The Mothership and Drone Routing Problem

Stefan Poikonen[1] and Bruce Golden[2]

[1]Business School, University of Colorado Denver, Denver, CO 80202, USA - stefan.poikonen@ucdenver.edu
[2]R.H. Smith School of Business, University of Maryland, College Park, MD 20742, USA - bgolden@rhsmith.umd.edu

August 18, 2018

Abstract

The Mothership and Drone Routing Problem (MDRP) considers the routing of a two vehicle tandem. The larger vehicle, which may be a ship or an airplane, is called the mothership; the smaller vehicle, which may be a small boat or unmanned aerial vehicle (UAV), is called the drone. We assume there exists a set of target locations T. For each $t \in T$, the drone must launch from the mothership, visit t, then return to the mothership to refuel. The drone has a limited range of R time units. In MDRP, we assume both mothership and drone operate in the "open seas" (i.e., utilizing the Euclidean metric). We also introduce the Mothership and Infinite Capacity Drone Problem (MDRP-IC), where a drone launches from the mothership, visits one or more targets consecutively, before returning to the mothership.

Our exact approach utilizes branch-and-bound, where each node of the branch-and-bound tree corresponds to a potential subsequence of the order of target visits. A lower bound at each node is given by solving a second order cone program, which optimally chooses a launch point and landing point for each target in the subsequence. A set of heuristics, which also use a second order cone program as an embedded procedure, is presented. We show that our schemes are flexible to accomodate a variety of additional constraints and/or objective functions. Computational results and interesting variants of MDRP and MDRP-IC are also presented.

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 [17]. Amazon, Google, UPS, and DHL [4, 20, 7] 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 [16, 1, 8], 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-IC, and associated solution methods. Section 8 provides computational results for MDRP-IC. Section 9 discusses the flexibility of our solution methods, future research, and conclusions.

2 Literature Review

In 2015, Murray and Chu [16] introduced the Flying Sidekick Traveling Salesman Problem (FSTSP). In the 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. [1] 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. [19] adapt the partitioning procedure of Agatz et al. [1], and use it as an embedded procedure within each node of a branch-and-bound tree to produce optimal solutions. In [19], a divide-and-conquer technique is applied to break a larger master problem into a sequence of smaller subproblems to increase computational speed. Campbell, Sweeney, and Zhang [3] use continuous approximation to help compute expected delivery costs. Ha et al. [8] introduce a greedy randomized adaptive search procedure (GRASP) to generate solutions to TSP-D.

In Wang et al. [23] and Poikonen et al. [18], 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.

Ham [9] applies constraint programming to find efficient solutions to a multi-depot, multi-truck, and multi-drone model where the drones are capable of both dropping off and picking up parcels. Campbell, Corberán, et al. [2] explore drone arc routing problems, their relationship with postman arc routing problems, and consider the possibility of the drone not needing to service an entire arc, because they are not constrained to a street network. A solution method is described that iteratively generates and solves rural postman problems. Ulmer and Thomas [22] study the same-day delivery routing problem with a heterogeneous fleet (SDDPHF) for the case where there exists a ground-based fleet of vehicles in addition to a drone fleet. Their goal was to identify a heuristic policy for assigning packages (to ground-based vehicle or drone) and subsequently routing vehicles. They found that a policy, where customers beyond a threshhold distance from the depot were serviced by drone and those near the depot were serviced by the truck, performed favorably relative to other benchmark policies. Dayarian and Savelsbergh [6] also consider drone assisted same-day delivery. However, rather than using the drone to service the last mile segment, a drone may be used to resupply a truck that is away from the depot.

In Coutinho et al. [5], 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 [5]. For the curious reader, the work of Lobo et al. [13] 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 [13] 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 [15]. The key takeaway from [13] and [15], 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 [21], 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 [21], 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.

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*.

Each launch and rendezvous location may be chosen from the Euclidean plane. Times when a drone launches, times when a drone lands, and flight durations are continuous variables, not restricted to discrete time intervals.

The mothership and drone do not need to arrive at a rendezvous location at the same time; the faster arriving vehicle may wait for the other at the rendezvous location. However, in all cases, the drone may not be separated from the mothership for more than R consecutive time units due to limited battery capacity. 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 mothership and the drone must travel together from *orig* to the first launch location. Similarly, after the drone visits the last target location, the mothership and the drone must meet at the final rendezvous location before traveling together back to *dest*. The first launch location and final rendezvous location are allowed to be *orig* and *dest*, respectively, but it is not required. 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 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 clock does not stop until both vehicles have returned to *dest*. The MDRP is a generalization of the Euclidean Traveling Salesman Problem. In Figure 1, we display an example solution path for the MDRP with four targets. We point out the following result.

Theorem 1. *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 1 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 1 is valid, because, at worst, the ship may travel to each target location $T \cup \{orig\}$ and launch the drone at negligible distance from the target.

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.

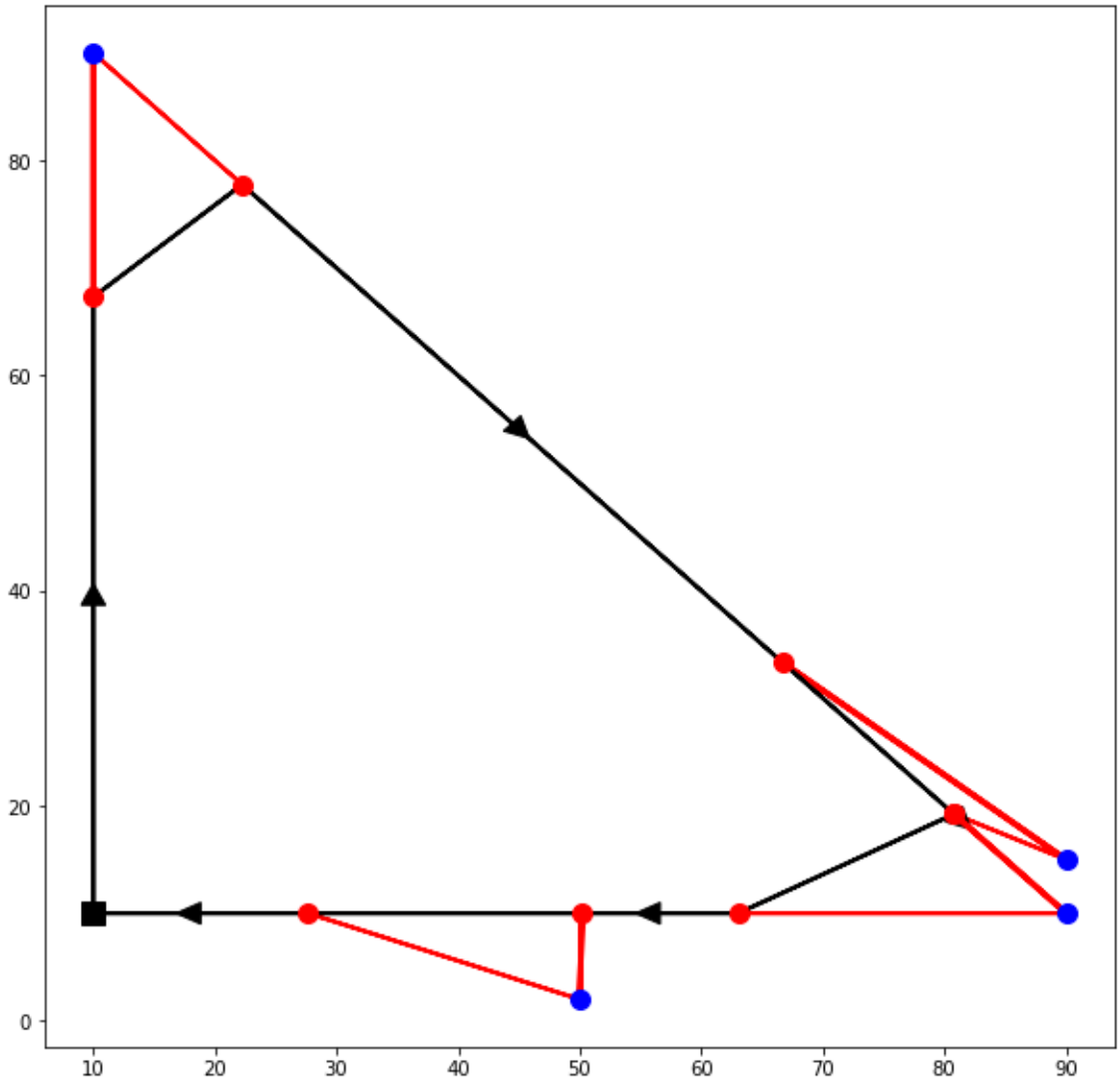

Figure 1: An example solution path for the MDRP with $R = 20$ and $\alpha = 2$. Black line segments trace the path of the mothership. Red line segments trace the flight path of the drone. Blue circles are target locations. Red circles are locations where the drone launches from or returns to the ship.

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 `LENSEQ`(S).

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

`LENSEQ`(S):

$$minimize(\sum_{k=0}^{n+1}(cTime(k) + sTime(k)) \tag{A1}$$

`Subject to:`

`For k=0 to n:`

$$\|lPoint(k+1) - rPoint(k)\| \leq cTime(k) \tag{A2}$$

$$\|lPoint(k) - rPoint(k)\| \leq sTime(k) \tag{A3}$$

$$\|s_k - lPoint(k)\| \leq outFlightDist(k) \tag{A4}$$

$$\|s_k - rPoint(k)\| \leq inFlightDist(k) \tag{A5}$$

$$(outFlightDist(k) + inFlightDist(k))/\alpha \leq sTime(k) \tag{A6}$$

$$sTime(k) \leq R \tag{A7}$$

`End For`

$$lPoint(0) = orig \tag{A8}$$

$$rPoint(0) = orig \tag{A9}$$

$$lPoint(n+1) = dest \tag{A10}$$

$$rPoint(n+1) = dest \tag{A11}$$

Objective (A1) 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 (A2) ensures that $cTime(k)$ is at least as large as the mothership travel time from $rPoint(k)$ to $lPoint(k+1)$. Constraint (A3) ensures that $sTime(k)$ is at least as large as the travel time of the mothership from $lPoint(k)$ to $rPoint(k)$. Together, constraints (A4), (A5), and (A6) ensure that $sTime(k)$ is at least as large as

Name	Var./Fixed	Domain	Meaning
n	Fixed	$\mathbb{N}$	Number of target locations
s_k	Fixed	$\mathbb{R}^2$	Location of the kth target location of fixed sequence S
R	Fixed	$(0, \infty)$	Maximum flight time of drone
α	Fixed	$(1, \infty)$	Speed of drone
$cTime(k)$	Var.	$\mathbb{R}^2$	Combined time between s_k landing and s_{k+1} launch
$sTime(k)$	Var.	$\mathbb{R}^2$	Separated time while drone airborne to/from target s_k
$lPoint(k)$	Var.	$\mathbb{R}^2$	Location where drone launches before flying to s_k
$rPoint(k)$	Var.	$\mathbb{R}^2$	Location where drone is retrieved after flying to s_k
$outFlightDist(k)$	Var.	$[0, \infty)$	Distance of the outbound flight from $lPoint(k)$ to s_k
$inFlightDist(k)$	Var.	$[0, \infty)$	Distance of the inbound flight from s_k to $rPoint(k)$

Table 1: A table summarizing the decision variables and parameters of `LENSEQ(`S`)`.

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 (A7) ensures the drone is retrieved before its maximum flight time has elapsed. Constraints (A8) through (A11) 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 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.

In Table 1, for each decision variable and parameter in `LENSEQ(S)`, we provide its name (Name), whether it is a decision variable (Var.) or a fixed parameter (Fixed), its domain of possible values, and a brief description (Meaning). We emphasize that each s_k is fixed for a given sequence of target visitations S.

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

4.2 Branch-and-Bound: Finding the Best Sequence

Our solution method is predicated upon the following theorem.

Theorem 2. *If S_1 is a subsequence of S_2, then $lenSeq(S_1) \leq lenSeq(S_2)$.*

Theorem 2 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 2. 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.

The structure of this branch-and-bound tree is similar to that in Poikonen et al. [19] for the TSP-D and Coutinho et al. [5] for the CETSP. However, [19] applies a dynamic program rather than a second order cone program at each node of the branch-and-bound tree.

ALGBAB:

$\texttt{tree} = \emptyset$ $(B1)$

$\texttt{rootNode.sequence} \leftarrow [orig, t_1, dest]$ $(B2)$

$\texttt{rootNode.lowerBound} \leftarrow lenSeq(\texttt{rootNode.sequence})$ $(B3)$

$\texttt{rootNode.upperBound} \leftarrow \infty$ $(B4)$

$\texttt{rootNode.parent} \leftarrow \texttt{none}$ $(B5)$

$\texttt{rootNode.isLeaf} \leftarrow \texttt{true}$ $(B6)$

$\texttt{tree.add(rootNode)}$ $(B7)$

$\texttt{while(rootNode.upperBound > rootNode.lowerBound):}$ $(B8)$

  $\texttt{currNode} \leftarrow min_{node \in \texttt{tree} | \texttt{node.isLeaf=true}}(\texttt{node.lowerBound})$ $(B9)$

  $\texttt{newTarget} \leftarrow max_{t \in T}(min_{s \in \texttt{currNode.sequence}}(distance(s,t))$ $(B10)$

  $\texttt{For position from 1 to length(currNode.sequence):}$ $(B11)$

    $\texttt{newNode.sequence} \leftarrow insert(\texttt{currNode.sequence,newTarget,position})$ $(B12)$

    $\texttt{newNode.lowerBound} \leftarrow lenSeq(\texttt{newNode.sequence})$ $(B13)$

    $\texttt{If}(\forall t \in T, t \in \texttt{newNode.sequence}):$ $(B14)$

      $\texttt{newNode.upperBound} \leftarrow \texttt{newNode.lowerBound}$ $(B15)$

    $\texttt{Else:}$ $(B16)$

      $\texttt{newNode.upperBound} \leftarrow \infty$ $(B17)$

    $\texttt{newNode.isLeaf} \leftarrow \texttt{true}$ $(B18)$

    $\texttt{newNode.parent} \leftarrow \texttt{currNode}$ $(B19)$

    $\texttt{currNode.children.add(newNode)}$ $(B20)$

    $\texttt{tree.add(newNode)}$ $(B21)$

  $\texttt{currNode.isLeaf} \leftarrow \texttt{false}$ $(B22)$

  $\texttt{currNode.lowerBound} \leftarrow min_{\texttt{child} \in \texttt{currNode.children}}(\texttt{child.lowerBound})$ $(B23)$

  $\texttt{currNode.upperBound} \leftarrow min_{\texttt{child} \in \texttt{currNode.children}}(\texttt{child.upperBound})$ $(B24)$

  $\texttt{while}(\texttt{currNode.parent} \neq \texttt{none}):$ $(B25)$

    $\texttt{currNode} \leftarrow \texttt{currNode.parent}$ $(B26)$

    $\texttt{currNode.lowerBound} \leftarrow min_{\texttt{child} \in \texttt{currNode.children}}(\texttt{child.lowerBound})$ $(B27)$

    $\texttt{currNode.upperBound} \leftarrow min_{\texttt{child} \in \texttt{currNode.children}}(\texttt{child.upperBound})$ $(B28)$

In (B1) to (B7) 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 (B8), 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 (B9).

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` (B10).

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 (B12). 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`) (B13).

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` (B14, B15), because this represents a feasible solution to the overall problem that visits each $t_i \in T$. Otherwise, set `child.upperBound` $= \infty$ (B16, 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` (B18, B19, B20, B21).

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 (B22, B23, B24, B25, B26, B27, B28).

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

This branch-and-bound approach (BAB) is an exact approach, because Theorem 2 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. Once the upper bound and lower bound of the root node are identical, to extract the optimal sequence, we begin at the root node and iteratively descend through the tree, choosing the child node with the smallest lower bound at each step until we have reached a leaf node at the bottom of the branch-and-bound tree. We then apply `LENSEQ` to the sequence corresponding with that leaf node to determine optimal launch and landing points. Thus, we do not need to store in memory the optimal launch and landing locations for every node of the branch-and-bound tree.

In Figure 2, we display the branch-and-bound tree for a small instance with three targets. Next to each node in Figure 2 is its associated sequence. The lower bound of a node with associated sequence S is initially computed as $lenSeq(S)$.

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.

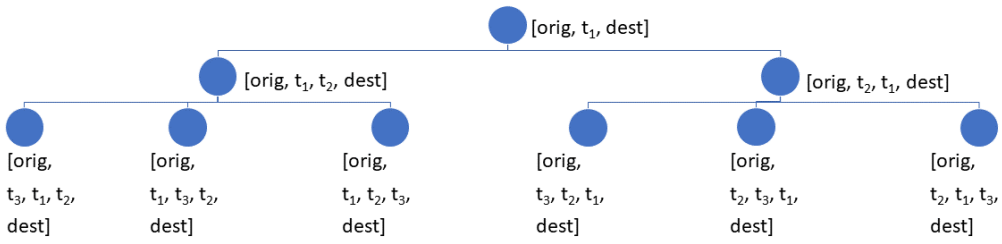

Figure 2: A branch-and-bound tree that explores all sequences for visiting targets t_1, t_2, and t_3.

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 $lenSeq(TSPSeq)$.

5.2 Greedy Sequence with Local Search

In the Greedy Sequence with Local Search (GSLS) heuristic, we begin by initializing the value of `currSeq` to be the Euclidean TSP solution on the set of locations $\{orig\} \cup T$.

The optimal Euclidean TSP solution is not necessarily the best visit order for MDRP. Thus, we wish to construct a neighborhood of similar sequences to `currSeq` by slightly modifying or perturbing the order. We then will evaluate the quality of each neighbor sequence.

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 move.

After initializing `currSeq`, we perform an iterative process. In the first step of the iterative process, we compute the objective value associated with `currSeq` (i.e., $lenSeq$(`currSeq`)). For each neighbor in $neighborhood$(`currSeq`), we will also apply the second order cone program `LENSEQ` to compute an MDRP objective value. If any neighbor sequence produces a better objective value than $lenSeq$(`currSeq`), then `currSeq` will be replaced by the neighbor sequence that produced the best objective value, and a new iteration will occur, where a neighborhood is constructed around the updated `currSeq`. If there is no neighbor that produces a better solution than `currSeq`, then the algorithm terminates.

Pseudocode for this local search algorithm is labeled `ALGGSLS`.

```
ALGGSLS:
currSeq ← TSPSeq                                          (C1)
contLocalSearch ← true                                    (C2)
While(contLocalSearch=true)                               (C3)
   contLocalSearch ← false                                (C4)
   bestObjVal ← lenSeq(currSeq)                           (C5)
   For each neighbor in neighborhood(currSeq):            (C6)
      objVal ← lenSeq(neighbor)                           (C7)
      If(objVal<bestObjVal):                              (C8)
         bestObjVal ← objVal                              (C9)
         bestSeq ← neighbor                               (C10)
   If(bestSeq ≠ currSeq):                                 (C11)
      currSeq ← bestSeq                                   (C12)
      contLocalSearch ← true                              (C13)
```

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

In our computational experiments, we found I to be small. Furthermore, we suspect that it is possible to substantially reduce the number of neighbors considered (and, therefore, the running time) without increasing the resulting objective value by very much. We hope to explore this idea in future work.

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. In Phase 1, we are effectively solving using the BAB algorithm,

but only on the subset of target locations, T_p, instead of the full set of target locations, T. The solution path from Phase 1 is then labeled $bestPartialSeq$, which represents the best possible solution for the subset of target locations T_p.

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:
currSeq ← bestPartialSeq                                   (D1)
For each t_i ∈ T \ T_p :                                   (D2)
   bestObjVal ← ∞                                          (D3)
   For each position = 2 to |bestPartialSeq|-1:            (D4)
      trialSeq ← insert(currSeq, t_i, position)            (D5)
      objVal ← lenSeq(trialSeq)                            (D6)
      If objval < bestObjVal:                              (D7)
         bestObjVal ← objVal                               (D8)
         nextSeq ← trialSeq                                (D9)
   currSeq ← nextSeq                                       (D10)
```

The idea of Phase 2 is that we will insert the remaining target locations, $T \setminus T_p$, one at time. For each $t_i \in T \setminus T_p$, we try to insert t_i into every possible position of `currSeq`. For each possible insertion position, we construct a trial sequence, `trialSeq`, which is formed by inserting t_i into `currSeq` in the given insertion position. We then compute $lenSeq$(`trialSeq`), which uses the second order cone program to compute the objective value that results if we inserted t_i in the given position. After all insertion positions have been tried, we then permanently insert t_i in the position that yielded the lowest objective value. We repeat until all remaining targets have been inserted.

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 [10], where violated subtour 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.

6.1 Comparing Solution Methods for MDRP

In Table 2 and Table 3, 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 2, 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 2 as the uniform instances. In Table 3, 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 3 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		
$\|T\|$	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 2: Computational results for the MDRP on uniformly distributed instances.

	TSP	BAB				GS			GSLS			PSGI		
$\|T\|$	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	277.003	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 3: Computational results for the MDRP on clustered instances.

6.2 Analysis of MDRP Computational Results

In Table 2 and Table 3, 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 3 were computationally more costly than the uniform instances of Table 2. 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 [11]) 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. The runtimes, however, were significantly higher than GS due to applying `LENSEQ` to each of $O(|T|^2)$ neighbors at each iteration, rather than evaluating a second order cone program for only the single initial sequence. In applying GSLS, the number of iterations before termination was never larger than three. Thus, the increased runtime of GSLS (relative to GS) is mostly attributable to the size of neighborhood searched, rather than the number of iterations.

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, because $|T_p|$ is the maximum length of sequence considered in the branch-and-bound tree of PSGI. Thus, if $|T_p|$ is large, then in the branch-and-bound tree of PSGI, many nodes must be evaluated. This growth is similar to the computation time growth of BAB as $|T|$ increases.

7 The Mothership and Infinite 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. This may be because the drone is capable of carrying only a single homogeneous package at a time. 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 infinite capacity drone routing problem (MDRP-IC) in the same way as MDRP, except we now allow the drone to visit multiple targets consecutively before returning to the mothership. There are no capacity constraints on the drone. The drone is assumed to carry very light packages or no packages at all to the targets. However, we continue to require that the drone must not be separated from the mothership for more than R consecutive time units. (Appendix C describes how we may easily modify our approach to the case where a drone has finite carrying capacity.)

Theorem 3. *If $R = \infty$, the solution of the MDRP-IC 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.*

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 3, we display an example solution for the MDRP-IC, which contains three drone 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_j. For example, suppose $S = [orig, t_4, t_2, t_3, t_1, dest]$. Then $compositions(S) =$

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

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.

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 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

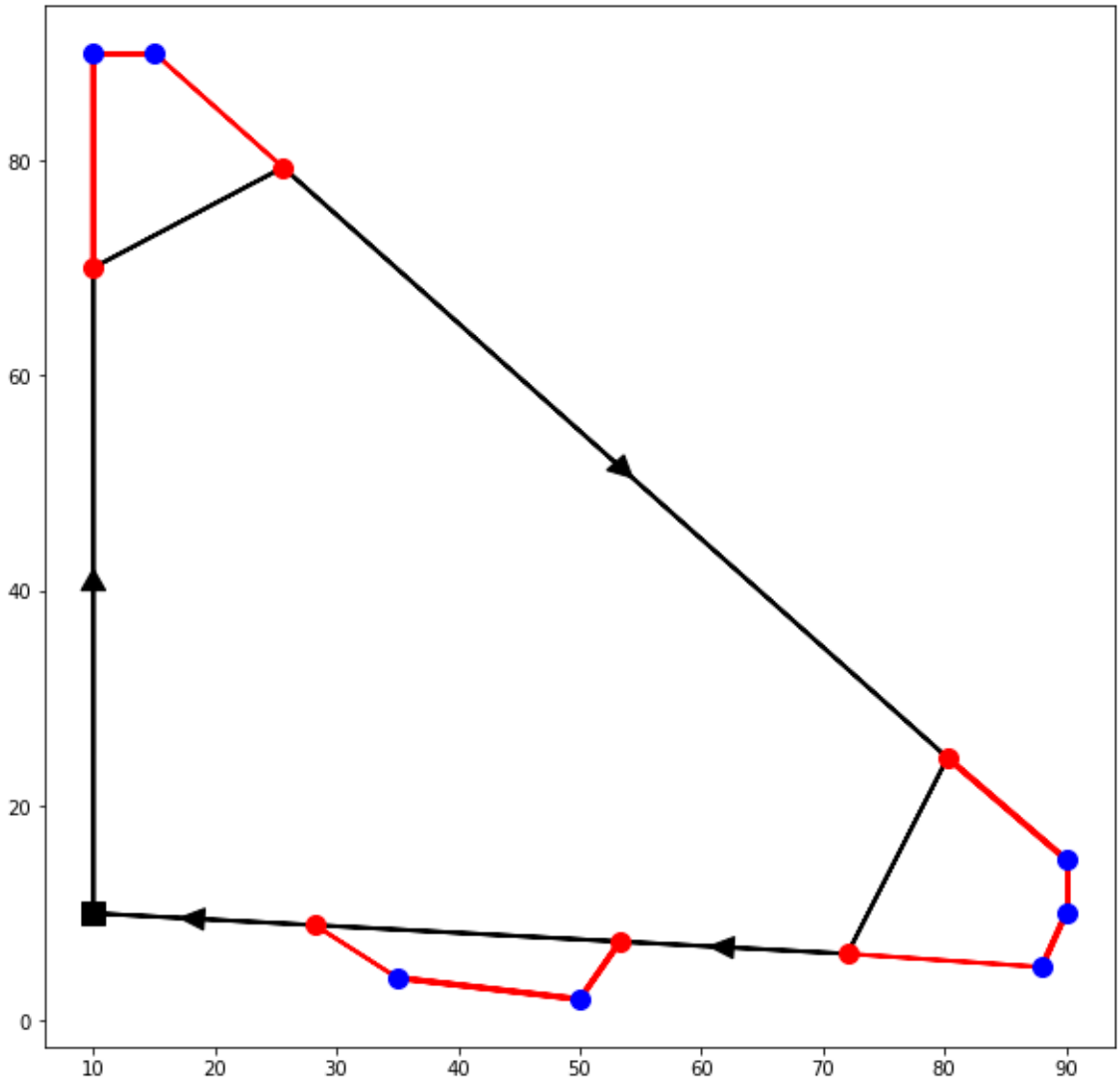

Figure 3: An example solution path for the MDRP-IC with $R = 20$ and $\alpha = 2$. Black line segments trace the path of the mothership. Red line segments trace the flight path of the drone. The black square is location $orig = dest$. Blue circles are target locations. Red circles are locations where the drone launches from or returns to the ship. By applying the `LENCOMP` function, we are optimally choosing locations for the red circles.

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

$$minimize(\sum_{k=0}^{n+1}(cTime(k) + sTime(k)) \quad (E0)$$

`Subject to:`

`For k=0 to m:`

$$\|lPoint(k+1) - rPoint(k)\| \leq cTime(k) \quad (E1)$$
$$\|lPoint(k) - rPoint(k)\| \leq sTime(k) \quad (E2)$$
$$\|first(ST_k) - lPoint(k)\| \leq outFlightDist(k) \quad (E3)$$
$$\|last(ST_k) - rPoint(k)\| \leq inFlightDist(k) \quad (E4)$$
$$len(ST_k) \leq intraFlightDist(k) \quad (E5)$$
$$(outFlightDist(k) + intraFlightDist(k) + inFlightDist(k))/\alpha \leq sTime(k) \quad (E6)$$
$$sTime(k) \leq R \quad (E7)$$

`End For`

$$lPoint(0) = orig \quad (E8)$$
$$rPoint(0) = orig \quad (E9)$$
$$lPoint(m+1) = dest \quad (E10)$$
$$rPoint(m+1) = dest \quad (E11)$$

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 (E5), (E6), and (E7) 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).

7.3 Finding the Best Composition

Section 7.2 describes describes how to optimize MDRP-IC 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.

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 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 $min_{C \in compositions(S)}(lenComp(C))$. That is, at each node with sequence S, we search for the composition of S that produces the lowest objective value.

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 and lower bound of the root node. We then return the best composition of the leaf node with the lowest lower bound as our solution.

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.

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.

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.

In Figure 4, we illustrate the GSGC and GSGC+S solution methods on the same instance. This example showcases why GSGC+S may produce better objective values on some instances. In the GSGC solution, all three targets are in the same drone subtour. Flight from the first target to the second target and from the second target to the third target exhausts almost all of the drone battery. Thus, the launch point of the drone must be very near the first target and the land point must be very near the third target. In the GSGC+S solution, the drone returns to the mothership after the second target to recharge/refuel. After relaunching the drone to visit the third target, the ship begins moving to a more convenient rendezvous location, closer to $orig = dest$. In GSGC+S, the distance traveled by the mothership, which is the slower vehicle, is reduced significantly.

8 MDRP-IC Computational Results

Computational results for algorithms related to MDRP-IC are found in Table 4 and Table 5. In Table 4, instances are generated by randomly selecting $orig$ and the target set over a uniform distribution on grid of size 100 by 100. In Table 5, 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 and Table 5, $R = 20$ and $\alpha = 2$ are fixed. The columns under BAB-C corre-

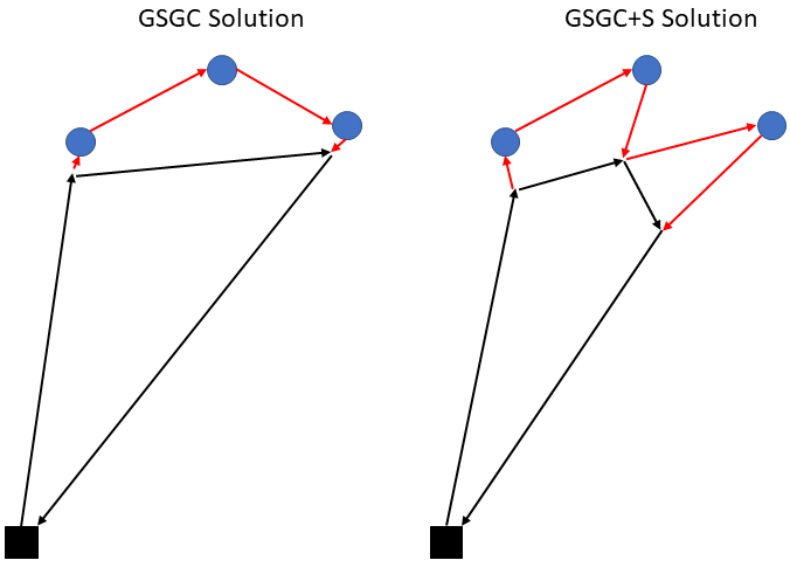

Figure 4: Black line segments show the path of the mothership. Red line segments display the flight path of the drone. The black square is the location $orig = dest$. The blue circles are target locations. Left: the GSGC solution. Right: the GSGC+S solution. We point out that the distance traveled by the mothership is smaller on the right.

spond with the BAB-C solution method; the columns under GSEC correspond with the GSEC solution method; the 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: Computational results for MDRP-IC 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 5: Computational results for MDRP-IC on clustered instances.

8.1 Analysis of MDRP-IC 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-IC. 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 of $slackFactor$ may produce even better results.

9 Variants, Conclusions, and Future Work

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-IC that can be solved by modifying our proposed solution method in minor ways. 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 or launching many disposable UAVs/gliders from a hybrid airship, weight constraints, energy constraints, minimizing the sum of waiting times, and forcing the mothership itself to visit particular targets.

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 Infinite Capacity Drone Problem (MDRP-IC), where a drone may visit multiple targets consecutively without returning to the ship. We proposed both exact and heuristic methods to MDRP-IC. The exact solution method was slow. However, the GSEC solution method, the GSGC heuristic, and the GSGC+S methods provide faster solution methods. GSEC produces 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.

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?

References

[1] Niels Agatz, Paul Bouman, and Marie Schmidt. "Optimization approaches for the traveling salesman problem with drone". In: *Transportation Science* (2018), to appear.

[2] James F. Campbell et al. *Drone Arc Routing Problems*. Tech. rep. University of Missouri, St. Louis, Aug. 2018.

[3] James Campbell, D.C. Sweeney, and Zhang J. *Mobile Systems IV*. Tech. rep. University of Missouri, St. Louis, Apr. 2017.

[4] UPS YouTube Channel. *UPS Tests Residential Delivery Via Drone*. [YouTube Video]. Feb. 2017. URL: https://www.youtube.com/watch?v=%5C%5Cxx9%5C_6OyjJrQ.

[5] Walton Pereira Coutinho et al. "A branch-and-bound algorithm for the close-enough traveling salesman problem". In: *INFORMS Journal on Computing* 28.4 (2016), pp. 752–765.

[6] Iman Dayarian and Martin Savelsbergh. *Same-Day Delivery with Heterogeneous Fleets of Drones and Vehicles*. Tech. rep. Georgia Institute of Technology, Aug. 2018.

[7] Michael Franco. *DHL uses completely autonomous system to deliver consumer goods by drone*. Accessed: March 28, 2018. May 2016. URL: https://newatlas.com/dhl-drone-delivery/43248/.

[8] Quang Minh Ha et al. "Heuristic methods for the traveling salesman problem with drone". arXiv preprint arXiv:1509.08764. 2015.

[9] Andy M. Ham. "Integrated scheduling of m-truck, m-drone, and m-depot constrained by time-window, drop-pickup, and m-visit using constraint programming". In: *Transportation Research Part C: Emerging Technologies* 91 (2018), pp. 1–14.

[10] Gurobi Optimization Inc. *tsp.py*. Python code. Accessed: March 1, 2018. 2017. URL: http://www.gurobi.com/documentation/7.5/examples/tsp_py.html.

[11] Brian Kernighan and Shin Lin. "An efficient heuristic procedure for partitioning graphs". In: *The Bell System Technical Journal* 49.2 (1970), pp. 291–307.

[12] Arjun Kharpal. *Amazon wins patent for a flying warehouse that will deploy drones to deliver parcels in minutes*. [Accessed 1-Aug-2018]. Dec. 2016. URL: https://www.cnbc.com/2016/12/29/amazon-flying-warehouse-deploy-delivery-drones-patent.html.

[13] Miguel Sousa Lobo et al. "Applications of second-order cone programming". In: *Linear Algebra and its Applications* 284.1-3 (1998), pp. 193–228.

[14] Lockheed Martin. *Hybrid Airship*. [Accessed 1-Aug-2018]. Aug. 2018. URL: https://www.lockheedmartin.com/en-us/products/hybrid-airship.html.

[15] Renato D.C. Monteiro and Takashi Tsuchiya. "Polynomial convergence of primal-dual algorithms for the second-order cone program based on the MZ-family of directions". In: *Mathematical Programming* 88.1 (2000), pp. 61–83.

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

[17] Alena Otto et al. "Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey". In: *Networks* (2018), to appear.

[18] Stefan Poikonen, Xingyin Wang, and Bruce Golden. "The vehicle routing problem with drones: Extended models and connections". In: *Networks* 70.1 (2017), pp. 34–43.

[19] Stefan Poikonen, Edward Wasil, and Bruce Golden. "A Branch-and-Bound Approach to the Traveling Salesman Problem with a Drone". In: *INFORMS Journal on Computing* (), to appear.

[20] Linley Sanders. *Google Drone Delivery: The Future of Burritos, Drugs, and Fast Food*. Accessed: March 28, 2018. Oct. 2017. URL: `http://www.newsweek.com/google-drones-future-drugs-burritos-and-disaster-relief-686604`.

[21] Halil Savuran and Murat Karakaya. "Efficient route planning for an unmanned air vehicle deployed on a moving carrier". In: *Soft Computing* 20.7 (2016), pp. 2905–2920.

[22] Marlin W. Ulmer and Barrett W. Thomas. *Same-Day Delivery with Heterogeneous Fleets of Drones and Vehicles*. Tech. rep. Technische Universität Braunschweig, Aug. 2018.

[23] Xingyin Wang, Stefan Poikonen, and Bruce Golden. "The vehicle routing problem with drones: Several worst-case results". In: *Optimization Letters* 11.4 (2017), pp. 679–697.

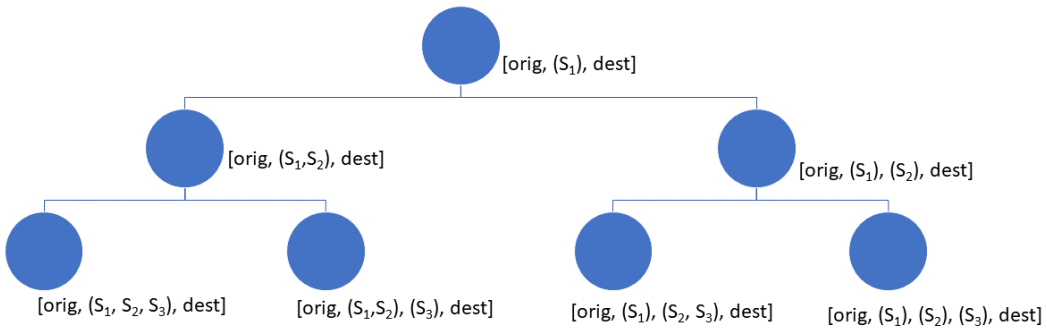

Figure 5: A binary branch-and-bound tree that explores all compositions for the fixed sequence $S = [orig, s_1, s_2, s_3, dest]$. Each left branch appends the new target into the preceding drone subtour. Each right branch appends the new target as a new drone subtour. Next to each node in the figure is the associated composition.

Appendix 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 5.

Appendix 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 2 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 (A1) 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 (A1) 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:`

$$minimize(\sum_{k=0}^{n+1}(cTime(k) + sTime(k)) \quad (F0)$$

`Subject to:`

`For k=0 to n:`

$$\|lPoint(k+1) - rPoint(k)\| \leq cTime(k) \quad (F1)$$

$$\|lPoint(k) - rPoint(k)\| \leq sTime(k) \quad (F2)$$

$$(outFlightDist(k) + inFlightDist(k))/\alpha \leq sTime(k) \quad (F3)$$

$$sTime(k) \leq R \quad (F4)$$

`End For`

$$lPoint(0) = orig \quad (F5)$$

$$rPoint(0) = orig \quad (F6)$$

$$lPoint(n+1) = dest \quad (F7)$$

$$rPoint(n+1) = dest \quad (F8)$$

`For k=1 to n:`

$$\|readPoint_k - lPoint(k)\| \leq outFlightDist(k) \quad (F9)$$

$$\|readPoint_k - rPoint(k)\| \leq inFlightDist(k) \quad (F10)$$

$$\|s_k - readPoint_k\| \leq rad_i \quad (F11)$$

`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.

The *Close-Enough* Variant: Use with a Hybrid Airship

Another interesting application of the close-enough variant follows. In 2016, Amazon won a patent related to the use of an airship that is capable of launching drones to fulfill parcel requests [12]. Lockheed Martin [14] characterizes their Hybrid Airship, which carries up to 21 metric tons of cargo, as extremely fuel efficient relative to alternative means of transporting goods to remote regions. Suppose there exists a large sea ship that takes the role of the mothership. Also suppose that a hybrid airship (or a cargo airplane) takes the role of the drone, which only must come close-enough to each target. The hybrid airship may then launch a glider or smaller UAV once it is close-enough to a given target. The glider or UAV is not required to return to the airship. Because the hybrid airship operates at altitude, it effectively extends the range of the gliders/small UAVs. Moreover, it may facilitate access to inland areas that would not be reachable by launching a small UAV directly from a sea ship.

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 time is a fixed constant $minWait$, we can model this by adding the following constraints to `LENSEQ`.

$$\texttt{For k=1 to n-1:}$$
$$minWait \leq cTime(k)$$

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.

$$\texttt{For k=1 to n-1:}$$
$$\delta * sTime(k) \leq cTime(k)$$

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`.

$$\texttt{For k=1 to n:}$$
$$e(w_k) * outFlightDist(k) + e(0) * inFlightDist(k) \leq E$$

Forcing Mothership to Visit a Target

Suppose we wish to force the mothership to visit some specified target $t_i \in T$. Also suppose that $S = [orig, s_1, s_2, ..., s_k, ..., s_n, dest]$ is any arbitrary sequence of target visitations and that $t_i = s_k$. We may force the ship to visit t_i by adding the following constraints to the second order cone program `LENSEQ(`S`)`:

$$lPoint(k) = t_i$$
$$rPoint(k) = t_i.$$

Appendix C: Variants of MDRP-IC

We may incorporate additional constraints or features into the MDRP-IC 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-IC.

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-IC 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:

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

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`.

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

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 (E6) of `LENCOMP` from:

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

to:

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