

In Table 5.1, each row of observations reports the average over 25 randomly generated instances for the given number of obstacles, $|Obst|$, and the given number of targets, $|T|$. In total, there were 200 random instances tested. In column Init Obj, we report the average initial objective value corresponding to the initial solution that follows directly from the Generalized TSP solution. In column Final Obj, we report the lowest observed objective value after applying 25 iterations of the sequential second the order cone program. Gap is computed as (Init Obj - Final Obj)/(Init Obj). The columns Step 1 Time, Step 2 Time, Step 3 Time, and Step 4 Time report the average computational time in seconds for each of the four steps of the algorithm.

$\|T\|$	$\|Obst\|$	Init Obj	Final Obj	Gap	Step 1 Time	Step 2 Time	Step 3 Time	Step 4 Time
5	5	281.001	237.246	0.1557	0.283	0.096	0.051	0.737
5	10	301.114	252.100	0.1628	2.504	0.401	0.056	1.268
5	15	292.470	245.475	0.1607	7.920	0.930	0.056	2.041
5	20	310.527	263.400	0.1518	19.542	2.003	0.044	3.380
10	5	420.879	309.417	0.2648	0.339	0.566	4.427	1.462
10	10	420.271	315.597	0.2491	2.527	2.208	1.701	2.416
10	15	440.237	329.089	0.2525	7.407	5.082	2.123	3.510
10	20	436.171	329.784	0.2439	19.621	9.964	1.209	5.295

Table 5.1: Computational results for the MDRP+O.

Figure 5.10: The horizontal axis displays the number of iterations of the sequential second order cone program completed. The vertical axis displays the best known objective value after the given number of completed iterations, averaged over the 200 instances tested.



In Figure 5.10, we visually display aggregate data for the 200 separate instances that were tested. The horizontal axis displays the number of iterations of the sequential second order cone program completed. The vertical axis displays the average objective value over the 200 instances after the given number of iterations of the second order cone program were completed. Very little objective value improvement is seen after the first several iterations of the sequential second order cone program.

Additionally, we generated instances with 15 or more target locations. No batch of 25 instances solved in less than 5 hours. The computational bottleneck appeared to be related to solving the Generalized TSP (i.e., Step 3).

5.10 Generalizing to Energy Constraints

Suppose that rather than having maximum flight duration, a drone has, instead, a maximum energy capacity given by $EMAX$. Suppose e is an increasing function, where $e(W)$ gives the rate of energy depletion for the drone while carrying a package with weight W. Let us suppose each target location $t_i \in T$ corresponds to the location of a package that must be delivered. In particular, suppose w_i is the weight of a package to be delivered to target location t_i. The objective and other constraints are otherwise identical to before.

To account for this new version of the problem, we make a few small adjustments to the algorithm. Firstly, in Step 2, when forming a discretized ring of points surrounding a target location t_i, we now use a radius of $\alpha * shrinkFactor * (EMAX/(e(0)+e(w_i))$, instead of $\alpha * shrinkFactor * (R/2)$. Secondly, in the second order cone program, we replace the line:

$$sTime_i \leq R$$

with:

$$(e(w_i) - e(0)) * outboundDist_i + e(0) * sTime_i \leq EMAX.$$

We note that setting $EMAX = R$ and $e(W) \equiv 1$ for all values of W is equivalent to the original problem with a maximum flight time of R.

5.11 Future Work

There are several future directions we would like to pursue. Firstly, after the Generalized TSP has been solved, the relative order of target visitation is fixed for the remainder of the algorithm. We would like to consider ways to modify the visit order, perhaps by randomly perturbing some inputs and restarting the algorithm.

Additionally, we would like to do more parameter tuning and experiment with alternative stopping criteria. Instead of terminating the algorithm after a fixed number of iterations, we will seek to detect objective value convergence before terminating. We may also wish to consider replacing circular obstacle-free regions with elliptical regions.

We wish to consider using alternative methods to generate initial solutions for larger instances. Initial testing indicated that the computational time increases very rapidly within step 3 (i.e., solving the Generalized TSP) as the number of targets increase. Rather than solving the Generalized TSP exactly, we could replace with a heuristic method such as the one described by [59].

We would like to use real-world coastlines and map data to form real obstacle polygons. Along with this, we would like to see if it is feasible to account for the curvature of the earth.

Naturally, we would like to consider a multi-mothership and/or multi-drone extension to this problem. It may also be interesting to consider a discretized approach to the problem, rather than using a continuous second order cone program-based method.

An entirely different approach based on disjunctive constraints, perhaps with some similarities to the work of [63], might be possible.

5.12 Conclusions

We extended the mothership and drone routing problem to the case where obstacles (dry land, national boundaries, etc.) force a ship to deviate from using straight-line Euclidean distances. We displayed how we may find an initial feasible solution utilizing a Generalized traveling salesman formulation. We then iteratively improve an existing solution by utilizing sequential second order cone programming. The second order cone program utilizes circular obstacle-free regions around each launch and landing location to model obstacle constraints in a convex manner. From iteration to iteration, the launch and landing points are allowed to drift, which means that the optimal solution is not confined to the initial circular obstacle-free regions.

The mothership and drone routing problem with obstacles may have application to delivering emergency supplies to remote inland villages after a major disaster that may severely impact transportation and communication networks. The MDRP+O also may have application to planning military operations.

Chapter 6: The Multi-visit Drone Routing Problem

6.1 Introduction

Because truck-and-drone models of delivery are relatively new to the academic literature, many papers thus far have studied the case of a single truck and single drone model of delivery, where the drone is capable of carrying only a single homogeneous package at a time. It is frequently assumed that the maximum drone flight duration is constant and does not depnd on the weight of any packages to be delivered.

6.2 Problem Definition

The Multi-visit Drone Routing Problem (MVDRP) is a model of delivery with a single truck and a single drone. We describe the problem in the context of package delivery to fulfill online orders, although other applications may be possible.

In MVDRP, both truck and drone start at a predefined warehouse. The truck acts as a mobile depot and recharging platform for the drone. The drone may launch from the truck with one or more packages, deliver these packages to their respective locations, then return to the truck for recharging and to pick-up additional packages.

MVDRP is distinct from most other papers in the literature, as (1) it allows for the drone to visit multiple customers consecutively before returning to the truck, and (2) the final leg of delivery is conducted only by the drone. If the truck is self-driving, this model removes the need for a delivery driver on the route. (Later, we relax the assumption that all deliveries are made by the drone.)

The goal of MVDRP is to minimize completion time. Completion time is the elapsed time from the first departure of a vehicle from the warehouse until the return of the last vehicle to the warehouse. All packages must be delivered before completion time.

In the remainder of this section, we define additional problem input parameters and constraints.

6.2.1 Problem Input Parameters

The following parameters are required as input to MVDRP.

- V is a set of feasible locations where a drone may launch or land from a truck. We assume each $v \in V$ represents a location along the street network or a parking location.

- Let C be a set of customer delivery locations. We note that there is no requirement that $C \subseteq V$ or $V \subseteq C$. That is, customer delivery locations and allowable launch/landing locations may be defined independently.

- *depot* $\in V$ is a warehouse location where the truck and drone pair will start and end its route.

- $t_t(v_i, v_j)$ denotes the travel time for the truck from v_i to v_j, for any $v_i, v_j \in V$.

- $t_d(loc_i, loc_j)$ denotes the travel time for the drone from location loc_i to location loc_j, with $loc_i, loc_j \in V \cup C$.

- For each customer delivery location $c_i \in C$, we denote the weight of the package to be delivered as w_i.

- $EMAX$ is the maximum energy capacity of the battery of the drone.

- $e(loc_i, loc_j, W)$ denotes the average rate of energy dissipation by the drone per unit time, when flying from loc_i to loc_j, with $loc_i, loc_j \in V \cup C$, while carrying packages whose weight sums to W. The energy dissipation rate for a drone varies by origin/destination pair for a variety of reasons (e.g., wind direction and elevation differences between origin and destination). We only require that e be a non-decreasing function of W. In the event that the sum of package weights is infeasible for the drone to carry (i.e., too heavy to take-off), we set $e(v_i, c_j, W) = \infty$. Also, if $e \equiv 1$ is a constant function, then this is equivalent to allowing a maximum flight time of $EMAX$.

- HOV is a constant that denotes the rate of energy dissipation per unit time for a drone, whenever it is hovering. Hovering occurs when the drone arrives at a rendezvous point before the truck and must wait for the return of the truck.

6.2.2 Problem Constraints and Additional Assumptions

Additional constraints and assumptions of MVDRP are as follows.

- A drone may launch from the truck or land on the truck at a location v, only if $v \in V$.
- A drone must not run out of battery before returning to the truck.
- The capacity of the truck is infinite.
- Any service time by the drone at a customer location and associated energy dissipation is already accounted for in problem inputs t_d and e.
- The triangle inequality holds for t_t and for t_d.
- After the drone is launched, the truck begins immediately towards the rendezvous location and does not stop in between.
- The function e always returns a non-negative value. That is, the drone can never recuperate more energy than it expends while flying, even if elevation differences exist between launch and landing locations.

6.3 Solution Method: Route, Transform, Shortest Path

The solution method of this section, “Route, Transform, Shortest Path” (RTS), has three major phases.

1. Decide which order the packages should be delivered. (“Route”)

2. Construct a transformed graph with $|V| * (|C| + 1)$ vertices and compute edge costs. ("Transform")

3. Solve a shortest path problem over the graph. ("Shortest Path")

6.3.1 Phase 1: Route

Let us compute the optimal solution to the traveling salesman problem on the set of locations $C \cup \{depot\}$, using t_d as the measure of time between any pair of locations. Let us denote the result as:

$$Path = [p_0 = depot, p_1, p_2, ..., p_{|C|}, p_{|C|+1} = depot].$$

The first customer location to be visited is p_1; the second customer location to be visited is p_2, and so on.

6.3.2 Phase 2: Transform

Let us construct a graph $G' = (V', E')$ with a vertex set V' and edge set E'. For each $v \in V$, there will be $|C| + 1$ different vertices in V'. If we say that the truck and drone are at launch location $v'_{i,j}$, we mean that truck and drone are at the physical location of v_i and that the first j customer package locations $(p_1, p_2, ..., p_j)$ have been satisfied, but $p_{j+1}, ..., p_{|C|}$ have not been visited yet.

For each pair of vertices v'_{i_1,j_1} and v'_{i_2,j_2} where $j_1 < j_2$, we compute:

$$cost(v'_{i_1,j_1}, v'_{i_2,j_2}) = max(truckTime, droneTime).$$

We define $truckTime = t_t(v_{i_1}, v_{i_2})$, which represents the amount of time required for the truck to travel from launch location v_{i_1} to launch location v_{i_2}. The term $droneTime$ represents the amount of time for the drone to fly from launch location v_{i_1} to customers locations $p_{j_1+1}, p_{j_1+2}, ..., p_{j_2}$ (in order), then to return to the truck at v_{i_2}. If the flight is infeasible due to maximum energy expenditure of the drone, we set $droneTime = \infty$. We note that if $cost(v'_{i_1,j_1}, v'_{i_2,j_2}) = \infty$, then for any $j_3 > j_2$, $cost(v'_{i_1,j_1}, v'_{i_2,j_3}) = \infty$. This detail is important to reduce computational time for the construction of the modified graph for large instances.

Additionally, for each pair of vertices $v'_{i_1,j}$ and $v'_{i_2,j}$, we compute:

$$cost(v'_{i_1,j}, v'_{i_2,j}) = t_t(v_{i_1}, v_{i_2}).$$

This cost is relevant in the case that we land a drone after delivering to customer p_j at location v_{i_1}, but wish to reposition the truck to location v_{i_2} before launching the drone towards customer p_{j+1}.

An edge $(v'_{i_1,j_1}, v'_{i_2,j_2})$ is added to E' if and only if $j_1 \leq j_2$ and $cost((v'_{i_1,j_1}, v'_{i_2,j_2})) < \infty$.

6.3.3 Phase 3: Shortest Path

We apply Dijkstra's Algorithm with starting vertex $v'_{depot,0}$ and terminal vertex $v'_{depot,|C|}$ on the graph G' where the cost of an arc $(v'_{i_1,j_1}, v'_{i_2,j_2})$ given by $cost((v'_{i_1,j_1}, v'_{i_2,j_2}))$. The result is a feasible solution to the MVDRP.

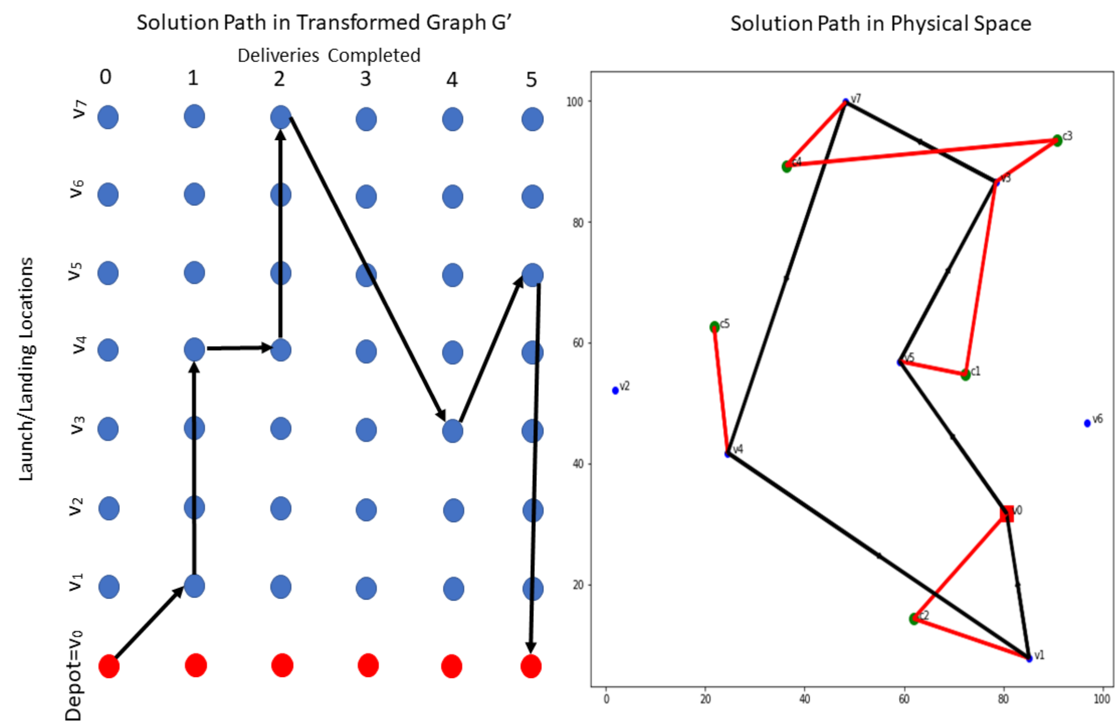


Figure 6.1: Left: the solution path traced through the transformed graph G'. Right: We display the solution path in the 2-D plane. The red square displays the depot location. Black line segments trace the path of the truck (traversed in roughly clockwise direction) and red line segments trace the flight path of the drone. Green circles show customer delivery locations and blue circles display feasible launch locations.

In a later section, we discuss how Dijkstra's Algorithm may be replaced with the A-star algorithm.

6.3.4 Figures to Visualize Algorithm

In Figure 6.1, on the left side, we display the solution path through the transformed graph G'. On the right side, we show the corresponding physical path of truck and drone. As an example, one edge of G' connects $v_{7,2}$ to $v_{3,4}$, indicating that after two packages have been delivered, the drone departs v_7 with a rendezvous point of v_3. Upon reaching v_3, four packages will have been delivered.

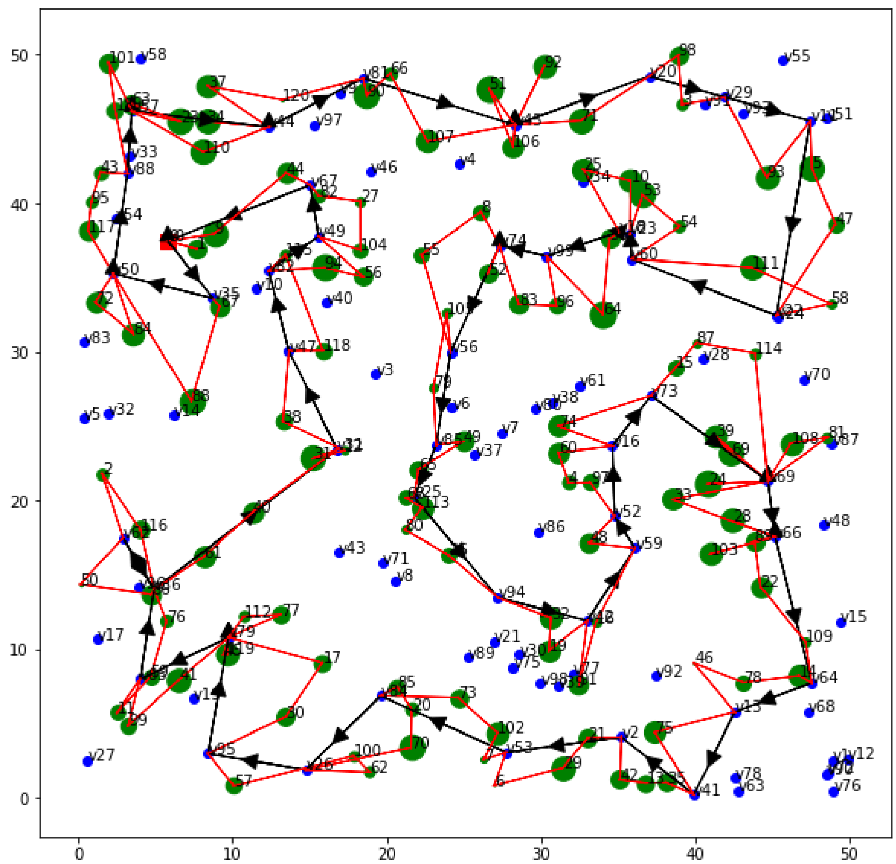


Figure 6.2: The red square (near top left) is the depot location. Black arrows display the path of the truck. Red line segments display the flight path of the drone. Customer locations are indicated with green circles. The diameter of green circles scales linearly with the weight of the packaged to be delivered at that location. Blue circles are feasible launch/landing locations.

In Figure 6.2, we display a sample solution for an instance with $|C| = 120$ customer locations. Aside from solving a TSP to initialize, the solution required 5.6 seconds of computational time. In the example, $\alpha = 2$, $|V| = 100$, $EMAX = 800$, the weight of packages were distributed uniformly over $\mathcal{U}(0, 30)$, and $e(loc_i, loc_j, W) = 10 + W^{1.5}$.

6.3.5 Theoretical Results

Let us define $\texttt{VAL}(Path)$ as the objective value returned by applying Phase 2 and Phase 3 to an input delivery order $Path = [depot = p_0, p_1, ..., p_{|C|}, depot = p_{|C|+1}]$. Let us define $\texttt{SOLN}(Path)$ as the corresponding MVDRP route formed by applying Phase 2 and Phase 3 to $Path$.

Theorem 12. *Among feasible solutions to MVDRP that obey the delivery order dictated by Path, SOLN(Path) is the best one, with corresponding objective value* $\texttt{VAL}$*(Path).*

Corollary 2. *For some input delivery order Path,* $\texttt{SOLN}$*(Path) is the optimal solution to MVDRP and* $\texttt{VAL}(Path)$ *is the optimal objective value to MVDRP.*

The worst-case computational performance of RTS, aside from solving the initial TSP, is $O(|C|^2|V|^2)$. However, if we know the drone cannot make more than k_1 consecutive deliveries before running out of battery, the worst-case performance is reduced to $O(max(k_1|C|, log(|C||V|)) * |V|^2)$. If we also know that at any launch location $v \in V$, there are no more than k_2 feasible landing locations for the drone, worst-case performance is reduced further to $O(max(k1 * k2, log(|C||V|))|C||V|)$.

6.4 MVDRP with Select Truck Delivery

Suppose $C_t^o \subseteq C$ is a set of package locations for which we have the option to deliver by truck. Suppose $C_t^r \subseteq C_t^o \subseteq C$ is a set of package locations that require delivery by the truck. We also make the assumption that a delivery by truck is not allowed to occur while a drone is airborne.

To model this problem, we will simply make the following modifications to MV-DRP inputs.

- For each $c \in C_t^o$, we will ensure that $c \in V$ and $t_d(c, c) = 0$.
- For each $c \in C_t^r$, for all $loc \in V \cup C \setminus c$, set $t_d(loc, c) = \infty$.

The idea is that we are allowing (or requiring, in the case $c \in C_t^r$) a zero-distance drone launch. In reality, the zero-distance drone launch is a delivery serviced by the truck.

6.5 RTS with Local Search

Our Route, Transform, Shortest Path, and Local Search (RTS+LS) algorithm operates similarly to RTS, but considers iteratively local neighborhoods of $Path$ and moves downhill. We define RTS+LS as follows.

1. Initialize $Path$ as the optimal TSP solution for $C \cup \{depot\}$ using t_d as the distance metric.
2. Set $oldPath = Path$.
3. Construct $neighborhood(Path)$.
4. For each $neighbor$ in the $neighborhood$, compute $\texttt{VAL}(neighbor)$.
5. Set $Path = argmin_{neighbor \in neighborhood}(\texttt{VAL}(neighbor))$.

6. If $oldPath = Path$, terminate algorithm. Else, go to step 2.

The $neighborhood(Path)$ is constructed by considering the following paths.

- Any path resulting from swapping the order of any pair $p_i, p_j \in Path \setminus depot$, such that $t_d(p_i, p_j) < maxSwapDist$. (2-point swap)

- Any path resulting from removing any single $p_i \in P \setminus depot$ and replacing after location p_j, such that $t_d(p_i, p_j) < maxSwapDist$. (1-point swap)

- Performing a 2-opt on any pair $p_i, p_j \in Path \setminus depot$ (i.e., reversing the string $p_{i+1}, p_{i+2}, ..., p_j$), such that $t_d(p_i, p_j) < maxSwapDist$. (2-opt)

Because the TSP serves as a sufficiently good initialization, we may reduce the size of the local neighborhood by only performing swaps that involve nodes that are sufficiently close to one another (i.e., within $maxSwapDist$). We assume any swaps involving nodes that are too far from one another are unlikely to improve solution quality.

Aside from imposing a maximum swap distance, the neighborhood of delivery sequences is constructed in a similar to Agatz et al. [2] in the heuristic TSP-ep-all.

6.6 Multiple Drones per Truck

In the k-Multi-visit Drone Routing Problem (k-MVDRP), we allow for a truck to carry k homogeneous drones at a time. While the truck is stopped, it can launch up to k drones simultaneously to deliver packages. However, the truck may not launch additional drones at a new location until all drones have landed. The objective and problem constraints in k-MVDRP are otherwise identical to those in MVDRP.

In the "Transform" portion of the RTS and RTS+LS algorithms, we computed the costs of edges in G' using:

$$cost(v'_{i_1,j_1}, v'_{i_2,j_2}) = max(truckTime, droneTime).$$

Our solution method for k-MVDRP is identical to the method in MVDRP, except that we compute $droneTime$ differently. For an edge $(v'_{i_1,j_1}, v'_{i_2,j_2})$, we do not set $droneTime$ as the flight time for a single drone to fly from i_2 to p_{j_1+1}, deliver $p_{j_1+1}, p_{j_1+2}, ...p_{j_2}$ in sequence, and rendezvous with the truck at i_2. Instead, we will partition the delivery of the package locations $p_{j_1+1}, p_{j_1+2}, ...p_{j_2}$ between the k drones in a manner that attempts to minimize the longest drone flight time. The longest flight time among the k drones then becomes the value for $droneTime$

If $j_2 - j_1 \leq k$, then the optimal partition is always to assign a single drone for each package. If $j_2 - j_1 > k$, we tried two simple methods for assigning the k drones to the set of package locations $p_{j_1+1}, p_{j_1+2}, ..., p_{j_2}$.

In the first method, called *block assignments*, we assign the first $\lceil (j_2 - j_1)/k \rceil$ packages to the first drone. The next $\lceil (j_2 - j_1)/k \rceil$ are assigned to the second drone, and so on until all packages $p_{j_1+1}, p_{j_1+2}, ..., p_{j_2}$ are assigned. For example, if $j_2 = 15$, $j_1 = 4$, and $k = 3$, then the first drone must deliver p_5, p_6, p_7, and p_8, the second drone must deliver p_9, p_{10}, p_{11}, and p_{12}, and the third drone must deliver p_{13}, p_{14}, and p_{15}.

The second method of partitioning is called *rotating assignments*. The assignment of packages to drones occurs in a rotating fashion. For example, if $j_2 = 15$, $j_1 = 4$, and $k = 3$, then the first drone must deliver p_5, p_8, p_{11}, and p_{14}, the second drone must deliver p_6, p_9, p_{12}, and p_{15}, and the third drone must deliver p_7, p_{10}, and p_{13}.

Regardless of partitioning method, it is assumed each drone flies from v_{i_1}, delivers its assigned packages in order, then returns to the truck at v_{i_2}.

6.7 Computational Results

We constructed a series of test instances. For each test instance, we computed (1) the optimal truck-only TSP solution, (2) the objective value for the MVRDP solution found by the RTS heuristic, and (3) the objective value for the MVDRP solution found by the RTS heuristic and 2-point swap local search. Additionally, we recorded the computational time elapsed to compute each.

For each set of instances with a specified number of customer locations, $|C|$, and a specified number of allowable launch locations, $|V|$, we randomly generated all customer locations and allowable launch locations uniformly over a 100 by 100 square grid. The depot location was also randomly generated over a 100 by 100 square grid. The weight of packages demanded by each customer was distributed uniformly over $\mathbb{U}[0,5]$. This is related to Jeff Bezos's comments which target packages up to five pounds for drone delivery. We fixed $EMAX = 40$ and we set the energy drain function $e(W) = (1 + (W/5)^4)$. This function implies a maximum drone flight time of 40 minutes while not carrying any packages ($e(0) = 40$), and a maximum flight time of only 20 minutes while carrying 5 pounds of goods ($e(5) = 20$). Maximum flight duration of the drone rapidly drops as the weight of packages carried exceeds 5 pounds. The constant HOV was set to a value of 0.5.

In these computational experiments, to determine the time of traversal for the truck between two locations, we assumed the truck moved at unit speed and traveled the Euclidean distance between two locations. The drone was assumed to move according to the Euclidean distance, but at a speed of 2 units.

In Table 6.1, we display a table of results for our test instances. Each row displays averages over 25 randomly generated instances. The columns TSP Obj, RTS Obj, and RTS+LS Obj display the average objective value for the standard TSP, the RTS heuristic, and the RTS heuristic with local search. The column TSP Time displays the average solve time, in seconds, for the standard TSP. RTS Time and RTS+LS Time display the average solve time, in seconds, for the RTS heuristic and the RTS heuristic with local search, except for the time required for the TSP initialization. RTS Gap is computed as (TSP Obj - RTS Obj)/TSP Obj. RTS+LS Gap is computed as (TSP Obj - RTS+LS Obj)/TSP Obj.

The addition of local search decreased objective values, on average, by 0.9%

(for $|C| = 50$, $|V| = 100$) and 2.69% (for $|C| = 100$, $|V| = 50$). The impact of local search (i.e., the improvement relevant to the RTS heuristic) was most pronounced for lower values of $|V|$.

$\lvert C\rvert$	$\lvert V\rvert$	TSP Obj	TSP Time	RTS Obj	RTS Time	RTS Gap	RTS+LS Obj	RTS+LS Time	RTS+LS Gap
50	50	586.62	0.213	450.63	0.663	0.2318	438.89	147.167	0.2518
50	75	569.53	0.215	416.22	1.556	0.2692	408.16	328.837	0.2833
50	100	573.17	0.204	397.88	2.708	0.3058	392.71	471.095	0.3148
75	50	683.60	0.779	572.83	1.050	0.1620	555.98	642.732	0.1867
75	75	687.499	0.672	532.05	2.318	0.2261	521.72	1134.411	0.2411
75	100	681.65	0.687	501.43	4.466	0.2644	492.62	1945.657	0.2773
100	50	784.97	1.799	695.48	1.462	0.1140	674.39	1599.216	0.1409

Table 6.1: Computational results for the MVDRP.

6.8 Using A-Star in Place of Dijkstra's Algorithm

In the A-star Algorithm, the label for a vertex x may be written as $f(x) = g(x) + h(x)$. The component $g(x)$ is the path with shortest known duration from the origin to vertex x, which is the same as the label found in Dijkstra's Algorithm. The component $h(x)$ is a lower bound on the amount of time to traverse from vertex x to the destination.

If for each $loc_i, loc_j \in V$, $t_d(loc_i, loc_j) \leq t_t(loc_i, loc_j)$, we may compute a valid value of h with a simple expression. For each $v_i \in V$ and for each $k = 0, 1, ..., |C|$, we define:

$$h(v'_{i,k}) = t_d(v_i, p_{k+1}) + \sum_{l=k+1}^{|C|} (t_d(p_l, p_{l+1})).$$

The idea is that if the drone is at location v_i, the remaining route duration after k packages have been delivered is, at minimum, the amount of time it takes the drone to fly directly from v_i to p_{k+1}, directly from p_{k+1} to p_{k+2}, directly from p_{k+2} to p_{k+3}, and so on, until $p_{|C|+1} = depot$.

If $\exists loc_i, loc_j \in V$, such that $t_d(loc_i, loc_j) > t_t(loc_i, loc_j)$, we may compute h in the following way.

$$h(v'_{i,k}) = min_{v_b \in V}(t_t(v_i, v_b) + t_d(v_b, p_{k+1}), t_d(v_i, p_{k+1}))+$$

$$\sum_{l=k+1}^{|C|} min_{v_a,v_b}(t_d(p_l, v_a) + t_t(v_a, v_b \in V) + t_d(v_b, p_{l+1}), t_d(p_l, p_{l+1}))).$$

The drone path between consecutive package locations p_l and p_{l+1} may be direct, hence the term $t_d(p_l, p_{l+1})$. Alternatively, it may be faster for the drone to fly from p_l to v_a, ride on the truck from v_a to v_b, then fly from v_b to p_{l+1}.

Chapter 7: Contributions and Future Research

7.1 Contributions

This dissertation explored operational models that require synchronization between a drone and another vehicle. For the vehicle routing problem with drones (VRPD), a model that allows multiple trucks each of which may launch multiple drones, we established a number of theoretical worst-case bounds. These bounds state the maximum speed-up potential utilizing this models under an ideal geometry. We also showed that the VRPD may be viewed as an intermediate problem between the min-max vehicle routing problem and the close-enough vehicle routing problem.

For the traveling salesman problem with drone (TSP-D), we constructed an exact solution method based on the combination of branch-and-bound and dynamic programming. Additionally, several fast heuristics were presented and the quality of the solutions was compared against the optimal solutions.

We introduced the mothership and drone routing problem (MDRP), the high capacity mothership and drone routing problem (MDRP-HC), and the mothership and drone routing problem with obstacles (MDRP+O). As far as we are aware, these problems are new contributions to the literature on drone routing, as the launching vehicle is capable of moving in continuous space and is not restricted to the street network. For MDRP and MDRP-HC, we found that second order cone programming was an efficient embedded procedure to find the optimal launch and landing locations. By embedding second order cone programs in branch-and-bound, we were able to find optimal solutions for the MDRP and MDRP-HC. We proposed greedy procedures and found that using the optimal Euclidean TSP solution for the order to visit targets generally provided high-quality solutions with significantly less computational time and was computationally tractable even for large instances. In the case of MDRP+O, we proposed a sequential second order cone program. We computed a circle of maximum radius around each launch and landing point of the incumbent solution such that the circle does not intersect with any obstacle. We optimally choose each launch and landing point from within those circles to form a new incumbent solution.

We introduced the multi-visit drone routing problem (MVDRP). In the MVDRP, a truck and drone work in tandem to deliver packages. Unlike previous problems in the literature, we (1) allowed the drone to visit multiple customers consecutively, (2) allowed the user to define an arbitrary increasing function (of weight) for the energy drain of the drone, and (3) decoupled the set of potential launch and landing locations from the set of customer locations. We presented heuristic solution methods that found high-quality solutions.

We showed that tandems combining one or more drones with a ship or truck may be complementary. By combining the larger capacity of a ship or truck with the mobility of one or more drones, we demonstrated theoretically and computationally that the time or cost required to visit all required targets or customers may be reduced.

7.2 Future Work

Aside from the VRPD model, we used a single drone in our models. We believe generalizing the MDRP and MVDRP to the case of multiple drones and/or multiple trucks merits consideration. We plan to explore other generalizations, such as those that allow the optimization of drone speed or allow a drone to launch along an arc of a graph.

The economics and practical application of drones are highly dependent on physical parameters and specifications of drones. Further testing and tuning of model parameters may yield insights about critical factors and sensitivities in the design of drones for different applications.

The study of operations related to drone technology is an exciting field with a rapidly expanding set of applications. In addition to questions that we can see on the horizon, there are many unknowns just beyond the horizon that will surely shape the trajectory of drone research moving forward.

Bibliography

[1] S. Adams and C. Friedland. "A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management.", *9th International Workshop on Remote Sensing for Disaster Response.* 2011.

[2] N. Agatz, P. Bouman, and M. Schmidt. "Optimization Approaches for the Traveling Salesman Problem with Drone." to appear in *Transportation Science.*

[3] J. Smith. "Exclusive: Afghan drone war - data show unmanned flights dominate air campaign." Reuters. April 20, 2016. Accessed: May 15, 2018. `https://www.reuters.com/article/us-global-markets/asian-shares-falter-as-u-s-readies-china-tariffs-euro-at-2-week-low-on-ecb-idUSKBN1JB03U`

[4] B. Albaker, N. Rahim. "A survey of collision avoidance approaches for unmanned aerial vehicles." Technical Postgraduates (TECHPOS) International Conference, IEEE, 2009.

[5] Amazon. "Amazon's Best of Prime 2017 Reveals the Years Biggest Trends More than 5 Billion Items Shipped with Prime in 2017". January 2, 2018. Press Release. `http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-newsArticle&ID=2324426`

[6] Amazon Prime Air. "Amazon Prime Air: Frequently Asked Questions." Accessed: April 9, 2018. `https://www.amazon.com/b?node=8037720011`

[7] Amazon YouTube Channel. "Amazon Prime Air." Amazon. November 29, 2015. Accessed: April 9, 2018. Note: YouTube video. `https://www.youtube.com/watch?v=MXo_d6tNWuY`

[8] C. Attanasio. "Drones Offer New Advantages for Police." GovTech.com. January 8, 2018. Accessed: March 5, 2018. http://www.govtech.com/public-safety/Drones-Offer-New-Advantages-for-Police.html

[9] G. Avellar, G. Pereira, L. Pimenta, and P. Iscold. "Multi-uav routing for area coverage and remote sensing with minimum time." *Sensors*, 15.11 (2015): 27783-27803.

[10] A. Barrientos, J. Colorado, J. Cerro, A. Martinez, C. Rossi, D. Sanz, and J. Valente. "Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots." *Journal of Field Robotics*, 28.5 (2011): 667-689.

[11] L. Babel. "Curvature-constrained traveling salesman tours for aerial surveillance in scenarios with obstacles." *European Journal of Operational Research.* 262.1 (2017): 335-346.

[12] J. Bezos. Interview with Charlie Rose. Broadcasted on: 60 Minutes, CBS. December 1, 2013. Transcript: https://www.cbsnews.com/news/amazons-jeff-bezos-looks-to-the-future/

[13] R. Bellman. "On a routing problem." *Quarterly of Applied Mathematics.* 16.1 (1958): 87-90.

[14] BI Intelligence Unit, *Drones could help protect US borders* Business Insider. April 10, 2017. Accessed: March 5, 2018. http://www.businessinsider.com/drones-could-help-protect-us-borders-2017-4

[15] Boeing YouTube Channel. "Future of autonomous air travel: Boeing unveils new cargo air vehicle prototype." Boeing. January 10, 2018. Accessed: April 29, 2018. Note: YouTube video. https://www.youtube.com/watch?v=uluR-dNUPvI

[16] M. Borak, *World's top donre seller DJI made $2.7 billion in 2017*, TechNode. January 3, 2018. Accessed: April 7, 2018. https://technode.com/2018/01/03/worlds-top-drone-seller-dji-made-2-7-billion-2017/

[17] J. Bowen, *Fayetteville police use drones to chase suspects, find the missing* WRAL. April 4, 2018. Accessed: April 7, 2018. http://www.wral.com/drones-to-help-fayetteville-police-nab-fleeing-suspects-find-missing-people-/17465062/

[18] A. Washburn and G. Brown. "An exact method for finding shortest routes on a sphere, avoiding obstacles." *Naval Research Logistics.* 63.5 (2016): 374-385.

[19] J. Campbell, D. Sweeney, and J. Zhang. "Strategic Design for Delivery with Trucks and Drones." Technical Report. 2017.

[20] W. P. Coutinho, R. Q. Nascimento, A. A. Pessoa, and A. Subramanian. "A branch-and-bound algorithm for the close-enough traveling salesman problem." *INFORMS Journal on Computing.* 28.4 (2016): 752-765.

[21] E. Dijkstra. "A note on two problems in connexion with graphs." *Numerische mathematik* 1.1 (1959): 269-271.

[22] M. Franco. "DHL uses completely autonomous system to deliver consumer goods by drone." New Atlas. May 10, 2016. Accessed: April 19, 2018. `https://newatlas.com/dhl-drone-delivery/43248/`

[23] P. Rey. "Paketzustellung per Drohne: DPDgroup startet den weltweit ersten Drohnenverkehr im Linienbetrieb" (Package delivery with a drone: DPDgroup starts the first regular drone service worldwide). Press Release DPDgroup (2016). `https://www.dpd.com/portal_de/home/news/aktuelle_neuigkeiten/dpdgroup_startet_den_weltweit_ersten_drohnenverkehr_im_linienbetrieb`

[24] L. Dubins. "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents." *American Journal of mathematics.* 79.3 (1957): 497-516.

[25] R. Ellis and J. Newsome, *Drone, social media make flood resuce happen in real time*, CNN. January 18, 2018. Accessed: April 2, 2018. `https://www.cnn.com/2016/10/11/us/weather-matthew-drone-rescue/index.html`

[26] C. Engelking, *Drone Pilot Helps Rescue Flood Victims in Texas* Discover Magazine. May 21, 2015. Accessed: April 7, 2018. `http://blogs.discovermagazine.com/drone360/2015/05/21/drone-rescue-flood-victims/`

[27] A. Ferris-Rotman, *How Drones Are Helping Nepal Recover From The Earthquake* Huffington Post. May 7, 2015. Accessed: March 28, 2018. `https://www.huffingtonpost.com/2015/05/07/nepal-earthquake-drones_n_7232764.html`

[28] B. Fessier, *Coachella using drones, security plans enacted after 9/11 to prevent another Las Vegas-type incident* April 4, 2018. Accessed: April 7, 2018. `https://www.desertsun.com/story/life/entertainment/music/coachella/2018/04/04/coachella-using-drones-security-plans-enacted-after-9-11-prevent-another-las-vegas-type-incident/487502002/`

[29] J. Fisher, *The Best Drones of 2018* PC Magazine. March 12, 2018. Accessed: April 3, 2018. https://www.pcmag.com/roundup/337251/the-best-drones

[30] N. Gageik, T. Mller, and S. Montenegro. "Obstacle detection and collision avoidance using ultrasonic distance sensors for an autonomous quadrocopter." University of Wrzburg, Aerospace Information Technology (Germany). Wrzburg September (2012).

[31] Gurobi Optimization Inc. "tsp.py." 2017. http://www.gurobi.com/documentation/7.5/examples/tsp_py.html Note: Python Code. Accessed: March 1, 2018.

[32] A.L. Henry-Labordere. "The record balancing problem: a dynamic programming solution of a generalized traveling salesman problem." *Revue Francaise D Informatique De Recherche Operationnelle.* 3.2 (1969): 43-49.

[33] D. Jenkins and B. Vasigh, *The Economic Impact of Unmanned Aircraft Systems in the United States* Association for Unmanned Vehicle Systems International. 2013.

[34] C. Goerzen, Z. Kong, and B. Mettler. *A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance* Journal of Intelligent Robotic Systems (2010) 57: 65. DOI: https://doi.org/10.1007/s10846-009-9383-1

[35] B. Golden, S. Raghavan, and E. Wasil (Eds.). *The vehicle routing problem: latest advances and new challenges* (Vol. 43). Springer Science and Business Media. 2008.

[36] C. Greenfield. "An analysis of U.S. Navy humanitarian assistance and disaster relief operations". Dissertation. Retrieved from: https://calhoun.nps.edu/bitstream/handle/10945/10769/11Jun_Greenfield_MBA.pdf?sequence=1. 2011.

[37] Q. Ha, Y. Deville, Q. Pham, and M. H. "On the min-cost traveling salesman problem with drone." *Transportation Research Part C: Emerging Technologies.* 86 (2018): 597-621.

[38] L. Kuo, *Africa is now the worlds testing ground for commercial drones* Quartz. June 24, 2017. Accessed: March 8, 2018. https://qz.com/1003810/the-worlds-first-commercial-drone-delivery-operates-from-a-hill-in-rwanda/

[39] K. Leary, *Matternet Is Expanding Drone Delivery to Hospitals in Switzerland* Futurism. September 27, 2017. Accessed: April 2, 2018. https://futurism.com/matternet-is-expanding-drone-delivery-to-hospitals-in-switzerland/

[40] B. Lillian, *Flying Drones Beyond Line of Sight: How Can We Get There?* Unmanned Aerial. May 10, 2017. Accessed: March 22, 2018. https://unmanned-aerial.com/flying-drones-beyond-line-sight-can-get

[41] L. Lin and M. Goodrich. "Hierarchical heuristic search using a Gaussian mixture model for UAV coverage planning." *IEEE Trans. Cybernet.* 44 (2014), 25322544.

[42] B. Kernighan and S. Lin. "An efficient heuristic procedure for partitioning graphs." *The Bell System Technical Journal.* 49.2 (1970): 291-307.

[43] M. S. Lobo, L. Vandeberghe, S. Boyd, and H. Lebret. "Applications of second-order cone programming." *Linear Algebra and its Applications.* 284.1-3(1998): 193-228.

[44] S. Manyam, S. Rathinam, and S. Darbha. "Computation of lower bounds for a multiple depot, multiple vehicle routing problem with motion constraints." *Journal of Dynamic Systems, Measurement, and Control.* 137(9): 094501-094505.

[45] A. Meola, *Drone market shows positive outlook with strong industry growth and trends* Business Insider. July 13, 2017. Accessed: March 6, 2018. http://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts-2017-7

[46] G. Mitchell, *Facebook's Internet-delivering drone takes flight* June 30, 2017. Accessed: April 2, 2018. https://www.usatoday.com/story/tech/nation-now/2017/06/30/facebooks-internet-delivering-drone-takes-flight/444559001/

[47] R. Monteiro and T. Tsuchiya. "Polynomial convergence of primal-dual algorithms for the second-order cone program based on the MZ-family of directions." *Mathematical Programming.* 88.1 (2000): 61-83.

[48] T. Mori and S. Scherer. "First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles." International Conference on Robotics and Automation, 2013. IEEE.

[49] C. Murray and A. Chu. “The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery.” *Transportation Research Part C: Emerging Technologies*. 54: 86-109.

[50] A. Nedjati, G. Izbirak, B. Vizvari, and J. Arkat. “Complete coverage path planning for a multi-UAV response system in post-earthquake assessment.” *Robotics* 5.4 (2016): 26-41.

[51] A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch. ”Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey”. To appear in *Networks.*

[52] F. Planer and T. Martin. “Panama’s Indigenous Technicians Are Using Drones to Fight Illegal Logging.” Vice Magazine. February 2018. Accessed: April 7, 2018. Web version: `https://www.vice.com/en_uk/article/ez84ka/panamas-indigenous-technicians-are-using-drones-to-fight-illegal-logging`

[53] S. Poikonen, E. Wasil, and B. Golden. “A Branch-and-Bound Approach to the Traveling Salesman Problem with a Drone.” To appear in *Informs Journal on Computing.*

[54] S. Poikonen, X. Wang, and B. Golden. “The vehicle routing problem with drones: Extended models and connections.” *Networks.* 70.1 (2017): 34-43.

[55] Reuters staff. “Finnish post office tests drone for parcel delivery.” September 14, 2015. Accessed: June 11, 2018. Reuters. `https://www.reuters.com/article/us-finland-postaldrone-iduskcn0re15e20150914`

[56] M. Raap, M. Zsifkovits, and S. Pickl, “Trajectory optimization under kinematical constraints for moving target search.” *Computers and Operations Research.* 88 (2017): 324331.

[57] Reuters staff. “Russian postal drone program hits wall in debut.” Reuters. April 2, 2018. Accessed: April 16, 2018. `https://www.reuters.com/article/us-russia-post-drone-crash/russian-postal-drone-program-hits-wall-in-debut-idUSKCN1H91B4`

[58] H. Savuran and M. Karakaya. “Efficient route planning for an unmanned air vehicle deployed on a moving carrier.” *Soft Computing.* 20.7 (2016): 2905-2920.

[59] J. Silberholz and B. Golden. “The generalized traveling salesman problem: A new genetic algorithm approach.” Extending the horizons: advances in computing, optimization, and decision technologies. Springer, Boston, MA, 2007. 165-181.

[60] P. Toth and D. Vigo (Eds.). *The vehicle routing problem.* Society for Industrial and Applied Mathematics. 2002.

[61] N. Cary and N. Bose. "UPS, FedEx and Amazon gather flight data to prove drone safety." Venture Beat. September 24, 2016. Accessed: May 17, 2017. https://venturebeat.com/2016/09/24/ups-fedex-and-amazon-gather-flight-data-to-prove-drone-safety/

[62] UPS YouTube Channel. "UPS Tests Residential Delivery Via Drone." Feburary 2017. Accessed: March 2017. https://www.youtube.com/watch?v=\xx9_6OyjJrQ Note: YouTube Video.

[63] J.P. Vielma and G. Nemhauser. "Modeling disjunctive constraints with a logarithmic number of binary variables and constraints." *Mathematical Programming* 128.1-2 (2011): 49-72.

[64] D. Walsh and E. Schmitt, "U.S. Strikes Al Qaeda Target in Southern Libya, Expanding Shadow War There." New York Times. March 25, 2018.

[65] X. Wang, S. Poikonen, and B. Golden. "The vehicle routing problem with drones: Several worst-case results." *Optimization Letters.* 11.4 (2017): 679-697.

[66] Project Wing. "Project Wing." Accessed: June 15, 2018. https://x.company/projects/wing/

[67] Zipline, Inc. "Tanzania Announces World's Largest National Drone Delivery Network Partnering with Zipline". August 24, 2017. Accessed: April 4, 2018. Note: Press Release. http://www.flyzipline.com/uploads/Tanzania%20Announcement%20Press%20Release%20vFinal.pdf