Project Number: ACT DL01

Pre-Mission Flight Plan Optimization

A Major Qualifying Project Report

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Alexander Sunde-Brown

with

Ethan M. Moon (Independent Study)

Date: March 27, 2015

Approved:

Professor Andrew C. Trapp, Advisor

Professor Suzanne L. Weekes, Advisor

Abstract

The goal of this project is develop a tool to help with mission planning undertaken at the National Aeronautic and Space Administration in partnership with Draper Laboratory. The aim of these missions is to gather data using Earth-observing aircraft at a set of sites. In this project, we aim to minimize the total mission time. We model this minimization problem as a variant of the Distance Constrained Vehicle Routing Problem. We present two integer programming formulations of the problem as well as a supporting proof. While the results we present show that the solution time is limited by the available flight time for each aircraft and the number of vehicles, our implementation of the integer programming problem is able to solve problems with up to 35 sites.

Contents

1 Introduction

In the late 1950's, the United States Air Force began working to develop unmanned aircraft, primarily for use in high-risk reconnaissance and attack missions. In recent years, the use of unmanned aerial vehicles (UAVs) has increased significantly in both military and civilian applications as a result of rapidly developing technologies and falling costs of production. UAVs can *i*) be autonomously controlled, meaning the aircraft is completely computer driven, *ii*) be operated by a human who controls it remotely, or *iii*) utilize a mixed-control scheme similar to a commercial aircraft. The ability to be continuously airborne for up to thirty-six hours makes UAVs particularly suitable for surveillance and data collection missions. This unique capability is of particular interest to The Charles Stark Draper Laboratory (hereafter referred to as Draper Lab) for use in studies to observe Earth phenomena, such as wildfires or landslides.

Draper Lab was divested from the Massachusetts Institute of Technology in 1973 to become a non-profit research and development laboratory focused on design development and deployment of advanced technological solutions. Draper Lab specializes in the areas of navigation, guidance, and control systems for application to problems of interest for a variety of governmental and private organizations. They are presently working on a National Aeronautics and Space Administration (NASA) funded project to help develop technologies that improve the total scientific value gained from observing a variety of Earth phenomena, e.g., landslides, volcanic eruptions, and hurricanes [1]. Ideally, Earth observing assets would be able to gather data before, during, and after such events. Using a combination of space-based and aircraft-based sensor observations provides advantages over the use of either by themselves for the development of improved science models.

Our project focuses on the use of UAVs as a method of observing Earth phenomena, specifically focusing on landslides. In their current and previous NASA funded projects, Draper Lab has worked with researchers who utilized unmanned aerial vehicles currently owned by NASA, namely the Ikhana-MQ9 and two of the earliest Global Hawks, AV-1 and AV-6. We will use these aircraft as representative UAVs in our study. The physical characteristics are given in Appendix A; the most important to our application are the velocities and the range of each of the two classes of aircraft. The effective use of aircraft can offer much higher resolution for a range of sensor types, as well as a large reduction in issues related to cloud coverage.

The focus of this project is the development of tools to aid in the planning of data collection missions that require observation of a large number of sites. To that end, we obtain the flight plan for a potential mission by solving a combinatorial optimization problem. Specifically, we utilize a variation on the vehicle routing problem to route each UAV through a different set of nodes in a complete graph. The vehicle routing problem will find the minimum total distance, or minimum total time, for the mission.

2 Problem Overview

Our goal for this project was to develop a pre-mission flight planning tool to help Draper Lab collect data on Earth phenomena such as landslides. The tool can be used to determine the routes taken by a fleet of UAVs fitted with data collection equipment for an arbitrary set of locations. The tool seeks to minimize the collective distance traveled by the UAVs over the course of the mission. With that as our objective, we model this problem using combinatorial optimization. Important constraints include that each of the UAVs must depart and return to one depot, there is a waiting time associated with each site, and both types of UAV have a limited total flight time.

We expect to model flight missions that span more than a single day, likely a week or more. Thus, the task at hand is to design a flight plan that visits β sites with $\tilde{k}$ UAVs over r days with the constraint that each vehicle can only fly for M hours each day. To do this, we model the problem as a Distance Constrained Vehicle Routing Problem (detailed in Section 3) to solve a single day mission to β sites with $k = r\tilde{k}$ vehicles. It will be necessary to choose a suitable r that will produce an effective flight plan.

3 Background and Literature Review

We provide background information to motivate our ensuing discussions on mathematical formulations and algorithmic design. This includes a briefing on mathematical programming, combinatorial optimization, as well as vehicle routing problems.

3.1 Mathematical Programming

Mathematical programming is an approach to solving optimization problems that developed primarily during the mid 20th century. It consists of defining variables to different elements of a given problem. These elements can be physical quantities, such as the amount of a chemical used in making a compound, or more abstract ideas, like whether or not a worker is used on a building project. Once defined, these variables are used in constructing two sets of equations, an objective function and a set of constraints. The objective function represents the goal of the problem, such as minimizing the cost of a set of ingredients or maximizing the amount of work finished on a building project. The constraints represent innate restrictions on the behavior of the variables, and provide what is known as a feasible region for the problem. Values of the variables within this feasible region are allowable solutions to the problem; any values that are outside are invalid, having violated one or more constraints. The goal of a mathematical program is to determine the optimal values of the variables, that is, the values that optimize the objective function while satisfying all constraints.

3.2 Combinatorial Optimization

Combinatorial optimization encompasses a class of optimization problems that are concerned with the idea of finding an optimal solution from a finite set of objects. One of the most well-known problems within combinatorial optimization is the Traveling Salesman Problem (TSP) which was discussed during the 18th century by Sir William Hamilton [2]. The TSP attempts to find the shortest possible route for a salesman to take through a set of houses or locations such that the salesman visits each location once and returns to where he began. The TSP has spawned a number of offshoots and subproblems for a variety of applications. These subproblems, and the TSP itself, are in general NP-Hard problems, and are thus computationally difficult [3].

A popular method of solving combinatorial optimization problems is through the application of linear programming (LP) as well as integer programming (IP) formulations. An overview of common solution methods for LP's and IP's is detailed in Appendix B.

3.3 Vehicle Routing Problems

The vehicle routing problem (VRP) was first formulated by George Dantzig and John Ramser in 1959, in a paper titled *The Truck Dispatching Problem* [4]. They describe the VRP as a variation of the Traveling Salesman Problem, with additional conditions to accommodate multiple salesman or vehicles. Clarke and Wright developed a heuristic that made the VRP much easier to solve [5]. Active research on the VRP continues in domains such as transportation and distribution problems. The VRP supposes that a single site exists, known as the depot, that is used as a source for multiple vehicles to make deliveries to a set of sites, known as the customers. The goal of the VRP is to minimize the total distance of the routes taken by the vehicles. Alternatively, this distance can be represented as the travel time, fuel consumption, or wages paid to the drivers.

There exist several different variations to the VRP [6]; a few of the more common varieties are described below.

- Capacitated Vehicle Routing Problem (CVRP): In this variation, each site is assigned a demand for some arbitrary object or service. A vehicle capacity, representing the amount of demand each vehicle can satisfy on its route, is also assigned.

- Vehicle Routing Problem with Time Windows (VRPTW): In these problems, each site has a window of time in which deliveries can be made. The distance from site to site is in terms of the time required to take that particular route. Typically, a secondary algorithm steps through each route and compares when the vehicle arrives and departs each site. Often, these problems involve a "waiting time" at each node, which represents the amount of time it takes to complete the delivery at that site. Many VRPTW are also CVRPs, as they include vehicle capacity and demand at each node, adding to the complexity of solving the problem.

- Vehicle Routing Problem with Backhauls (VRPB): In this variation, customers are divided into two categories: customers who require the product to be delivered, called

linehauls, and those who require it to be picked up, called backhauls. Tours that include both linehauls and backhauls must service the linehauls first.

- Vehicle Routing Problem with Pickup and Delivery (VRPPD): This variant has two demands associated with each node: one for pickup, p and one for deliveries, d. In addition, for each customer, the origin and destination of the product to be picked up and subsequently delivered must be specified.

- Distance Constrained Vehicle Routing Problem (DVRP): This variant introduces a new restriction in the form of a maximum tour distance where no tour in the solution of the DVRP can exceed this distance. DVRPs are often preprocessed to determine the minimum number of vehicles required to solve the problem.

In Section 3.3.3, we give the standard formulation for a CVRP from which many more complicated varieties of the VRP are derived.

3.3.1 VRP Properties

To formulate a VRP, we represent the set of sites and the connections between them as a graph, with sites being nodes and the connections as edges. In many cases, the connections are not given explicitly, and it is assumed that the graph is fully connected, such that every node has an edge connecting it to every other node in the graph. A solution to the vehicle routing problem will be a set of edges that are "active" in the solution. If an edge is "active" in the solution, it is being traversed by one of the vehicles.

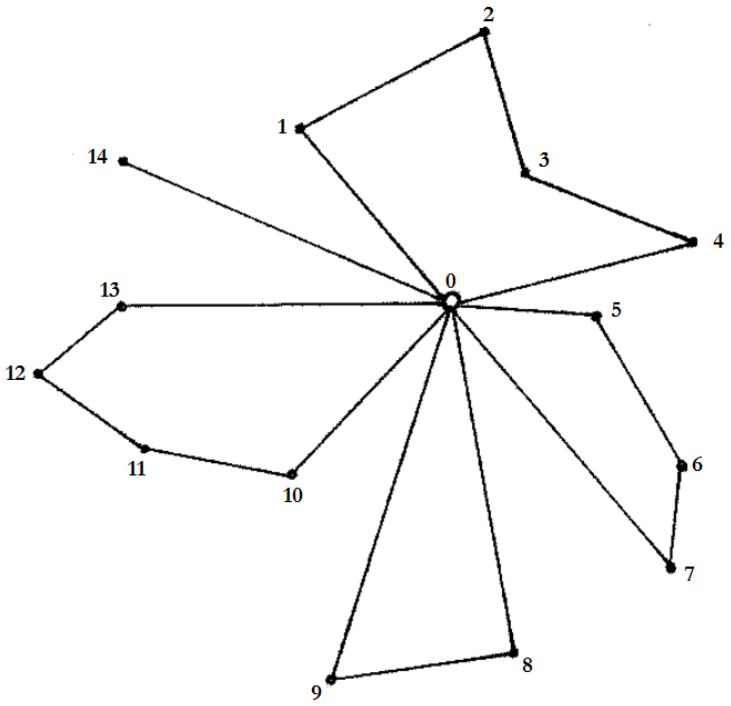
Each of these edges is assigned a weight, which represents some cost involved with using that edge in the solution. In our application, we will be dealing with symmetric graphs, where the weight on an edge connecting node A to node B is equal to the weight on the edge connecting node B to node A. The objective of the VRP is to minimize the total weight of all the edges involved in the solution, subject to the restrictions of the Vehicle Routing Problem: all vehicles must begin and end their routes at the depot, and each site must be visited exactly once. Figure 1 represents an example solution to a VRP with 5 vehicles over 16 sites. Edge weights have been omitted for visual clarity. Notice that the depot, site P, is connected to site O by only one edge, which represents a single vehicle going out to that node and back to the depot in a single trip.

In Section 3.3.3, we discuss a formulation of the CVRP. The only difference in the CVRP from the base VRP formulation is the addition of a set of capacity constraints, which collectively permit only those tours for which total demand at included nodes is within the specified capacity threshold.

3.3.2 Notation

Here we list the notation that is used for the standard integer programming formulation of a symmetric CVRP. We distinguish between two types of cycles in a graph: "**subtours**" which are cycles that do not include the depot node, and "**depot tours**" which are cycles that include the depot.

Figure 1: Example solution to a VRP with 5 vehicles



- G is the set of all nodes $\{0, 1 \dots \beta\}$, with β being the number of nodes in the graph. Node 0 represents the supply depot.
- $N = G \setminus \{0\}$ is the set of customer nodes. These are referred to as non-depot nodes.
- $c_{i,j}$ represents the distance between node i and node j for $i < j$.
- $x_{i,j}$ for $i < j$ is a variable indicating whether the edge between node i and node j is traversed in the graph:

$$x_{i,j} = \begin{cases} 0, & \text{if the edge between } i \text{ and } j \text{ is not traversed.} \\ 1, & \text{if the edge between } i \text{ and } j \text{ is traversed once.} \\ 2, & \text{if the edge between } i \text{ and } j \text{ is traversed twice (i.e a two-cycle).} \end{cases}$$

- d_i is the demand at node i.
- k is the number of identical vehicles.
- L is the capacity of any of the k identical vehicles.
- $b(S)$ is the minimum number of trucks needed to fulfill the demand of subtour S. It is defined as $b(S) = \lceil (\sum_{i \in S} d_i)/L \rceil$.

3.3.3 Standard Formulation of the CVRP

Below is the standard formulation of the CVRP (see, e.g., [7]).

$$\text{minimize} \quad \sum_{i,j \in G} c_{i,j} x_{i,j} \tag{1}$$

$$\text{subject to} \quad \sum_{i \in N} x_{i,j} + \sum_{i \in N} x_{j,i} = 2 \qquad \forall j \in N \tag{2}$$

$$\sum_{j \in N} x_{0,j} = 2k \tag{3}$$

$$\sum_{i \in S, j \notin S} x_{i,j} \geq 2b(S) \quad \forall S \subset N, \ \ |S| > 1 \tag{4}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall i, j \in N \tag{5}$$

$$x_{0,j} \in \{0, 1, 2\} \qquad \forall j \in N. \tag{6}$$

We note that $x_{i,j}$ is only defined for $i < j$. Equation (1) states that the objective of the problem is to minimize the total distance traveled by all of the vehicles. Equations (2) and (3) can be interpreted as constraints on the degree of the nodes. We can enforce them by saying that the number of edges connected to the depot is equal to twice the number of vehicles, and that the remaining sites each have exactly two edges connected to them. Equations (2) and (3) taken together guarantee that any solution is made up of subtours and k depot tours. Equation (4) ensures that all subsets of nodes are serviced by enough vehicles. This is accomplished by requiring that there are enough edges from inside a subset of the graph to outside of it for each vehicle required to meet the demand of the subset. Equations (5) and (6) define the integer requirements for the variables: the second giving all edges connected to the depot the ability to be traversed either 0, 1 or 2 times, while the first states all other edges must be traversed either 0 or 1 times.

4 Solution Methodology

In this section, we discuss our methodology in solving the problem that was briefly presented in Section 2. As stated there, we model the problem as a modified Distance Constrained Vehicle Routing Problem (DVRP).

To the DVRP that was described in Section 3.3, we add a waiting time at each node, remove the demand and capacity of the vehicles, and let the weights on the edges between nodes be given in time. In this section, we present two formulations of this modified DVRP. The approaches differ in the mathematical expression of the method of elimination of subtours.

4.1 Assumptions

For the sake of clarity and simplicity, we make the following assumptions:

- Symmetry is assumed with regard to travel time between sites. This means the time to travel from site A to site B is equal to the time to travel from site B to site A.
- All vehicles are assumed to be identical in that they move at the same speed, and have the same sensing and collecting capabilities. This means that any vehicle is able to gather all necessary data from any site.

4.2 Formulating a DVRP

While standard formulations for DVRPs exist, many include a process for determining the number of vehicles, which is not applicable in the approach we took, as the number of vehicles is fixed (see Section 2.2 and 4 for additional discussion). In addition, most DVRPs are formatted as a subset of CVRPs, with each node having a demand associated with it rather than a waiting time, as is the case with our problem. Thus, we developed our own, adapted formulation to represent our problem. The following is an overview of our notation and the steps we took in creating our formulations.

4.2.1 Notation

- G is the set of all nodes $\{0, 1 \dots \beta\}$. Node 0 represents the supply depot.
- $N = G \setminus \{0\}$ is the set of customer nodes. These are referred to as non-depot nodes.
- c_{ij} represents the travel time between node i and node j.
- $x_{i,j}$ for $i < j$ is a variable indicating whether the edge between node i and node j is traversed in the graph:

$$x_{i,j} = \begin{cases} 0, & \text{if the edge between } i \text{ and } j \text{ is not traversed.} \\ 1, & \text{if the edge between } i \text{ and } j \text{ is traversed once.} \\ 2, & \text{if the edge between } i \text{ and } j \text{ is traversed twice (i.e a two-cycle).} \end{cases}$$

- A tour S is an ordered subset of nodes, $\{a_1, a_2, a_3, \dots a_d\}$ such that x_{a_1,a_2}, x_{a_2,a_3}, $\dots$, x_{a_{d-1},a_d}, $x_{a_d,a_1} \geq 1$. We may also use the notation a_{d+1} which should be taken to mean a_1, so that $x_{a_d,a_{d+1}} = x_{a_1,a_d}$. Also, $x_{a_i,a_{i+1}}$ will be defined as x_{a_{i+1},a_i} when $a_i > a_{i+1}$.
- A depot tour D is a tour where $a_1 = 0$.
- w_i represents the wait time required to collect data at node i.
- $\tilde{k}$ is the number of identical vehicles.

- r is the number of days over which the mission is conducted.
- $k = r\tilde{k}$ is the number of artificial vehicles, used to represent the number of days r.
- M is the maximum trip time for each vehicle.

4.2.2 Formulation

The following traits are fundamental to the DVRP:

1. Each of the k vehicles arrives and departs from the depot.
2. Every other site has a vehicle arrive and depart from it.
3. No vehicle can travel a route with a trip time that exceeds M, $M > 0$.

As stated above in Section 4.2.1, we have two types of tours we need to eliminate: subtours, which don't include the depot, and depot tours with trip time longer than M. We disallow individual subtours and depot tours by enforcing that the set of edges that makes up the tour not appear again in any solution. This is accomplished by utilizing the following set of constraints:

$$\sum_{i=1}^{|P|} x_{a_i,a_{i+1}} \leq |P| - 1 \quad \forall \text{ tours } P \; s.t. \; |P| \geq 2. \tag{7}$$

This removes subtours within the graph by requiring at least one edge of the subtour to be 0. Eliminating depot tours with total trip time that exceeds the maximum trip time requires a more complex constraint set. We will now discuss our approach to eliminating depot tours that exceed the maximum trip time. Below is a proof to motivate our constraint to eliminate only depot tours that exceed the maximum trip time; before its presentation we must make two definitions.

Definition: The *time* of a tour $S = \{a_1, a_2, \ldots, a_d\}, \; d \geq 2$ is defined as:

$$T(S) = c_{a_1,a_d} + \sum_{i=1}^{d-1} c_{a_i,a_{i+1}} + \sum_{j=1}^{d} w_{a_j}.$$

Definition: Given a fixed tour $D = \{a_1, a_2 \ldots a_d\}$ and $\lambda \in \mathbb{R}$ we define the value of the tour S to be $V_D(S)$

$$V_D(S) = \lambda x_{0,a_2} + \lambda x_{0,a_d} + \sum_{i=2}^{d-1} x_{a_i,a_{i+1}}$$

for $\lambda \in (0, \frac{1}{2})$ and where the $x_{i,j}$ are those from tour S.

Theorem 4.1 *Given a depot tour $D = \{a_1, a_2, \ldots, a_d\}$ s.t. $T(D) > M$, and for any $\lambda \in (0, \frac{1}{2})$, the following constraint applied in conjunction with constraint* (3) *and* (2) *disallows only the tour D:*

$$\lambda x_{0,a_2} + \lambda x_{0,a_d} + \sum_{i=2}^{d-1} x_{a_i,a_{i+1}} < 2\lambda + (d-2). \tag{8}$$

Proof. Let $\beta \in \mathbb{N}$, $\beta \geq 1$, $G = \{0, \ldots, \beta\}$ be given. Let $D = \{0, a_2, \ldots, a_d\}$ be a depot tour with $T(D) > M$.

Thus, by definition of D, $V_D(D) = \lambda + \lambda + d - 2 = 2\lambda + (d-2)$. We want to show that for any tour $S = \{a_1, a_2 \ldots, a_d\}$, $S \neq D$, $V_D(S) < 2\lambda + (d-2)$. As x_{0,a_2} and x_{0,a_d} can take the values 0,1, and 2, and $x_{a_i,a_{i+1}}$ can be 0 or 1 for $i = 2, \ldots d-1$, we consider all tours expressed through the following cases:

i. $x_{0,a_2} = 0, x_{0,a_d} = 0$

ii. $x_{0,a_2} = 1, x_{0,a_d} = 0$ or $x_{0,a_2} = 0, x_{0,a_d} = 1$

iii. $x_{0,a_2} = 1, x_{0,a_d} = 1$

iv. $x_{0,a_2} = 0, x_{0,a_d} = 2$ or $x_{0,a_2} = 2, x_{0,a_d} = 0$

v. $x_{0,a_2} = 1, x_{0,a_d} = 2$ or $x_{0,a_2} = 2, x_{0,a_d} = 1$

vi. $x_{0,a_2} = 2, x_{0,a_d} = 2$

Case (i)

If $x_{0,a_2} = x_{0,a_d} = 0$, then $V_D(S) = \sum_{i=2}^{d-1} x_{a_i,a_{i+1}} \leq d - 2 < 2\lambda + d - 2$, thus tour S is allowed under (8)

Case (ii)

If $x_{0,a_2} = 0$ and $x_{0,a_d} = 1$, then $V_D(S) = \lambda + 0 + \sum_{i=2}^{d-1} x_{a_i,a_{i+1}} \leq \lambda + d - 2 < 2\lambda + d - 2$, so S is allowed under (8).

Case (iii)

If $x_{0,a_2} = x_{0,a_d} = 1$, then $x_{a_i,a_{i+1}} = 1$ for $i = 2 \ldots d-1$ and S is tour D. Thus, for S to be different from D, $\exists\ j$ s.t $x_{a_j,a_{j+1}} = 0$, thus $V_D(S) = \lambda + \lambda + \sum_{i=2}^{d-1} x_{a_i,a_{i+1}} < 2\lambda + (d-2) - 1$ so S is allowed under (8).

Case (iv)

If $x_{0,a_2} = 2$ and $x_{0,a_d} = 0$ then by (2), $x_{a_2,a_3} = 0$. Thus $V_D(S) = 2\lambda + 0 + \sum_{i=2}^{d-1} x_{a_i,a_{i+1}} \leq 2\lambda + d - 3 < 2\lambda + d - 2$, so S is allowed under (8).

Case (v)

If $x_{0,a_2} = 2$ and $x_{0,a_d} = 1$, then by (2), $x_{a_2,a_3} = 0$. Then either:

- $d = 3$, $V_D(S) = 3\lambda + x_{a_2,a_3} < 2\lambda + 1$, so S is allowed under (8).
- $d > 3$, $V_D(S) \leq 3\lambda + d - 3 < 2\lambda + d - \frac{5}{2} < 2\lambda + d - 2$, so S is allowed under (8).

Case (vi)
If $x_{0,a_2} = x_{0,a_d} = 2$, then either:

- $d = 3$, $x_{a_2,a_3} = 0$ and $V_D(S) = 2\lambda + 2\lambda < 2\lambda + \frac{1}{2} + \frac{1}{2} = 2\lambda + 1$, so S is allowed under (8).
- $d > 3$, $x_{a_2,a_3} = x_{a_{d-1},a_d} = 0$ and $V_D(S) = 4\lambda + \sum_{i=2}^{d-1} x_{a_i,a_{i+1}} \leq 4\lambda + d - 4 < 2\lambda + d - 2$, so S is allowed under (8).

Thus, all tours S, $S \neq D$ are allowed under (8). ■

Using constraint (12) and (13) results in our first formulation of our DVRP variant.

Formulation 1

$$\text{minimize} \quad \sum_{i,j \in G} c_{i,j} x_{i,j} \tag{9}$$

$$\text{subject to} \quad \sum_{i \in N} x_{i,j} + \sum_{i \in N} x_{j,i} = 2 \quad \forall j \in N, \tag{10}$$

$$\sum_{j \in N} x_{0,j} = 2k, \tag{11}$$

$$\sum_{i=1}^{d} x_{a_i,a_{i+1}} \leq |P| - 1 \quad \forall \text{ tours } P \; s.t. \; |P| \geq 2 \, , \tag{12}$$

$$\sum_{i=2}^{d-1} x_{a_i,a_{i+1}} + \lambda x_{a_1,a_2} + \lambda x_{a_d,a_{d+1}} \leq |D| + \lambda - 2 \tag{13}$$

$$\forall \text{ depot tours } D \; s.t. \sum_{j=1}^{d} c_{a_j,a_{j+1}} + \sum_{l=1}^{d} w_{a_l} > M, |D| \geq 3,$$

$$x_{i,j} \in \{0,1\} \qquad \forall i,j \in N, \tag{14}$$

$$x_{0,j} \in \{0,1,2\} \qquad \forall j \in N. \tag{15}$$

Formulation 1 is a valid formulation of our DVRP with exponentially many constraints of the form (12)-(13). This formulation gives the optimal solution when such a solution exists. However, we have found that it does so in an unreasonable amount of time for instances with very low mission time M.

Implementing all constraints at once is computationally infeasible; this is discussed more in Section 4.3. Thus, we consider an elimination constraint different from (12) that enforces connectivity of subtours to the remainder of the graph. Specifically, for each subtour it requires at least two edges that connect sites within the subtour to sites within the remainder of the graph to be active [8]. We call this Formulation 2.

Formulation 2

$$\text{minimize} \quad \sum_{i,j \in G} c_{i,j} x_{i,j} \tag{16}$$

$$\text{subject to} \quad \sum_{i \in N} x_{i,j} + \sum_{i \in N} x_{j,i} = 2 \quad \forall j \in N, \tag{17}$$

$$\sum_{j \in N} x_{0,j} = 2k \tag{18}$$

$$\sum_{i \in S, j \in G \setminus S} x_{i,j} + \sum_{i \in G \setminus S, j \in S} x_{i,j} \geq 2 \quad \forall \text{ tours } S \text{ } s.t. \text{ } |S| \geq 1 \tag{19}$$

$$\sum_{i=2}^{d-1} x_{a_i,a_{i+1}} + \lambda x_{a_1,a_2} + \lambda x_{a_d,a_{d+1}} \leq |Q| + \lambda - 2 \tag{20}$$

$$\forall \text{ depot tours } Q \text{ } s.t. \sum_{j=1}^{d} c_{a_j,a_{j+1}} + \sum_{l=1}^{d} w_{a_l} > M, |Q| \geq 3$$

$$x_{i,j} \in \{0,1\} \qquad \forall i,j \in N \tag{21}$$

$$x_{0,j} \in \{0,1,2\} \qquad \forall j \in N \tag{22}$$

4.3 Implementation of Subtour Constraints

The implementation of the constraints (12), (13) and (19), (20) in their respective formulations poses a significant challenge. To construct these constraints at the outset of the problem requires calculating every subset of the β sites observed, which results in finding $\sum_{k=1}^{\beta} \binom{\beta}{k}$ independent subsets. This can lead to an extremely large number of subsets. For example, with 30 sites to visit, we would have to find and create constraints for $1,073,741,823$ subsets. Even if finding a subset and creating a constraint took a millisecond (10^{-3} seconds), it would take 12.42756 days to create all of the subsets.

We instead start by solving (9) subject to (10), (11), (14), and (15) only. Then, we check if (12), (13) are violated anywhere in our solution and, if so, we re-solve the prior problem but with additional constraints that disallow the unwanted tours that were found. This procedure of solving optimization problems with more and more constraints is continued until the solution does not contain tours that violate (12) and (13).

During implementation, we found that subtour elimination constraint (12) was computationally taxing. To eliminate this we moved from a subtour elimination method using the constraint set (12) to an elimination method using the constraint set (19). This significantly reduced the time and number of iterations required to find an optimal solution. The reason this change reduced our total number of iterations is (12) creates a constraint specific to each violated subtour. The set (19) creates a constraint that forces connectivity of the nodes involved in the subtour to the rest of the graph. For comparison, implementations of Formulation 1 and Formulation 2 were run with identical parameters. Formulation 1 hit our iteration cap of 6,000 iterations, whereas Formulation 2 found an optimal solution in only 3 iterations.

4.4 Finding the Minimum Number of Mission Days

While the formulations presented in Section 4.2.2 find the optimal route for a single mission day (represented by the max mission time M), missions can take place over more than one day. For example, perhaps it is not possible to visit all of the sites with the available vehicles in a single day. To allow for this, we will employ a solution method that presents the mission planner with not only the solution corresponding to the minimum number of days, but also alternative solutions with greater numbers of mission days. This allows for the mission planner to make a decision that includes other factors not included in our model, such as vehicle scheduling issues, crew holidays, site availability, and other nonstandard restrictions.

To find the minimum number of days, we iteratively solve the DVRP formulation presented in Section 4.2.2 with a decreasing number of mission days until the problem becomes infeasible. To solve a DVRP with β sites, max tour time M, and $\tilde{k}$ vehicles over r days, we solve the same DVRP with $k = r\tilde{k}$ vehicles, keeping all other parameters the same. We begin stepping down our iterations from the maximum number of days, supposing that each vehicle visits only a single site each day, as $\left\{ \left\lfloor \frac{\beta}{k} \right\rfloor, \left\lfloor \frac{\beta}{k} \right\rfloor - 1, \ldots, 1 \right\}$. Each iteration reduces the number of days by one and resolves the problem, repeating until the problem is infeasible. Each solution is stored for the mission planner to consider.

5 Computational Discussion

In this section, we outline the method used to solve the equations in Formulation 2 (16)–(22) and related computational discussions. In addition, we discuss our computing environment and testing methods.

5.1 Algorithmic Outline

This section covers an algorithmic outline of the procedure we use to solve the DVRP, specifically focusing on the methods used to eliminate subtours and ensure that the tours in the solution are all within the maximum trip time. As stated in Section 3.3.2, the depot is site 0, and subsequent sites are increasing numbers up to β, the number of sites. As stated

previously, this problem is symmetric, so edges $x_{i,j}$ and $x_{j,i}$ are equivalent, and we do not list them both in x, we only list $x_{i,j}$ for $i < j$. As a result, the x vectors always have the same size, $\frac{1}{2}(\beta-1)^2 + \frac{1}{2}(\beta-1)$. Thus, the number of sites β can be determined from x using the quadratic formula, which yields that $\beta = \frac{1}{2} + \sqrt{\frac{1}{4} + 2x}$. Note that we only include the positive root for β since the negative root will always produce a negative β. A negative number of sites doesn't make sense in the context of our problem.

To illustrate the format of our representation of a solution to the DVRP, we present an example that has not had subtours or depot tours removed. In the example below, the tour calculated using only (17) and (18) consists of two triangles. One consists of the sites 0,1,2, and the other consists of the sites 3,4,5. In other words, we have a valid depot tour passing through the depot 0 and sites 1 and 2 and an invalid subtour through sites 3, 4 and 5.

$$x = \begin{bmatrix} x_{0,1} \\ x_{0,2} \\ x_{0,3} \\ x_{0,4} \\ x_{0,5} \\ x_{1,2} \\ x_{1,3} \\ x_{1,4} \\ x_{1,5} \\ x_{2,3} \\ x_{2,4} \\ x_{2,5} \\ x_{3,4} \\ x_{3,5} \\ x_{4,5} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{23}$$

A number of subroutines are required to handle smaller portions of our algorithm to solve Formulation 2. We chose to describe the accessory functions in words, reserving pseudo code for the two major algorithmic components: the algorithms to identify subtours and depotours that violate (20) and the algorithm to solve the problem as a whole.

Subroutine Overview:

- **varof**(i, x) returns the variable that corresponds to index i in vector x. This assumes that the vector x is of the form detailed above. For example, using (23), **varof**$(6, x) = \{x_{1,2}\}$.
- **lnode**(i, x) and **rnode**(i, x) Returns, respectively, the number of the left and right nodes of the variable at index i in vector x. For example, using (23), **lnode**$(6, x) = 1$ and **rnode**$(6, x_e) = 2$.

- **find**(n, x) returns the index of the lexicographically first nonzero entry in x that corresponds to an edge connected to node n. For example, using (23), **find**$(3, x) = 13$.

- **graph**(S) takes a list of latitudes and longitudes as inputs and calculates all pairwise distances. It then puts the distances into a matrix such that the distances from site to site match up with the edge that corresponds to that site connection.

- **time**(d, w) takes a set of edges d, like the one outputted by the algorithm **Tours** (defined below), and a waiting time vector w as input. Outputs the sum of the weights on those edges, drawn from the weight vector calculated by **graph**, and the sum of the waiting time of the nodes connected to those edges.

- **Tours**(x): Takes in as input an x vector and separates it into individual vectors consisting of the edges that make up the tours. Tours including the depot are contained in a vector that we call D, and the rest are contained in a vector we call T. For example, if we ran **Tours** on (23), we would get the following sets:

$$D = \{\{x_{0,1}, x_{1,2}, x_{0,2}\}\} \quad , \quad T = \{\{x_{3,4}, x_{4,5}, x_{3,5}\}\}$$

  As a second example, if we run **Tours** on Figure 1, we get the following sets:

$$D = \left\{ \begin{array}{c} \{x_{0,1}, x_{1,2}, x_{2,3}, x_{3,4}, x_{0,4}, \}, \\ \{x_{0,5}, x_{5,6}, x_{6,7}, x_{0,7}\}, \\ \{x_{0,8}, x_{8,9}, x_{0,9}\}, \\ \{x_{0,10}, x_{10,11}, x_{11,12}, x_{12,13}, x_{0,13}\}, \\ \{x_{0,14}\} \end{array} \right\} \tag{24}$$

$$T = \emptyset$$

We now present the pseudocode for the algorithm Tours.

Algorithm 1 Tour Separator

```
1: procedure TOURS(x)
2:     D, T, ← ∅
3:     c, t ← 1                                  ▷ initialize counters for sets D and T vectors
4:     l, r ← 0                                  ▷ initialize left and right node variables
5:     β ← 1/2 + sqrt(1/4 + 2x)                  ▷ solve for the number of nodes N from x
6:     for i = 1 ... β − 1 do
7:         a, V ← ∅                              ▷ V is a set of already visited nodes, a is temp storage
8:         s ← x(i)                              ▷ x(i) is the value of x at index i
9:         x(i) ← 0
10:        a ← varof(i, x)
11:        if s = 2 then                         ▷ check for a two-cycle from the depot
12:            D(c) ← a
13:            c ← c + 1
14:        if s = 1 then                         ▷ begin stepping through the tour
15:            V ← {0}
16:            D(c) ← a
17:            q ← i                             ▷ q is a temporary index variable
18:            l ← lnode(q, x)
19:            r ← rnode(q, x)
20:            repeat
21:                x(q) ← 0
22:                if l ∉ V then
23:                    q ← find(l, x)
24:                    V ← V ∪ {l}
25:                if r ∉ V then
26:                    q ← find(r, x)
27:                    V ← V ∪ {r}
28:                a ← varof(q, x)
29:                D(c) ← D(c) ∪ {a}
30:                l ← lnode(q, x)
31:                r ← rnode(q, x)
32:            until l = 0 or r = 0
33:            c ← c + 1
```

```
34:  repeat
35:      a ← ∅                                   ▷ clear the value of temp storage
36:      j, k, q ← 0                             ▷ clear/initialize index counters
37:      repeat
38:          k ← k + 1
39:          if x(k) = 1 then
40:              j ← k
41:      until j ≠ 0                             ▷ find the first edge that is on in x
42:      s ← x(j)
43:      x(j) ← 0
44:      a ← varof(j)
45:      T(t) ← a
46:      l ← lnode(j, x)
47:      r ← rnode(j, x)
48:      v ← v ∪ l
49:      repeat                                  ▷ begin stepping through the tour
50:          if l ∉ v then
51:              q ← find(l, x)
52:              v ← v ∪ {l}
53:          if r ∉ v then
54:              q ← find(r, x)
55:              v ← v ∪ {r}
56:          l ← lnode(j, x)
57:          r ← rnode(j, x)
58:          a ← varof(q, x)
59:          x(q) ← 0
60:          T(t) ← T(t) ∪ a
61:      until l, r ∈ v
62:      t ← t + 1
63:  until x = 0                                 ▷ 0 is a vector of all zeros
```

The algorithm UAV DVRP Solver is described below. It takes the following inputs:

- S: A list of latitudes and longitudes corresponding to the sites to be visited.
- k: The number of vehicles used in traversing the sites.
- C_0: Set of initial inequality constraints. These are typically passed as $\emptyset$, indicating no initial conditions outside of the existing formulation. However, additional constraints can be added to indicate forbidden routes, upper or lower bounds, etc.
- M: The maximum trip time allowed for each of the tours.
- w: A vector representing the waiting time associated with each of the nodes. Its size is equal to the number of sites to be visited, including the depot.

Algorithm 2 UAV DVRP Solver

1: **procedure** SOLVER(S, k, C_0, M, w)
2: $l \leftarrow 0$ ▷ l is the iterations counter
3: $C \leftarrow C_0$ ▷ C is the set of constraints, updated as subtours are added
4: $C_t \leftarrow \emptyset$
5: $c \leftarrow$ **graph**(S) ▷ build the weight matrix c used in constraint (16)
6: **repeat**
7: $C \leftarrow C \cup C_t$
8: $C_t \leftarrow \emptyset$
9: $x \leftarrow$ **Solve** (16) subject to (17), (18), (21), (22) with constraints C and graph c
10: $l \leftarrow l + 1$
11: $D, T \leftarrow$ **tours**(x) ▷ identify depot tours and subtours
12: **if** $T \neq \emptyset$ **then**
13: **for** $t \in T$ **do** ▷ eliminate all subtours
14: $C_t \leftarrow C_t \cup \sum_{i \in t, j \notin t} x_{i,j} + \sum_{i \notin t, j \in t} x_{i,j} \geq 2$
15: **if** $D \neq \emptyset$ **then**
16: **for** $d \in D$ **do**
17: **if time**(d, w) $> M$ **then** ▷ eliminate depot tours with time $> M$
18: $C_t \leftarrow C_t \cup \sum_{j \in d \setminus \{a,b\}} j + 0.45a + 0.45b \leq |d| - 1.55$ ▷ a,b depot edges
19: **until** $C_t = \emptyset$

Note that in Step 18, we choose a $\lambda = 0.45$ in the creation of our depot tour constraints arbitrarily. Other values of λ may be used, as long as they satisfy $0 < \lambda < \frac{1}{2}$.

5.2 Software

There are a variety of integer programming (IP) solvers available; several are free to the public, while others require expensive licenses. For our purposes, we have chosen to use IBM

ILOG CPLEX, a commercial solver which offers free academic licenses [9]. CPLEX has a built in GUI, CPLEX Optimization Studio, while also offering the ability to interface with several programming languages. MATLAB was chosen to interface because it was familiar software, though it does have certain limitations compared to other interfaces. One limitation is that advanced callbacks, commands that allow precise control over how CPLEX solves the IP, are not available.

5.3 Computational Setup

All computing was done on an AMD Opteron 6278 processor, with 256 GB of RAM in Windows Server 2008 R2 enterprise edition.

5.4 Computational Testing

Testing of our formulations was done over two separate data sets. Each set used independent entries, corresponding to geographic locations around the continental United States. The format for these tests is presented as the test set followed by the number of sites in that set.

- Set A: Sites correspond to state capitals of the continental United States. Instances ranged in size from 15 sites to 26 sites. The members of the subsets are randomly chosen from 26 available capitals.
- Set B: Sites correspond to the locations of volcanoes in the North-Western Continental United States. Instances ranged in size from 10 sites to 20 sites. The test set of size $n + 1$ is identical to the test set of size n plus an additional city.

For each set of data being tested, different permutations of the sets were taken with each iteration. This was done so that differing geometries of the graph produced could be examined. Note that for simplicity, the waiting times have been set to 0 for all sites in all examples during testing.

6 Results

In this section we present two example outputs of our MATLAB code, computational findings from running Algorithm 2 on the test sets discussed in Section 5.4, and a discussion of factors in convergence time of our implementation.To solve the integer programming formulation of our DVRP variant, (16)–(18), (21), (22), we call IBM ILOG CPLEX 12.5.1 from MATLAB (step 9 in Algorithm 2).

6.1 Example Mission Plan 1

To demonstrate that our implementation of Algorithm 2 correctly solves our DVRP, we present an example using test set A18 from the A set discussed in Section 5.4. Figures 2 and

3 show the flight routes coming from the planning algorithm applied with maximum trip times of $2J$ and $1.8J$, respectively. We calculate the distance from the depot (site 1) to the furthest site in the set, and set twice that to be J. We define J as such because J is a lower bound on the maximum trip time for which a solution could exist; if the trip time were below J there would be a site that could not be reached even with a vehicle going only to that site and back to the depot. Note that maximum trip times greater than J are not guaranteed to be feasible. Each example is run with 3 vehicles each traveling at 514 kilometers per hour, and for the A18 set, $J = 12.61$ hours.

Figure 2 represents a typical solution structure with high trip time where one UAV takes a much longer tour than the other two. The long flight is routed to sites 16, 4, 3, 2, 10, 5, 14, 13, 11, 12, 15, 17, 6, 7, 18 and then back to site 1; the others take single site tours out to site 8 and to site 9.

Figure 2: Solution of VRP with 18 sites and 3 vehicles. Maximum trip time is $2J$.

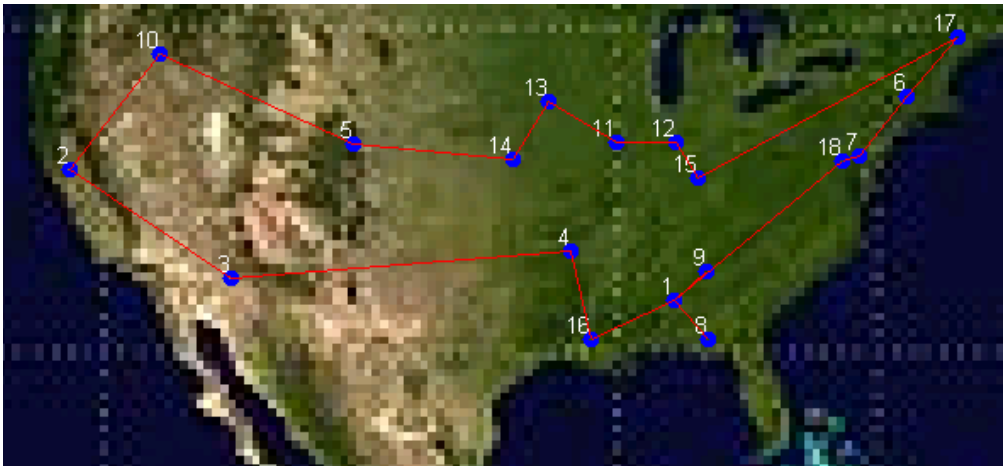


Figure 3: Solution of VRP with 18 sites and 3 vehicles. Maximum trip time is $1.6J$.

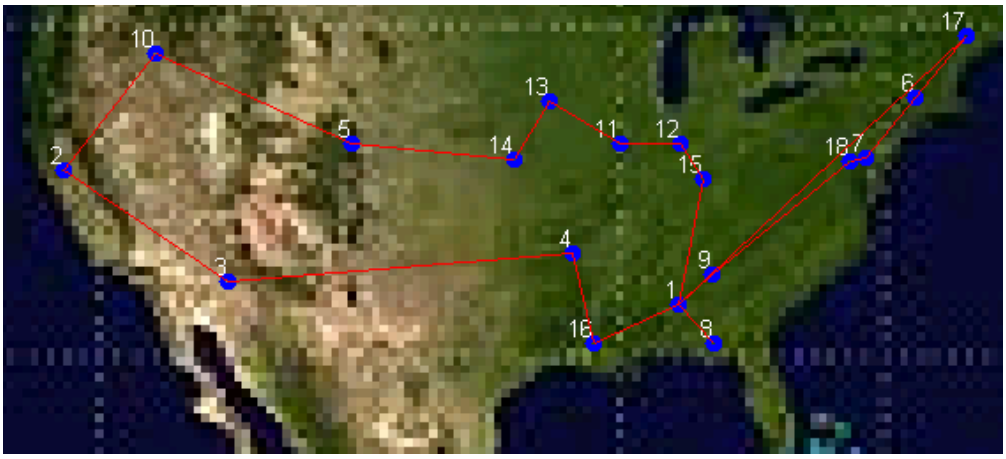
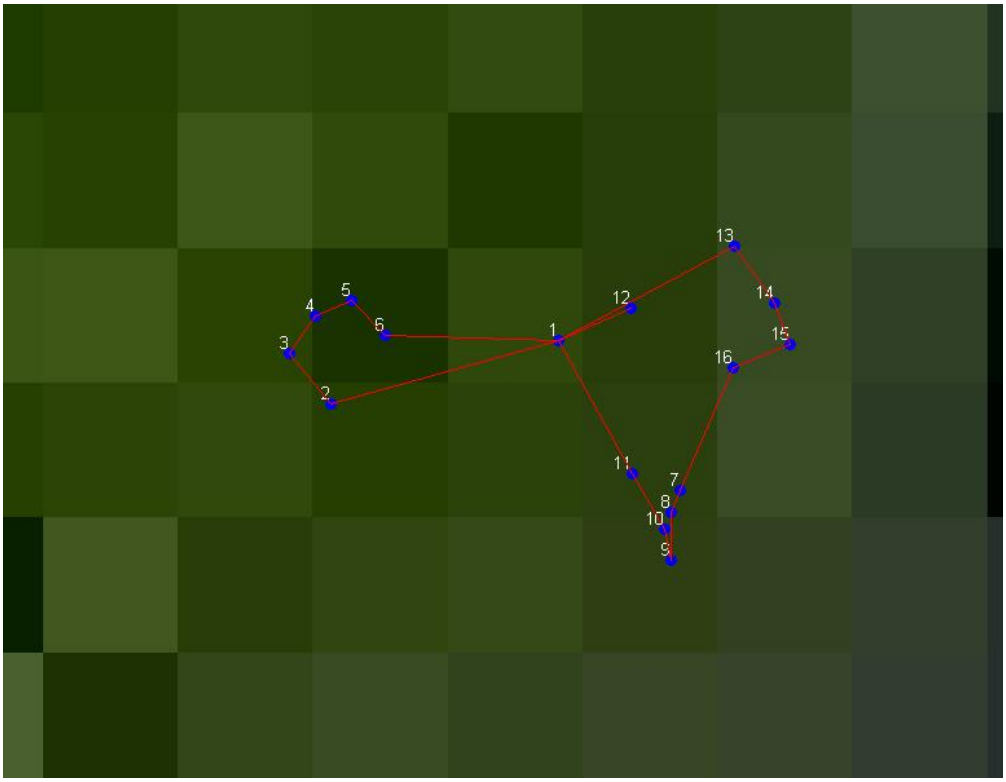


Figure 3 shows a different solution when the trip time is reduced enough to eliminate one or more depot tours. The tour from Figure 2 that traverses every site except sites 8 and 9 is no longer allowed since the time for that flight is greater than $1.6J$, and the vehicle routes are now significantly different. The first tour is a single site tour to 8 which is the same as in the solution associated with Figure 2. The second tour visits sites 1, 18, 7, 6, 17 and 9, and the third flight is routed through sites 16, 4, 3, 2, 10, 5, 14, 13, 11, 12 and 15.

6.2 Example Mission Plan 2

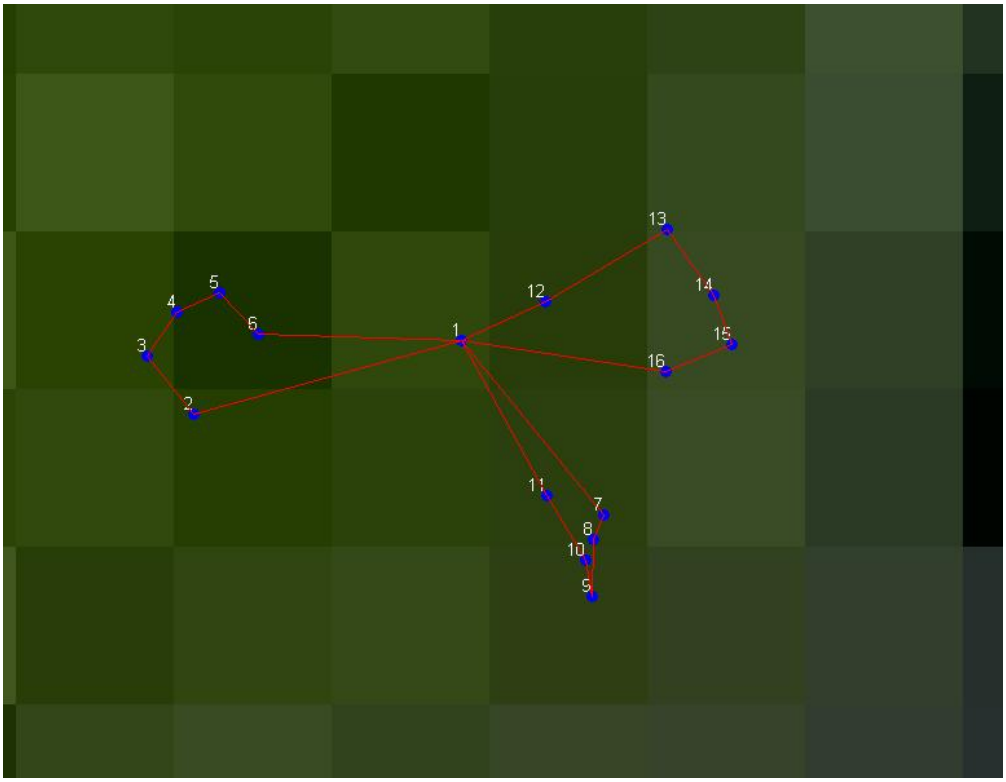
Figure 4: Solution of VRP with 16 sites and 3 vehicles. Maximum trip time is 0.5 hours.



The solutions depicted in Figures 4 and 5 represent the optimal route for 3 UAV's through a set of towns in Massachusetts and Rhode Island. The maximum trip times allowed are 0.5 hours and 0.4 hours, respectively. Figure 4 has a total travel time of 0.88 hours with the largest tour going through sites 11, 10, 5, 8, 7, 16, 15, 14, 13 and then back to site 1, the depot. The UAV completes this longest trip in 0.46 hours, just under the maximum trip time of 0.5 hours. The second tour in this solution routes the UAV through sites 2, 3, 4, 5, and 6. The trip time for this tour is 0.33 hours. The final tour depicted is from site 1 to 12 and back; this short tour takes only 0.09 hours. The algorithm took 2.52 seconds to converge to this solution.

When the maximum trip time is lowered to 0.4 hours, it is seen in Figure 5 that the

Figure 5: Solution of VRP with 16 sites and 3 vehicles. Maximum trip time is 0.4 hours.



sites are distributed more evenly among the three UAVs. The first UAV visits sites 12, 13, 14, 15, 16, the second visits 6, 5, 4, 3, and 2, and the last goes to 11, 10, 9, 8, and 7. The total flight time is 0.92 hours with each tour taking 0.33, 0.3, 0.28 hours. The algorithm took 41.2 seconds to converge to this solution.

This increased run time can be explained by the lower maximum trip time for the UAVs, as we discussed earlier. The set of feasible routes for the problem when M, the mission time, equals 0.4 is smaller than the set of feasible routes for the problem with $M = 0.5$, hence the minimum total flight time for $M = 0.4$ should be larger than that for $M = 0.5$. Our results reflect this. Also, the solution shown in Figure 5 when $M = 0.4$ is certainly a feasible solution when $M = 0.5$, but our algorithm correctly finds that a larger tour that directly connects site 7 (and 8, 5, 10, and 11) to site 16 (and sites 15, 14, and 13) along with the single destination tour from the depot to the closest site which is site 12, would result in a feasible solution with an even smaller total flight time.

6.3 Convergence Time Tests

Our implementation of the DVRP will find the optimal solution if an optimal solution exists but the number of iterations it takes to converge to such a solution can be extremely large. To examine this we ran the test sets detailed in Section 5.4 with decreasing maximum trip

times, $2J$, $1.8J$, $1.6J$, $1.4J$, with J as defined in Section 6.1. This avoids the potential issue involved with setting a global mission time that does not produce meaningful results because it is too high for the given data set. In Table 1, we present the wall clock time, in seconds, followed by the number of iterations it took for that set with that trip time to solve. If a given maximum trip time causes an instance to run for greater than 6,000 iterations, the instance was terminated and that entry is listed as "limit" in the table below. All test results are run with 3 vehicles each traveling at 514 kilometers per hour.

Test Set	$2J$	$1.8J$	$1.6J$	$1.4J$
A15	0.82(3)	0.84(3)	0.86(3)	106.83(120)
A16	0.91(3)	0.89(3)	0.86(3)	227.63(207)
A17	2.14(5)	1.9(5)	1.98(5)	limit
A18	4.25(5)	1.78(5)	4,847.26(900)	4,802.27(900)
A19	4.27(5)	3.79(5)	10,316.79(1,849)	limit
A20	3.46(6)	3.05(6)	9,379.78(1,590)	9,420.50(1,590)
A21	5.11(8)	31.75(24)	limit	limit
A22	4.56(7)	76.65(46)	195,479.13(5,640)	212,202.32(5,641)
A23	11.76(9)	limit	limit	limit
A24	7.93(7)	4,544.81(688)	limit	limit
A25	10.04(8)	27,761.14(2,504)	limit	limit
A26	5.94(8)	77,801.62(3,907)	limit	limit
B10	1.75(4)	1.67(4)	1.64(4)	1.58(4)
B11	1.68(4)	1.67(4)	1.68(4)	38.06(51)
B12	1.74(4)	1.65(4)	1.59(4)	41.12(51)
B13	1.74(4)	1.54(4)	1.54(4)	61.30(86)
B14	1.72(4)	1.54(4)	1.37(4)	67.01(71)
B15	1.15(4)	1.09(4)	0.99(4)	263.39(230)
B16	0.91(3)	0.84(3)	0.88(3)	0.09(3)
B17	1.39(3)	1.26(3)	1.12(3)	1.06(3)
B18	0.95(2)	0.82(2)	0.76(2)	72.67(70)
B19	0.98(2)	0.98(2)	0.98(2)	294.78(226)
B20	1.48(2)	1.14(2)	1.05(2)	361.34(267)

Table 1: Algorithm Testing

6.4 Discussion

Based on the results of our testing, we conclude that there are several significant factors that affect the convergence time of our algorithm.

The first of these factors is the mission time. For a fixed mission time, a reduction in the maximum trip time can drastically increase the convergence time. Row A19 of Table

1 illustrates this point effectively. This instance with $1.8J$ finished in 5 iterations, and a reduction by only $0.2J$ resulted in an increase of 1,844 iterations. This is because as the mission time is reduced, our implementation is forced to make more depot tour constraints. Since our depot tour constraints only eliminate one invalid depot tour, there exists the possibility that the next iteration will yield a solution with an equally invalid depot tour consisting of all of the same sites as the first but traversed in a different order, if that is still the optimal solution. Since the first depot tour was the shortest distance through those sites, and it was longer than the maximum trip time, this subsequent tour, and all similar tours must also be eliminated, each taking one iteration.

Similarly, another factor is the number of sites. If we examine the $1.8J$ column of Table 1, test set A goes from a convergence in 3 iterations for 15 sites to a convergence in 3,907 iterations for 26 sites. However, the $2J$ column barely increases at all over the same change in site number. Therefore, there appears to be a relationship between the number of sites and the maximum trip time of a vehicle. As the number of sites is increased, the number of iterations is increased significantly, provided there has been a reduction in trip time that eliminated one or more depot tours.

Another factor that drastically affects the convergence time is the geographic layout of the sites. For the B test sets, the first 3 trip times ($2J$,$1.8J$, $1.6J$) give convergence in under 5 iterations. The $1.4J$ trip time, however, gives extremely varied convergence rates. This variation is due to the construction method of the B test sets; B16 is simply B15 with one additional site added. The sudden drop in iterations from B15 to B16, when the maximum flight time is $1.4J$, is likely due to this additional site producing an optimal solution with tours that all satisfy the flight time constraint on the first iteration after subtours are eliminated. Since no tours need to be eliminated, no depot tour constraints are added. As discussed before, solving with depot tour constraints is extremely computationally taxing. Thus, this data suggests that the layout of the sites has a strong influence on the convergence time.

7 Future Work

The convergence time (in seconds or iterations) of our solution methodology is largely dependent on the mission time for a given vehicle; given a large mission time, solutions for sets of targets into the hundreds is possible with minimal CPU time. For example, as a stress test we ran the algorithm on a 100 site example with a mission time of 1,000 hours, and it reached a solution in 18 iterations. However, sets such as this become increasingly difficult to solve with lower mission time. For future work, we would like to address several directions that may improve the efficiency in finding an optimal solution, even when working with tight mission times.

First, in the execution of our Formulation 2 (described in Section 4.2.2), an integer program is solved each time Step 9 is executed. At successive iterations, the difference between the corresponding formulations is simply the addition of, at most, a few inequalities that forbid certain tours. In our present implementation, each time an integer program is solved at Step 9 of Algorithm 2, a new branch-and-bound tree is created and managed, including

progressing through that tree (solving LP relaxations at various nodes) until reaching the progress from the previous tree. These costs are significant, and could be entirely avoided by using the callback feature of integer programming solvers such as CPLEX [10] and Gurobi [11]. A callback would modify the branch and bound process so that the tree structure is preserved before a final solution is reached, rather than creating an entire new tree every time that Formulation 2 is solved. This presents the opportunity for significant computational savings, especially in instances where many iterations are required to find a final optimal solution. We believe both the subtour and depot tour constraints in our algorithm could be implemented with callback routines. Our implementation in MATLAB [12] does not allow for callbacks; however, alternative programming languages such as C++, Java, and Python do allow callbacks. We suggest development in one of these languages to take advantage of callbacks.

The second recommendation is to develop intelligent preprocessing methods. One common preprocessing technique is to cluster sites into several smaller instances, using point-to-point distance. The clusters are made up of sites that are relatively closer together in the original problem. For instance, if we were to look over a map of the United States, an efficient clustering algorithm would place sites in the northeast together and sites in the midwest together, creating two separate but smaller data sets from the original. Reliable methods to perform clustering are reviewed in [8]. Applying a clustering algorithm to the problem allows us to create k subsets of the original data, so that we can solve a relaxation of the original problem on each of the subsets then recombine the solutions. Using a clustering algorithm will allow for greater distribution of the network over several vehicles, as well as possibly increasing the solution speed on more tightly constrained problems.

Along with intelligent preprocessing methods, finding a lower bound on the number of vehicles necessary to solve the problem may help the algorithm perform significantly faster. Our solution method involves iteratively lowering the number of vehicles; each successive iteration becomes more difficult to solve, as discussed in Section 6.4. The final few iterations where the number of vehicles is approaching the lowest value it can attain before the problem is infeasible are extremely computationally taxing. If we knew, a priori, a lower bound then we would not attempt the computationally taxing iterations. Determining a lower bound may be possible through a variety of methods. For example, information from a minimum spanning tree over the tours of a previous solution may help to form a loose lower bound.

A final suggestion has more to do with an alternate form of our formulation. Presently, our approach minimizes the cumulative time to complete k tours. An alternative formulation is to minimize the maximum length of the individual tours, and letting the number of vehicles be determined dynamically by the objective function. This would be a minimax problem.

Appendices

A UAV Specifications

Provided below are the physical characteristics of the UAV's owned and operated by the National Aeronautics and Space Administration.

Vehicle	Max Altitude (ft)	Range (N. Miles)	Velocity (mph)	Wing Span (ft)	Length (ft)	Weight (lbs)	Sensor Capacity (lbs)
Ikhana-MQ9	50,000	1,000	230	66	36	4,900	2,000
Global Hawk	60,000	11,000	356.7	116	44	14,950	1,500

B Mathematical Techniques

B.1 Traveling Salesmen Problem (TSP)

A traveling salesman problem aims to solve an easily understood problem, assuming the distance between any two cities is known. The problem is to find the shortest route that starts from a home city, visits all other cities exactly once, and returns to the home city [2].

Notation

- E is the set of edges connecting the graph
- S is a subset of edges creating a subtour within the graph
- y_i is a binary variable indicating whether edge i is on (equal to 1) or off (equal to 0)
- c_i is the cost associated with taking edge i

Formulation of a Symmetric TSP

$$\text{minimize} \quad \frac{1}{2}\sum_{j=1}^{n}\sum_{k\in E_j} c_k y_k \tag{25}$$

$$\text{subject to} \quad \sum_{k\in E_j} y_k = 2 \quad \forall j \in E \tag{26}$$

$$\sum_{j\in E_s} y_j = |S| - 1 \quad \forall\, |S| = 2, 3, \cdots, n-2 \tag{27}$$

$$y_j = 1 \text{ or } 0, \quad \forall j \in E \tag{28}$$

$$\tag{29}$$

B.2 Solving the LP Relaxation Problem

The solution for the LP relaxation problem is found utilizing a modified simpelx method. As a refresher for the reader, we offer a brief overview of the Simplex method as well as pertinent vocabulary below.

- Objective function: The function that is either being minimized or maximized. For example, it may represent the cost that you are trying to minimize
- Constraints: A set of equalities and inequalities that the feasible solution must satisfy
- Feasible solution: A solution vector, x, which satisfies the constraints
- Optimal solution: A vector x which is both feasible and optimal
- Basic Solution: x of $Ax = b$ is a basic solution if the n components of x can be partitioned into m "basic" and $n - m$ "non-basic" variables in such a way that:
  - the m columns of A corresponding to the basic variables form a non-singular basis
  - the value of each "non-basic" variable is 0

  The constraint matrix A has m rows (constraints) and n columns (variables)
- Basis: The set of basic variables
- Basic variables: A variable in the basic solution (value is not 0)
- Non-basic Variables: A variable not in the basic solution (value is 0)
- Slack variable: A variable added to the problem to eliminate less-than constraints
- Surplus variable: A variable added to the problem to eliminate greater-than constraints
- Artificial variable: A variable added to a linear program in phase 1 to aid finding a feasible solution
- Unbounded solution: For some linear programs, it is possible to make the objective arbitrarily small (without bound). Such an LP is said to have an unbounded solution

Simplex Method Overview
The simplex method is an iterative algorithm with 3 main steps.

1. Initialization: Find an initial basic solution that is feasible.
2. Iteration: Find a basic solution that is better, adjacent and feasible.
   (a) determine the entering variable

(b) determine the leaving variable

(c) pivot on the pivot element and exchange variables, then update the tableau

3. Optimality Test: Test if the current solution is optimal if not repeat step 2

B.3 Branch and Bound Method

The branch and bound method is a straight forward way to solve IPs or MIPs. In essence, the process is to establish an initial feasible region, then slowly cut away optimal solutions until we find a best "candidate" solution. To determine the places at which we cut the feasible region[13]:

1. Solve the LP relaxation of the IP to find the optimum point. Set the upper bound for the problem as the optimum objective value.

2. Identify integer variables in the optimum solution of the LP relaxation that are not integer (and thus not feasible in the original problem) and branch on one of them.

3. Add an additional constraint for each branch, one where the branched variable must be the floor of its value in the LP relaxation, and another where it must be higher than the ceiling of its previous value. We then solve the LP relaxation again using these new constraints.

4. With that new value for the objective, we deem this as the next appropriate lower bound.

5. After each iteration, we should see the lower bound slowly moving up. (i.e. approaching the upper bound) Once we have branched out on the necessary variables, we pick the integer solution with the highest valued objective function.

The algorithm for the branch and bound method is given below, first a list of relevant variables is provided.

- S = the given IP problem
- S_{LP} = the LP relaxation of S
- y_{LP} = the solution to the LP relaxation of the given IP
- $\bar{Z}$= lowest (best) upper bound on Z^* of the given IP problem
- Z_0= highest (best) lower bound on Z^* of the given IP problem
- S^k= subproblem k of the problem S
- S^k_{LP}= the LP relaxation of subproblem k

- Z^k*= the optimum objective value of S^k
- $\bar{Z}^k$= the best (lowest)upper bound of subproblem S^k
- Z^k= the best (highest) lower bound of subproblem S^k
- y^k_{LP}= the optimal solution of the LP subproblem S^k_{LP}
- $\bar{y}_j$ = non integer value of integer variable y_j (current numerical value of y_j)
- $\lfloor a \rfloor$= the largest integer $\leq a$ (or rounding down a)
- $\lceil a \rceil$= the smallest integer $\geq a$ (or rounding down a)

We now describe the branch and bound algorithm

1. Step 1 (Initialization). Solve the LP relaxation (S^k_{LP}) of the given IP problem (S). If it is in-feasible, so is the IP problem, so terminate. If the LP optimum solution satisfies the integer requirement, the IP problem is solved, so terminate. Otherwise, initialize the best upper bound ($\bar{Z}$ by the optimal objective value of S_{LP} and the best lower bound by Z_0 = negative infinity. Place S^k_{LP} on the active list of nodes (subproblems) initially, there is no incumbent solution.

2. Step 2 (Choosing a node) If the active list is empty, terminate. The incumbent solution y^* is optimal. Otherwise, choose a node (subproblem) S^k with S^k_{LP} by one of the rules. (Depth first, best bound first etc.)

3. Step 3 (Updating Upper Bound) Solve the set $\bar{Z}^k$ equal to the LP optimum objective value. Keep the optimum LP solution y^k_{LP}

4. Step 4 (Prune by In-feasibility) If S^k_{LP} has no feasible solution, prune the current node and go to step 1. Otherwise go to step 4

5. Step 5 (Prune by Bound) If $Z^k \leq Z_0$ prune the current node and go to step 1. Otherwise go to step 5.

6. Step 6 (Updating Lower Bound and Prune for Optimality)

   - If the LP optimum y^k_{LP} is integer, a feasible solution to S is found an incumbent solution to the given problem. Set $\bar{Z}^k = y^k_{LP}$ and come compare $\bar{Z}^k$ with $\bar{Z}$. If $Z_{0k} < Z_0$ set $Z_{0k} = Z_0$ otherwise Z_0 does not change. The current node is pruned because no better solution can be branched down from this node. Go to Step 1.
   - If the LP optimum y^k_{LP} is non-integer, go to step 6.

7. Step 7 (Branching) From the current node S^k choose a variable y_j with fractional value generate two subproblems S^k_1 and S^k_2 defined by

   $S^k_1 = S^k \cap \{y : y_j \leq |\hat{y_j}|\}$
   $S^k_2 = S^k \cap \{y : y_j \leq |\hat{y_j}|\}$

Methods for choosing which variables to branch on:

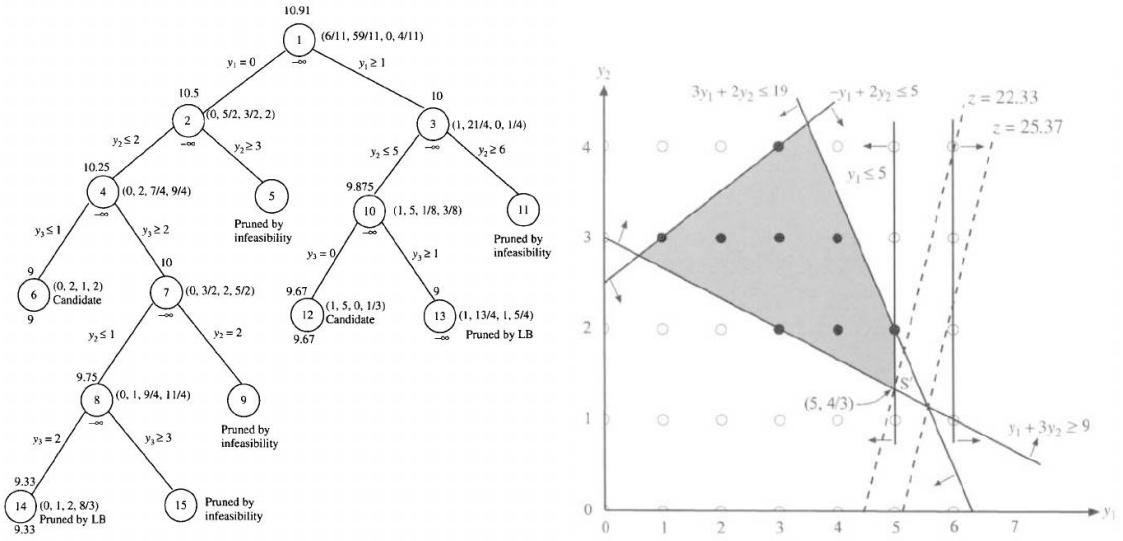
- The variable with fractional value closest to .5
- The variable with highest impact on the objective function value
- Variable with the least index

Suggested methods for choosing which un-pruned node to explore first.

- Depth First (solve newest generated subproblems first)
- Best bound principle (Go with highest Z on an active node)

Lastly, for the Bound and Branch Algorithm, we include images of the first cut and entire Branch tree.

Figure 6: Branch and Bound Tree and Example Cut



B.4 Branch and Cut Method

The Branch and cut method is the newest development in MIP solution algorithms. The method roots itself in making "tight" bounds before we prune or branch the node. These tight bounds are accomplished by generation of strong cuts, improving formulations, problem pre-processing, and applying a primal heuristic [13]. To describe the algorithm, we require the definition of some symbols.

- S = the given IP problem
- S_{LP} = the LP relaxation of S

- y_{LP} = the solution to the LP relaxation of the given IP
- $\bar{Z}$= lowest (best) upper bound on Z^* of the given IP problem
- Z_0= highest (best) lower bound on Z^* of the given IP problem
- S^k= subproblem k of the problem S
- S^k_{LP}= The LP relaxation of subproblem k
- $S^k(t)$= sub-problem k at iteration t
- $S^k_{LP}(t)$= sub-problem k at iteration t
- $y^k_{LP}(t)$= the optimal solution of the LP subproblem $S^k_{LP}(t)$
- y^*= the current incumbent solution
- $Z^k_{LP}(t)$= the optimum objective value of $S^k_{LP}(t)$
- $\bar{Z}^k$= the best (lowest) upper bound of subproblem S^k
- Z^k_0= the best (highest) lower bound of subproblem S^k
- t is the iteration number.

Now that we have the necessary symbols defined, we can move onto describe the algorithm in its entirety.

1. Initialization: Pre-process the given IP formulation. Solve its LP relaxation (S_{LP}). If S_{LP} if in-feasible, so is the IP problem, terminate. If the LP optimum solution satisfies the integer requirement, the IP problem S is also optimized, so terminate. Otherwise, set the best lower bound to negative infinity and the upper bound to Z^* for S_{LP}. Set $k = 1$. Let $S^k = S$. Place S^k on the active list of nodes (subproblems). Initially, there is no incumbent solution.

2. Choosing a node: If the active list is empty, terminate. The current incumbent solution, y^* is optimal. Otherwise, choose a node (subproblem) k with S^k. Remove S^k from the active list. Set iteration number $t = 1$. Denote the current subproblem by $S^k(t)$, which has LP relaxation $S^k_{LP}(t)$. Go to step 3.

3. Solving LP relaxation of the subproblem: Solve $S^k_{LP}(t)$. If it is in-feasible, prune node k and go to step 1. Otherwise, keep the optimal LP solution to $S^k_{LP}(t)$, which is $y^k_{LP}(t)$, and the optimal objective value $z^k_{LP}(t)$. Go to step 4.

4. Generating Cuts: Try to generate cuts on $S^k_{LP}(t)$ to cut off the point $y^k_{LP}(t)$. If no cut can be added, go to step 5. Otherwise, add a cut to $S^k_{LP}(t)$, resulting in a new LP problem $S^k_{LP}(t+1)$. Increase iteration number. Go to step 3.

5. Pruning: If $Z^k_{LP}(t) \leq Z_0$, prune node k and go to step 1. Otherwise, if $y^k_{LP}(t)$ satisfies all the integer requirements of the given IP problem, go to step 5. If $y^k_{LP}(t)$ violates some integer requirements go to step 7.

6. Updating Lower Bound: Since the optimal LP solution $y^k_{LP}(t)$ satisfies all integer requirements, a feasible solution to S is found and $y^k_{LP}(t)$ becomes a candidate solution. Set Z^k_0 to the optimal objective value of $S^k_{LP}(t)$ that is $Z^k_0 = Z^k_{LP}(t)$ and compare Z^k_0 with Z. If $Z^k_0 > Z_0$, set $Z_0 = Z^k_0$ and $y* = y^k_{LP}(t)$ becomes the incumbent; otherwise Z_0 does not change. Node k is pruned because no better solution can be branched down from this node. Go to step 2.

7. Branching: Branch on the current node k to create another subproblem S^{k+1}, S^{k+2} and so on. Place the new subproblem in the active list and go to step 2.

How the cuts are generated is what differentiates branch and cut from branch and bound.

1. Rounding Cut: This is the simplest of the three cut types and provides a cut that is effective on both IP's and MIP's. Often times, this provides a very weak cut. This can either be rounding down our answers as before with branch and bound, or we can use the GCD method, then rounding the right hand side down to its integer equivalent.

2. Disjunctive cut: This is a very effective and popular way to produce cuts for MIP's or IP's. It uses an optimal solution that we then take the corresponding variable from and round down to achieve an integer result for either y or x. The variable we do not seek is set to whatever value it must be to achieve the rounding of the dominant variable. We then connect the points created through rounding and use this line as our cut.

3. Lifting technique: This is especially useful for binary IP's however it is not nearly as effective for MIP's or IP as the disjunctive cut is.

References

[1] M. Abramson, D. Carter, and S. Kolitz. *The design and implementation of Draper's Earth phenomena observing system (EPOS)*, in AIAA Space Conference and Exposition. Albuquerque, NM, 2001. American Institute of Aeronautics and Astronautics.

[2] R. Matai, S. P. Singh, and M. L. Mittal. *Traveling Salesman Problem, Theory and Applications*. InTech, Rijeka, Croatia, 2010.

[3] R. Karp. *Reducibility among combinatorial problems*, in Complexity of Computer Computations. Yorktown Heights, New York, 1972. IBM Research Mathematical Sciences Department.

[4] G. Dantzig and J. Ramser. *The truck dispatching problem.* Management Science, 6(1):pp.80–91, 1959.

[5] G. Clarke and J. W. Wright. *Scheduling of vehicles from a central depot to a number of delivery points.* Operations Research, 12(4):pp.568–581, 1964.

[6] S. Almoustafa. *Distance-Constrained Vehicle Routing Problem: Exact and Approximate Solution.* PhD thesis, School of Information Systems, Computing and Mathematics Brunel University, Uxbridge, England, 2013.

[7] G. Laporte, Y. Nobert, and M. Desrochers. *Optimal routing under capacity and distance restrictions.* Operations Research, 33(5):pp.1050–1073, 1985.

[8] G. Laporte. *The vehicle routing problem: an overview of exact and approximate algorithms.* European Journal of Operational Research, 59(3):pp.345–358, 1992.

[9] B. Meindl and M. Templ. *Analysis of commercial and free and open source solvers for linear optimization problems.* Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS, 2012.

[10] IBM. *IBM ILOG CPLEX 12.5.1 Release Notes.* IBM ILOG CPLEX Division, Incline Village, NV, 2012.

[11] Gurobi Optimization Inc. *Gurobi Optimizer Reference Manual.* Gurobi Optimization Inc., Houston, TX, 2014.

[12] MathWorks. *MATLAB Primer.* The MathWorks Inc., Natick, MA, 2014.

[13] D. S. Chen, R. G. Batson, and Y. Dang. *Applied Integer Programming Modeling and Solution.* John Wiley and Sons, Hoboken, NJ, USA, 2011.