UNMANNED AERIAL VEHICLE ROUTING PROBLEM WITH LIMITED RISK

by

SIRIWAT VISOLDILOKPUN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2008

This dissertation is dedicated to my father and my sister,

and to the loving memory of my mother.

ACKNOWLEDGEMENTS

I want to thank my advisor, Dr. Jay M. Rosenberger, for his supports during my doctoral studies. He introduced me to a very interesting research topic and also guided me through many challenges. Difficult tasks became easy because of his invaluable advice. I am truly grateful for such an optimistic and understanding advisor. I want to thank the members of my dissertation committee, Dr. H. W. Corley, Dr. Jamie Rogers, and Dr. Eli V. Olinick, for their encouragement and interest in my research. It is my honor to have them as my committee. I would like to extend my sincere gratitude to Dr. Victoria C. P. Chen, Dr. Seoung Bum Kim, and Dr. H. W. Corley for their kindness. There were countless of times that they went above and beyond to be very nice to me even when they did not have to. Thanks go especially to Dr. H. W. Corley, who recommended me to work with Dr. Rosenberger, and whose gracious words tremendously lifted my spirit. It is my privilege to have known, worked with, and studied with these wonderful people.

I want to extend my appreciation to the COSMOS lab graduates (Drs. Wen, Siddappa, Pilla, Sung, Shih, Hwang, Punnakitikashem, Sundaramoorthi, Fan, Tarun, Temiyasathit, and Sukchotrat), the current COSMOS lab students (Panaya, Subrat, Weerawat, Poovich, Chingfen, Passakorn, Chatabush, Narakorn, and Surachai), and my friends (Ta, Kenneth, Bodin, Punnapob, Jason, Ake, Sanya, Panita, and Phayak). Their friendships made my stay in Arlington an enjoyable experience. I also want to thank a wonderful Thai couple, Jessada and Suthenee Virattanajun, who kindly looked after me while I was here.

I owe a great debt to Prattana Punnakitikashem, with who I shared much laughter and tears. It was a tough long journey but it was pleasurable because of

her. I want to thank her for her faith and for seeing something more in me. My appreciation to her is always kept dearly in my heart.

Last, but certainly not least, I want to thank my entire family for their unconditional love and unwavering support. I want to thank my dad, Montri, for his constant support, both morally and financially, and for his infinite patient which allowed me to take as much time as I needed to reach my goal. I eternally thank him for being the rock of our family who stood by us through the best and worst of times. I want to thank my mom, Supaporn, for her undying love. I thank her for instilling within me great compassion and a wonderful sense of humor. I know how proud she would be if she were here with me today. Her trust in me motivated me to keep going in the face of great difficulties during my doctoral studies. I want to thank my sister, Sivinee, for knowing what to say and when to say it. She has been more than a sibling; she is also my best friend. Her voice of encouragement was never late and always came when I needed it the most. I also want to thank my brother-in-law, Nisapol, for taking care of my father and sister while I was away on this journey. He is a wonderful addition to our family and I could never ask for more.

I am so fortunate to have a family that never, for a single moment, gave up on trying to make me a better person. I would not be where I am today if it was not for my family, and yes I do love them very much.

November 15, 2008

ABSTRACT

UNMANNED AERIAL VEHICLE ROUTING PROBLEM
WITH LIMITED RISK

Siriwat Visoldilokpun, Ph.D.

The University of Texas at Arlington, 2008

Supervising Professor: Dr. Jay M. Rosenberger

Due to safety and uninhabitable surrounding environments, long range Unmanned Aerial Vehicles (UAVs) have become an increasingly popular option in both scientific and military applications. With troubling oil prices, a logistic planner requires an efficient method to identify an optimal routing solution. The solution should not only optimize the bottom line objective but must also control variations in an operation.

We study the UAV routing problem with limited risk (URPR) in which the considered risk is a fuel burn variance caused by wind variation. The URPR determines the optimal route that minimizes total expected fuel burn to visit all assigned targets while maintaining the variability to a constant parameter. The URPR is modeled as a set-partitioning problem with a quadratic variance constraint (SPQC). However, the quadratic variance constraint is simplified to a single linear variance constraint. In this study, we discuss two types of the URPR, which are the classical URPR and the URPR with time windows (URPRTW). We present algorithms in a Branch-and-Cut-and-Price methodology to solve the URPR, and the URPRTW. Within the BCP methodology, variables with negative reduced costs are generated to be added to the restricted master problem (RMP) in the pricing step, and minimum dependent set

(MDS) constraints are generated in the cutting step to encourage integrality of a solution. Minimum-cost path algorithms such as the Dynamic Programming Shortest Path (DPSP) and the Integer Programming Shortest Path (IPSP) were implemented as column generation engines in the pricing step. Although, the DPSP algorithm quickly found optimal solutions for the URPRTW, it is not applicable in the URPR due to computational complexity. The generating of MDS constraints shows competitive results in the URPRTW and impressive results in the URPR in terms of a computational time. In the conducted computational experiments, medium-sized URPRTWs and small-sized URPRs were optimally solved.

Furthermore, we discuss a special case of SPQC with a different variation model where the quadratic constraint is irreducible. We propose the Delayed Columns-and-Cuts Generation algorithm (DCCG) to solve the special case SPQC. The algorithm solves the continuous relaxation of special case SPQC in branch-and-bound nodes. A cut selection strategy adds two types of cuts to cut off infeasible solution. Finally we discuss future extensions of this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Unmanned Aerial Vehicles (UAV) are an attractive alternative for many scientific and military organizations. A UAV can perform operations that are considered to be risky or uninhabitable for humans. The operations range from monitoring ozone depletion, inclement weather, and traffic congestion, taking images of dangerous territory, and dropping bombs in war zones. The U.S. spent about $680 millions in 2002 on UAV development programs in military applications. The amount went to more than double in 2005. The worldwide spending on UAV programs is likely to reach $3.35 billions in 2012 [1]. The two major U.S. organizations in UAV development are the Department of Defense (DOD) and the National Aeronautics and Space Administration (NASA). In this early stage of UAV development, the long range communication between UAVs and their base is very difficult due to the nature of the UAV application. Hence, a routing for UAVs is usually predetermined before the operation starts. The static plan for UAV routing is very vulnerable to changes and uncertainties that happen after the operation starts.

There are several major contributors for uncertainties in air transportation, however none is considered to have more influence than the weather. Despite, recent technical developments on aircraft and navigational systems, inclement weather is still capable of closing airports, canceling flights, which disrupt an entire schedule of airliners. Although accurate forecasting is required, it is almost impossible due to the chaotic nature of the weather. This implies that we may not be able to totally prevent the effects caused by the weather in air transportation. Nevertheless, in the routing decision making, we should try to maintain limited variability of the planned

route. In this research, we consider the wind effect to be the only major uncertainty. Obviously, wind has more of an effect on a UAV than it does on a passenger aircraft because of their cruising speed and weights. Traveling with an unexpected head wind or tail wind can significantly affect the operating cost of a UAV operation. Usually, in UAV routing, UAVs must visit their targets within their specified time windows. This means that a UAV must increase its air speed in order to maintain its scheduled ground speed when it flies a route with stronger head wind. In contrast, a UAV can decrease its air speed in order to maintain its scheduled ground speed and save its fuel. Efficiently choosing routes can significantly reduce the total amount of fuel burn a UAV consumes to visit all of its targets.

In this research, we study a UAV routing problem with limited risk (URPR) caused by wind variation. The URPR is modeled as a vehicle routing problem (VRP) using the set-partitioning problem with a quadratic constraint (SPQC). Although the traditional VRP for transportation and logistics networks is well-studied in the optimization literature, only minimal research has considered optimally routing UAVs. The URPR minimizes the total amount of fuel burn to visit all targets and maintain the fuel burn variation caused by the wind effect to a predetermine constant $d \geq 0$. We propose algorithms in a Branch-and-Cut-and-Price methodology (BCP) to solve the URPR. The algorithms solve the continuous relaxation of the URPR (CURPR) in a branch-and-bound tree to obtain the optimal integer solution. With different values of d, we can construct an efficient frontier that represent the relationship between the total fuel burned to visit all targets and their corresponding variations. Logistic planners can access their risk preferences and identify the best solution from the efficient frontier.

1.2 Dissertation Objectives and Outline

Although the URPR is similar to a vehicle routing problem, most research on routing of unmanned aerial vehicles focuses on quickly generating an acceptable solution using heuristics and simulation techniques. In our research, the objective is to develop and implement an exact solution method to the URPR. The proposed method is within the BCP methodology, which models the URPR as a SPQC. The proposed method is not only capable of solving the URPR, which usually has an enormous number of variables, but also able to control variations in UAV operations, which is a measurement that classical routing problem usually ignore. We consider implementing the proposed method on both the URPR and the URPRTW. Because of time window requirements, the URPRTW is easier to solve than the URPR. Expectedly, the proposed algorithms should be able to optimally solve larger instances of the URPRTW than those of the URPR. Furthermore, we improve computational efficiency of the proposed method by adding a new valid inequality that encourages integrality in a solution. Finally, we will generalize the proposed method

The outline of this dissertation is as follows. In Chapter 2, we give a brief background on the VRP, the vehicle routing problem with time windows (VRPTW), the stochastic vehicle routing problem (SVRP), the UAV routing problem, and the Branch-and-Cut-and-Price methodology. In Chapter 3, we formulate the URPRTW as a set-partitioning problem with a quadratic constraint that limits variance of fuel burn to a predetermine constant d. We show that the quadratic constraint is reducible to a single linear constraint. Furthermore, we present a new valid inequality that can be derived from a 0-1 knapsack constraint to encourage integrality in a solution. We discuss three algorithms in the Branch-and-Cut-and-Price framework to solve the URPR problem. Computational experiments on medium-sized URPRTW are discussed, and results are presented. In Chapter 4, we present a special case of the SPQC where the variation in a routing solution is modeled differently and hence the

quadratic constraint is irreducible. Finally, we present a Delayed Column-and-Cut algorithm to solve the special case of SPQC. In Chapter 5, we discuss the URPR without time windows. We show that although the set-partitioning models of the URPR and the URPRTW are the same, a new column generation algorithm is required. Computational experiments are done on a small-sized URPR, and results are discussed. Finally, in Chapter 6, we discuss contributions and some future extensions of our research.

CHAPTER 2

LITERATURE REVIEW

2.1 Vehicle Routing Problem

The vehicle routing problem (VRP) is a combinatorial optimization problem that determines a set of minimum cost routes from one or multiple *depots* for a fleet of homogeneous vehicles to service a set of geographically scatter customers. Vehicles are dispatched from a depot to deliver goods or services, and then return back to the depot. Each route starts and ends at the same depot, all customers are visited exactly once, and the total demand of customers in a route must not exceed the vehicle capacity. In practice, the objective is equivalent to minimizing a total traveling distance, or minimizing a total number of vehicles used. The VRP plays a major role in distribution managements, and was originally described in Dantzig and Ramser [2] as the *Truck Dispatching Problem*. The VRP is *NP*-hard, which means that its computational complexity exponentially increase when the problem size increases. Although, the VRP is well-lknown and has attracted attentions of researchers for a long time, it is still largely unsolved [3] and exact algorithms are inefficient to problems with more than 50 customers [4].

The VRP is usually defined on a graph $G = (V, E)$, where $V = \{v_0, v_1, v_2, ..., v_n\}$ is a set of nodes and $E = \{e_{ij} = (v_i, v_j) : i \neq j \text{ and } v_i, v_j \in V\}$ is a set of edges. Node v_0 represents a common depot, and the remaining nodes $v_1, v_2, ..., v_n$ represent customers associated with nonnegative demands $d_1, d_2, ..., d_n$, respectively. For each edge e_{ij}, there is a nonnegative distance, c_{ij}, for a vehicle to travel the edge. In other contexts, c_{ij} could represent a travel cost or a travel time. Vehicles are homogeneous and have a capacity of h. A vehicle route starts at the depot, visits customers, and then returns back to the depot. A route is commonly represented as

$f = \langle v_1(f), v_2(f), v_3(f)... \rangle$ where $v_i(f)$ represents the i^{th} customer in route f. Therefore, a total demands served by each route, $d(f)$, must not exceed the vehicle capacity, that is $d(f) = \sum_{i \in \Omega(f)} d_i \leq h$, where $\Omega(f)$ represents a set of customers visited in route f. The VRP is often formulated as an integer set-partitioning problem

$$\min c^T x \tag{2.1}$$

$$\text{s.t. } Ax = 1 \tag{2.2}$$

$$x \in \{0,1\}^n, \tag{2.3}$$

where A is a constant 0-1 $m \times n$ matrix, c is a constant $n \times 1$ vector, 1 is a $m \times 1$ vector of ones, and x is a 0-1 $n \times 1$ vector. Each row in the matrix A represents a customer, while each column represents a route for a vehicle. An entry $[a_{ij}]$ in matrix A is one if customer i is serviced by route j, while a zero indicates otherwise. Typically, the objective coefficient $[c_j]$ in the vector c is cost, distance, or time of route j. Finally, the binary vector x indicates routes that are selected in a solution of the VRP. A major difficulty of this formulation is that, even with a medium size problem, there could be an enormous number of possible routes. In Chapter 3, we propose an algorithm to overcome this difficulty. Usually, to reflect real word applications, many side constraints are added to the classical VRP. Some of the important variations to the classical VRP are capacitated VRP (CVRP), multiple depots VRP (MDVRP), split delivery VRP (SDVRP), VRP with pick-up and delivery (VRPPD), VRP with time windows (VRPTW), and stochastic VRP (SVRP).

According to Laporte and Nobert [5], exact algorithms for the VRP can be classified into three categories (i) direct tree search method, (ii) dynamic programming, and (iii) integer linear programming. Laporte et al. [6] exploited the relationship between the VRP and one of its relaxations, the m-TSP. The VRP was solved in a branch-and-bound tree where subproblems were assignment problems and subtour elimination constraints were considered. The algorithm optimally solved asymmetrically CVRP up to 260 customers. Christofides et al. [7] developed an algorithm

based on k-degree center tree relaxation of the m-TSP. They successfully solved the VRP ranging from 10 to 25 customers. Eilon et al. [8] was the first to use dynamic programming to solve the VRP. Lower bounds on optimal solutions of the VRP were obtained were varied between 93.1% and 100% for problems up to 25 customers. Christofides [9] could solve the VRP with the same approach up to 50 customers. Belinski and Quandt [10] were one of the first to consider using set-partitioning model to formulate the VRP. The column generation technique was used to overcome difficulty due to an enormous number of variables. This procedure was successfully applied to the classical VRP and its variants by Foster and Ryan [11], Agarwal et al. [12], Desrosiers et al. [13], and Desrochers et al. [14].

As mentioned, the VRP becomes computationally intractable when the number of customers is large, it is often desirable to implement heuristic methods to quickly obtain *good* solutions. Consequently, developments of heuristic algorithms for the VRP is usually a primary interest. Heuristic algorithms using simulated annealing for the VRP includes Alfa et al. [15], Golden and Skiscim [16], Hiquebran et al. [17], and Osman [18], [19]. Buxey [20] implemented Monte-Carlo simulation, while Gendreau et al. [21], and Pureza and Franca [22] considered tabu search methods. Kinderwater and Savelsbergh [23], Savelsbergh [24], and Thompson [25] implemented local search methods and Kopfer et al. [26] used genetic algorithm as heuristic algorithms to solve the VRP. In the following sections, we discuss some important variants of the VRP.

2.1.1 Vehicle Routing Problem with Time Windows

In this section, we discuss an important variation of the classical VRP, which is the VRP with time windows (VRPTW). The VRPTW is similar to the VRP however each customer is associated with time windows. Usually, for the VRPTW defined by a graph $G = (V, E)$ where $V = \{v_0, v_1, ..., v_n\}$ is a set of customers and $E = \{e_{ij} = (v_i, v_j) : i \neq j \text{ and } v_i, v_j \in V\}$ is a set of edges, the time windows for

customer v_i are represented by $[o_{v_i}, p_{v_i}]$. To service a customer v_i, a vehicle must reach the customer location before the starting service time o_{v_i} and remain at the customer location during the service until p_{v_i}. Typically, vehicles may arrive early, but they have to wait until time o_v before servicing a customer. Given that node v_0 represents a depot, o_{v_0} is an earliest time that a vehicle can leave the depot, and p_{v_0} is the latest time that a vehicle must return to the depot, a feasible solution to the VRPTW exists if and only if $o_{v_0} \leq \min_{i \in V \setminus \{v_0\}}(o_{v_i} - t_{0i})$ and $p_{v_0} \geq \min_{i \in V \setminus \{v_0\}}(p_{v_i} + t_{i0})$, where t_{ij} represents a travel time from customer v_i to customer v_j. An edge e_{ij} can be eliminated if it violates temporal requirements, such as $p_{v_i} + t_{ij} > o_{v_j}$. Usually, the VRPTW is modeled as a multi-commodity network flow problem with additional sets of time window constraints and capacity constraints. However, just to find a feasible solution of the VRPTW is very difficult and is an *NP*-complete problem [27], hence recent research focuses on developing heuristics that can solve a realistic size VRPTW.

According to Cordeau et al. [28], heuristic algorithms include (i) route construction heuristics, (ii) route improvement heuristics, (iii) composite heuristics, and (iv) metaheuristics. The route construction heuristic constructs a route by inserting a new customer into an existing route. The insertion concerns conditions such as additional distances and time savings. In Solomon [29], a sequential insertion heuristic was proposed. Other studies using the route construction heuristic are Solomon [30] , Potvin and Rousseau [31]. The route improvement heuristic modifies the current solution using local search methods by swapping subset of edges between solutions. The heuristic is considered to be very effective, but time consuming. Baker and Schaffer [32], and Russell [33] are the early implementations of the route improvement heuristic to the VRPTW. Other successful implementations are Savelsbergh [27], [24], [34], and Kinderwater and Savelsbergh [35]. The composite heuristic is a combination between the route construction and the route improvement heuristics. Kontoravdis

and Bard [36] and Russell [37] developed algorithms that embed route improvement within a route construction process. Finally, metaheuristics include simulated annealing, tabu search, and evolutionary algorithms. There are many implementations of metaheuristics to the VRPTW. Tabu search examples include Taillard et al. [38], Rochat and Taillard [39], and Taillard [40], and examples of evolutionary algorithm include Homberger and Gehring [41], Potvin and Bengio [42], and Blanton and Wainwright [43].

2.1.2 Stochastic Vehicle Routing Problem

The stochastic VRP (SVRP) is the VRP with one or several random components in the problem. The most common random components are stochastic customers, where each customer v_i will be present with a probability p_i, stochastic demands, where demand for each customer v_i is random, and stochastic times, where service times or travel times are random. Usually, solving the SVRP requires a *Stochastic Program with Recourse* (SPR) in which the first stage solution is determined before random variables are realized. In the second stage, a recourse decision is taken based on the random event and the first stage solution. In the SPR, the objective to minimize the total expected cost.

Laporte and Louveaux [44] developed the *Integer L-Shaped* method that allows the SVRP with recourse function to be solved. Stewart and Golden [45] presented several formulations and heuristics to the SVRP with stochastic demands. Laporte et al. [46] studied a location-routing problem in which the first stage depot locations, fleet size, and routes are determined. Vladimirou and Zenios [47], Morton and Wood [48], Beraldi et al. [49], and Sherali and Fraticelli [50] provided formulations and algorithms for the SPR. For other references on the SVRP see Bertsimas [51], and Bertsimas et al. [52], and for recent survey see Gendreau et al. [53].

2.2 Unmanned Aerial Vehicle Routing Problem

Unmanned Aerial Vehicles (UAVs) have received significant attention in recent years. A UAV can replace manned aerial vehicles in unsafe and uninhabitable situations. Applications for UAV include both scientific and military purposes, such as monitoring ozone depletion, monitoring inclement weather, monitoring traffic congestion, and dropping bombs in a war zone. Because of the nature of UAV applications, a UAV operation is often disrupted and long range communications between UAVs and their base are very difficult. Consequently, most UAV routing plans are static and predetermined before the operation starts. In addition, UAV operations are vulnerable to uncertainties. For example, in military applications, risks in UAV operations include a probability of being shot down, and stochastic targets. In scientific applications, risks include mechanical failures, and infeasible routes due to increment weather. From these reasons UAV routing is very uncertain and requires an efficient solution method that not only find optimal routing plan but also controls variations in a UAV opration. According to literature, UAV routing problem is normally modeled as a traditional vehicle routing problem where side constraints will be added to reflect natures of applications. In early years, most research on UAV routing mainly focused on developing heuristics and simulations to find a quick solution with acceptable quality. Some studies that focused on developing exact solution methods principally examined static scenario, and ignored variability in the problem.

Sisson [54] determined a minimum number of predators required to cover assigned targets. The minimum risk route and expected number of covered targets were identified using tabu search method coupled with Monte Carlo simulation. O'Rourke et al. [55] examined the use of metaheuristic to solve dynamic UAV routing problem. Using information on wind, target, and threat information, the metaheuristic based on reactive tabu search was able to dynamically generate and update UAV routes in real-time. Ryan [56] used stochastic simulation to produce a robust tour for UAV to

visit targets with unknown threats and wind condition at a time of mission execution. Russell and Lamont [57] presented a more precise mathematical model of Genetic Vehicle Representation (GVR). Their results showed that Genetic Algorithm (GA) could produce high quality solution in a relatively short time for a dynamic routing problem with swarms of UAVs. Corner and Lamont [58] implemented a discrete event simulation to study the UAV routing problem. Shetty et al. [59] considered a strategic routing of a fleet of unmanned combat aerial vehicles to service a set of targets where targets were characterized by their priorities or importance levels. In this study, a tabu search heuristic was developed to coordinate a target assignment problem with a vehicle routing problem. Rathinam and Sengupta [60] extended Held-Karp's lower bound available for a single Traveling Salesman Problem to the Multiple Depot UAV Routing Problem. Other related literature on UAV task allocation can be found in [61], [62], [63], [64], [65], and [66].

2.3 Branch-and-Cut-and-Price Methodology

The Branch-and-Cut-and-Price Methodology (BCP) is an LP-based branch and bound algorithm in which a continuous relaxation of a discrete optimization problem (DOP) is solved within a node of a branch-and-bound tree. The algorithm uses a divide-and-conquer strategy to partition a solution space of the DOP into subproblems usually by imposing bounds to variables, this step is called *branching*. A subproblem will be discarded, or *pruned*, if either the subproblem has no feasible solution or the subproblem has a feasible solution that is greater than or equal to the current upper bound. However, if the subproblem has a solution that is lower than the current lower bound, then the new lower bound is obtained, otherwise branching must be performed to the subproblem to create new subproblems, called *children*. The process continues until there is no more subproblem to be processed.

The BCP methodology is incorporated with a *cutting* step and a *pricing* step. The cutting step increases an efficiency of the BCP methodology by adding valid inequalities to tight the relaxation so that the relaxed sets closely approximate feasible regions of the subproblems. By include only a subset of variables to the relaxation, the pricing step can improve an efficiency of the BCP methodology by generating new variables to be added to the relaxation only when they are needed. The technique bases on column generation techniques such as the Dantzig-Wolfe decomposition. Therefore the BCP methodology is very efficient with a DOP with a large number of variables since only a subset of variables will be considered. Implementations of the BCP methodology can be found in Applegate et al. [67], and Eso et al. [68], [69]. Implementations of the Branch-and-Cut algorithm are [70], and [71], and implementations of the Branch-and-Price algorithm are Barnhart et al. [72], and Savelsbergh [73].

In the next chapter, we discuss the UAV routing problem with limited risk. We present an implementation of the BCP methodology to the problem and computational results are discussed.

CHAPTER 3

UAV ROUTING PROBLEM WITH TIME WINDOWS

3.1 Introduction

Risk has become a major part of decision making. Fluctuations in the global economy, threats of terrorism, war, and soaring oil prices are major sources of instability. As this variation increases, a logistic planner must find a method that not only optimizes the bottom line profit but also controls variation in its operation. In this section, we formulate the UAV routing problem with limited risk as an integer set-partitioning problem with a quadratic constraint that limits the variance of fuel burn affected by wind variation. The proposed model is a revised version of the set-partitioning model for the VRP in (2.1)-(2.3) by adding the quadratic covariance constraint, which is a measure that most traditional VRPs usually ignore. The set-partitioning problem with the quadratic constraint (SPQC) is

$$\min c^T x \tag{3.1}$$

$$\text{s.t. } Ax = 1, \tag{3.2}$$

$$x^T Q x \leq d, \tag{3.3}$$

$$x \in \{0,1\}^n, \tag{3.4}$$

where Q is a symmetric positive semi-definite matrix in which its entry $[q_{ij}]$ is a covariance from selecting both route i and route j, simultaneously, and d is a non-negative constant that limits the amount of variation in a feasible solution. In this study, the fuel burn variation is considered as a risk in a UAV operation, therefore q_{ij} is a fuel burn variation from selecting route i and route j, simultaneously. The considered problem in this chapter is the UAV routing problem with limited risk with time windows (URPRTW). The URPRTW determines a set of routes that minimizes

the expected amount of fuel burn for a fleet of homogeneous UAVs to visit all targets within specific time windows, while maintains an operational risk at less than a predetermine amount d. UAVs are required to start and finish their routes at the base. Only one UAV is allowed to visit each target, and time windows for both UAVs and targets must be respected.

Let U be a set of groups of UAVs to be scheduled, where a group of UAVs is defined as a set of homogeneous UAVs located at the same location. For each group $u \in U$, let n_u denotes the total number of homogeneous UAVs in group u, and let F_u denote a set of possible routes of UAVs in group u. Therefore, the set F, where $F = \bigcup_{u \in U} F_u$, denotes the set of all possible routes for all groups. Let K be the set of all targets to be visited, and let L be the set of all links between available targets. Note that an edge and a link are the same and used interchangeably through out this study. The binary constant $a_{kuf} = 1$ indicates that a target $k \in K$ is visited by a UAV in group $u \in U$ in a route $f \in F_u$. For each link $l \in L$, a random variable $\tilde{e}_l$ denotes the total amount of fuel burn for traveling link l. Consequently, the total fuel burn for a UAV u in a route f can be calculated by

$$\tilde{e}_{uf} = \sum_{l \in f} \tilde{e}_l. \tag{3.5}$$

The integer programming formulation of the URPRTW can be written as

$$\min \sum_{u \in U} \sum_{f \in F_u} E\left[\tilde{e}_{uf}\right] x_{uf} \tag{3.6}$$

$$\text{s.t.} \sum_{u \in U} \sum_{f \in F_u} a_{kuf} x_{uf} = 1 \qquad \forall k \in K, \tag{3.7}$$

$$\sum_{f \in F_u} x_{uf} \leq n_u \qquad \forall u \in U, \tag{3.8}$$

$$var\left(\sum_{u \in U} \sum_{f \in F_u} \tilde{e}_{uf} x_{uf}\right) \leq d, \tag{3.9}$$

$$x_{uf} \in \{0, 1\} \qquad \forall f \in F_u, u \in U. \tag{3.10}$$

Given a decision variable

$$x_{uf} = \begin{cases} 1, & \text{if a UAV from group } u \text{ services route } f \in F_u; \\ 0, & \text{otherwise,} \end{cases}$$

and a constant

$$a_{kuf} = \begin{cases} 1, & \text{if a target } k \text{ is visited by a UAV from group } u \text{ in route } f \in F_u; \\ 0, & \text{otherwise.} \end{cases}$$

In the URPRTW, the objective function (3.6) minimizes the expected fuel burn for UAVs to visit all targets. The set-partitioning constraints in set (3.7) restrict all targets to be visited exactly once by one UAV, while constraints in set (3.8) imply that each UAV in a group is assigned to at most one route. The quadratic constraint (3.9) limits the variance of total fuel burn affected by wind variation to a fixed parameter $d \geq 0$. Finally, constraints in set (3.10) represent a binary requirement on decision variable x_{uf}. We can rewrite the quadratic constraint in (3.9) as

$$\sum_{u_1 \in U} \sum_{f_1 \in F_{u_1}} \sum_{u_2 \in U} \sum_{f_2 \in F_{u_2}} cov\left(\tilde{e}_{u_1 f_1}, \tilde{e}_{u_2 f_2}\right) x_{u_1 f_1} x_{u_2 f_2} \leq d. \tag{3.11}$$

Let $q_{u_1 f_1 u_2 f_2}$ denotes the covariance of fuel burn from assigning a UAV from group u_1 to route $f_1 \in F_{u_1}$ and assigning a UAV from group u_2 to route $f_2 \in F_{u_2}$, simultaneously. Because the variance/covariance matrix $Q = [q_{u_1 f_1 u_2 f_2}]$ is symmetric and positive semi-definite, and the quadratic function $x^T Q x$ is convex, using Kelley's cutting plane method [74], we can replace the quadratic constraint in (3.11) by an infinite number of first-order constraints

$$2 \sum_{u_1 \in U} \sum_{f_1 \in F_{u_1}} \sum_{u_2 \in U} \sum_{f_2 \in F_{u_2}} q_{u_1 f_1 u_2 f_2} w_{u_1 f_1} x_{u_2 f_2} \leq d + \sum_{u_1 \in U} \sum_{f_1 \in F_{u_1}} \sum_{u_2 \in U} \sum_{f_2 \in F_{u_2}} q_{u_1 f_1 u_2 f_2} w_{u_1 f_1} w_{u_2 f_2} \quad \forall w \in \Re^{|F|}. \tag{3.12}$$

It follows [74] that the integer program represented by (3.6)-(3.10) and the integer program represented by (3.6)-(3.8),(3.10), and (3.12) are equivalent. In the next section, we discuss variabilities in the URPRTW caused by wind variation.

3.2 Modeling Wind Variation

There are several major contributors for uncertainties in an air transportation, however none is considered to have more influence than the weather. The airline industry depends heavily on the weather in a decision making process. Especially, with oil at its recorded high price, a variation in wind plays a major role in a routing decision. In our URPRTW, which minimizes the expected fuel burn, wind is the only major factor of uncertainty under our consideration. Traveling with a stronger tail wind or a lesser head wind than expected can significantly reduces the amount of expected fuel burn to complete an assigned route. In contrast, a lesser tail wind or a stronger head wind will require more effort, more fuel burn, and thus a higher operating cost to complete an assigned route. To simplify the URPRTW, we made the following assumptions.

Assumption 1. *Wind speed is the only source of uncertainty.*

Assumption 2. *Wind direction is the same for all targets.*

Assumption 3. *Originally planned ground speed is the same for each route.*

Assumption 4. *A route must be followed as planned regardless of actual wind speed.*

In this research, we account for the effect of variations in wind speed and direction when we calculate fuel burn. Let ω_{ij} be the relative difference between the true traveling angle from target i to target j, θ_{ij}, and the wind direction, θ_w, that is $\omega_{ij} = |\theta_{ij} - \theta_w|$. Suppose a set of links, L, is divided into two sets: L_t is the set of links with tailwind, and L_h is the set of links with headwind. When a UAV travels a link $l \in L_t$, $\omega_l < 90$, so the traveling will require a lower airspeed, v_l^a, in order to maintain the scheduled groundspeed, v_l^g, which will result in less fuel burn. In contrast, when a UAV travels a link $l \in L_h$, $90 < \omega_l < 180$, it will require a higher airspeed, v_l^a in order to maintain the scheduled groundspeed, v_l^g, which will result in

more fuel burn. Suppose v^w is the wind speed, the ground speed of traveling from target i to target j with the wind speed adjustment can be computed as

$$v_{ij}^g = v_{ij}^a + v^w cos(\omega_{ij}). \tag{3.13}$$

Therefore, the transit time between targets, t_{ij}, can be simply calculated as $t_{ij} = \frac{\tau_{ij}}{v_{ij}^g}$, where τ_{ij} represents the distance between target i and target j. According to the fuel burn identity, fuel burn is directly related to and can be calculated by a product of airspeed, transit time, and fuel consumption rate. Without loss of generality, any variations of the actual fuel burn from the fuel burn identity, ie. engine size, load size, and wingspan, can be adjusted into the fuel consumption rate. Let ϕ be the fuel consumption rate, ie. pounds per nautical mile traveled in the air, we can compute the total fuel burn used to travel a link $l \in L$ as

$$\tilde{e}_l = v_l^a t_l = \tau_l \phi - \frac{\tau_l \phi v^w cos(\omega_l)}{v_l^g}, \tag{3.14}$$

The first term in (3.14) is the total amount of fuel burn to travel link l, and the second term is either an addition or a deduction of fuel burn accounted for a headwind or a tailwind, respectively. Furthermore, the value of an actual wind speed, v^w, can be adjusted in term of an expected wind speed, $E[v^w]$, and the deviation from the expected amount, $\tilde{v}^w$, that is $v^w = E[v^w] + \tilde{v}^w$. In general, it is assumed that the mean of $\tilde{v}^w$ is 0, that is $E[\tilde{v}^w] = 0$, and the variance of the deviation is the same as the variance of the actual wind, that is $var(\tilde{v}^w) = var(v^w)$. We can rewrite the fuel burn formulation in (3.14) in term of the deviation from its expected amount as

$$\tilde{e}_l = \frac{\tau_l \phi E[v_l^a]}{v_l^g} - \frac{\tau_l \phi cos(\omega_l)}{v_l^g} \tilde{v}^w. \tag{3.15}$$

The first term in (3.15) represents the total amount of fuel burn to travel the link l accounted for the expected airspeed and the expected wind speed, while the second

term represents the amount of fuel burn from the deviation between the actual wind speed and the expected wind speed. Suppose, we let

$$\alpha_l = \frac{\tau_l \phi E[v_l^a]}{v_l^g}, \tag{3.16}$$

$$\beta_l = \frac{\tau_l \phi cos(\omega_l)}{v_l^g}, \tag{3.17}$$

then (3.15) is simplified to

$$\tilde{e}_l = \alpha_l - \beta_l \tilde{v}^w, \tag{3.18}$$

where $\tilde{v}^w$ is an independent random variable represens the fluctuation of wind, α_l is the expected fuel burn of traveling link l, and β_l is a constant rate of how the wind fluctuation affects the total amount of fuel burn $\tilde{e}_l$. The values α_l, and β_l can be adjusted so that $E[\tilde{v}^w] = 0$, $E[\tilde{e}_l] = \alpha_l$, and $var(\tilde{e}_l) = \beta_l^2 var(\tilde{v}^w)$. Recall that $l \in L_t$ implies $\beta_l \geq 0$, and $l \in L_h$ implies $\beta_l \leq 0$. Given that,

$$\alpha_{uf} = \sum_{l \in f} \alpha_l, \tag{3.19}$$

$$\beta_{uf} = \sum_{l \in f} \beta_l, \tag{3.20}$$

for each pair of UAVs from group u_1 and u_2, $(u_1, u_2) \in U \times U$, we can calculate elements of the variance/covariance matrix Q. The covariance of fuel burn for assigning UAV u_1 to route $f_1 \in F_{u_1}$ and assigning UAV u_2 to route $f_2 \in F_{u_2}$, $q_{u_1 f_1 u_2 f_2}$ is the following,

$$\begin{aligned} cov[e_{u_1 f_1}, e_{u_2 f_2}] =& E[(e_{u_1 f_1} - E[e_{u_1 f_1}])(e_{u_2 f_2} - E[e_{u_2 f_2}])] \\ =& E[(\alpha_{u_1 f_1} - \beta_{u_1 f_1} \tilde{v}^w - \alpha_{u_1 f_1})(\alpha_{u_2 f_2} - \beta_{u_2 f_2} \tilde{v}^w - \alpha_{u_2 f_2})] \\ =& E[\beta_{u_1 f_1} \beta_{u_2 f_2}] E[(\tilde{v}^w)^2] \\ =& \beta_{u_1 f_1} \beta_{u_2 f_2} E[(v^w - E[\tilde{v}^w])^2], \end{aligned}$$

hence,

$$q_{u_1 f_1 u_2 f_2} = \beta_{u_1 f_1} \beta_{u_2 f_2} var(\tilde{v}^w). \tag{3.21}$$

As shown above, the covariation between two routes is the product of the summations of the wind deviation effect on fuel burn, β_{uf}, in both routes multiplied by the variation of the wind, $var(\tilde{v}^w)$.

3.3 Quadratic Constraint Simplification

As mentioned earlier, the integer program (3.6)-(3.10) and the integer program (3.6)-(3.8),(3.10), and (3.12) are equivalent in convex optimization. The former simply uses a single quadratic constraint to limit the variance, while the latter uses an infinite number of first-order constraints to duplicate the quadratic constraint. Although they are equivalent, the former is a difficult quadratic integer program, while the latter is a linear integer program and considered to be easier in practice. In our case, we can further simplify the infinite number of first-order constraints in (3.12) to a first-order constraint which most importantly is independent of the constant vector $w \in \Re^{|F|}$.

Lemma 1. *$2w^T Qx \leq d + w^T Qw$, $\forall w \in \Re^{|F|}$ is equivalent to $-\beta^T x \leq \sqrt{\frac{d}{var(\tilde{v}^w)}}$.*

Proof. For a vector $w \in \Re^{|F|} \backslash 0^{|F|}$, let $u = hw$, where a positive constant $h = \sqrt{\frac{d}{w^T Qw}}$. The constraint

$$2u^T Qx \leq d + u^T Qu \tag{3.22}$$

is in the constraint set (3.12) and is a valid inequality. It follows that

$$\begin{aligned}
&2(hw)^T Qx \leq d + (hw)^T Q(hw) \\
&\Rightarrow 2w^T Qx \leq \frac{d + h^2 w^T Qw}{h} \\
&\Rightarrow 2w^T Qx \leq \sqrt{\frac{w^T Qw}{d}} d + \sqrt{\frac{d}{w^T Qw}} w^T Qw \\
&\Rightarrow 2w^T Qx \leq 2\sqrt{dw^T Qw} \\
&\Rightarrow w^T Qx \leq \sqrt{dw^T Qw} \qquad (3.23)
\end{aligned}$$

Given the relationship in equation (3.21), $Q = \beta\beta^T var(\tilde{v}^w)$, where β is an $n \times 1$ vector whose elements are the wind deviation effects on fuel burn, β_{uf}. The constraint set (3.23) can be further simplified from normalizing by a nonnegative term $-w^T \beta var(\tilde{v}^w)$.

$$\Rightarrow \frac{w^T Qx}{-w^T \beta var(\tilde{v}^w)} \leq \frac{\sqrt{dw^T Qw}}{-w^T \beta var(\tilde{v}^w)}$$

$$\begin{aligned}
\Rightarrow \frac{w^T \beta\beta^T var(\tilde{v}^w) x}{-w^T \beta var(\tilde{v}^w)} &\leq \frac{\sqrt{dw^T \beta\beta^T var(\tilde{v}^w) w}}{-w^T \beta var(\tilde{v}^w)} \\
\Rightarrow -\beta^T x &\leq \sqrt{\frac{d}{var(\tilde{v}^w)}}. \qquad (3.24)
\end{aligned}$$

□

The first-order constraints in set (3.23) are tighter than the constraints in set (3.12) and are tangent of the quadratic constraint (3.11). Lemma 1 implies that we can use the constraint (3.24) to represent the constraint sets (3.23) independent of $w \in \Re^{|F|}$. We are now going to prove the claim that $-w^T \beta var(\tilde{v}^w) \geq 0$.

Lemma 2. *All elements of the vector β are nonpositive, $\beta_{uf} \leq 0, \forall u \in U, \forall f \in F_u$.*

Proof. Given a route starts and ends at the depot, the summation of traveling vectors, $\vec{l}$, in the route must be zero,

$$\sum_{l \in f} \vec{l} = 0.$$

Since, the magnitude of traveling vector on link l is $\frac{\tau_l \phi}{E[v^a]}$. We can project the traveling vectors onto the wind path as

$$\sum_{l \in f} \left(\frac{\tau_l \phi}{E\left[v^a\right]} \right) cos(\omega_l) = 0,$$

For the set of links L, we have $cos(\omega_l) > 0$, $\forall l \in L_t$, $cos(\omega_l) < 0$, $\forall l \in L_h$, and $cos(\omega_l) = 0$, $\forall l \in L \backslash \{L_t \cup L_h\}$. It is true that

$$\frac{\tau_l \phi cos(\omega_l)}{E\left[v^a\right] + E\left[v^w\right] cos(\omega_l)} \leq \frac{\tau_l \phi cos(\omega_l)}{E\left[v^a\right]} \qquad ,\forall l \in L.$$

Consequently,

$$\begin{aligned}
\beta_{uf} &= \sum_{l \in f} \beta_l, \\
&= \sum_{l \in f} \frac{\tau_l \phi cos(\omega_l)}{E\left[v^a\right] + E\left[v^w\right] cos(\omega_l)}, \\
&\leq \sum_{l \in f} \frac{\tau_l \phi cos(\omega_l)}{E\left[v^a\right]}, \\
&\leq 0
\end{aligned}$$

□

The proof above shows that $\beta_{uf} \leq 0$, $\forall u \in U$, and $f \in F_u$. Given that $w \geq 0, \forall w \in \Re^{|F|}$ and $var(\tilde{v}^w) \geq 0$, the normalizing term in Lemma 1 is greater or equal to zero, $-w^T \beta var(\tilde{v}^w) \geq 0$.

3.4 Minimum Dependent Set Constraint

We showed in Lemma 1 that an infinite number of first-order constraints in (3.12) could be reduced to a single constraint (3.24) by normalizing with the nonnegative term $-w^T \beta var(\tilde{v}^w)$. Because of the number of constraints, solving the integer program (3.6)-(3.8),(3.10), and (3.24) is much easier than solving the integer program (3.6)-(3.8),(3.10), and (3.12). The constraint (3.24), rewritten in a scalar form

$$-\sum_{u \in U} \sum_{f \in F_u} \beta_{uf} x_{uf} \leq \sqrt{\frac{d}{var(\tilde{v}^w)}}, \tag{3.25}$$

is known as a 0-1 knapsack constraint in integer programming. From the fact that the coefficients of constraint (3.25) are positive, proved by Lemma 2 that $\beta_{uf} \leq 0$, we propose an additional set of first-order constraints called a *Minimum Dependent Set (MDS)* constraint. The MDS constraint can be derived from (3.25), and later shown in this section that it cuts off some fractional solutions $\hat{x}$ that the constraint (3.25) does not. In the following, we derive the MDS constraint. Let S be a polyhedron formed by a 0-1 knapsack constraint,

$$S = \left\{ \sum_{j \in N} a_j x_j \leq b, x \in \{0,1\}^{|n|} \right\}, \tag{3.26}$$

where $N = \{1, 2, ..., n\}$, $a_j, b \in Z_+^1$, and $a_j \leq b$, $\forall j \in N$. We assume that the coefficients are ordered such that $a_1 \geq a_2 \geq \ldots \geq a_n$. Given a characteristic vector $x^C \in \{0,1\}^{|N|}$, a set C is said to be a *dependent* set when characteristic vector $x^C \notin S$ if its components $x_j^C = 1, \forall j \in C$, and $x_j^C = 0, \forall j \notin C$; otherwise C is an independent set. If all subsets of a dependent set C are independent, then set C is said to be a *minimum dependent* set. It follows that given a minimum dependent set C, an MDS constraint $\sum_{j \in C} x_j \leq |C| - 1$ is a valid inequality for S [75]. In our case, a knapsack polyhedron formed by (3.25) is

$$\tilde{S} = \left\{ -\sum_{u \in U} \sum_{f \in F_u} \beta_{uf} x_{uf} \leq \frac{d}{var(\tilde{v}^w)}, x \in \{0,1\}^{|F|} \right\}, \tag{3.27}$$

Let $\mathcal{D}$ be a finite set of all possible minimum dependent sets for the knapsack polyhedron $\tilde{S}$. Then, constraints in set (3.28) are valid inequalities for the polyhedron $\tilde{S}$.

$$\sum_{(u,f) \in C} x_{uf} \leq |C| - 1 \qquad \forall C \in \mathcal{D}, \tag{3.28}$$

We show in Lemma 3 that there exists at least one fractional solution $\hat{x} \in \hat{S}$ violates the valid inequalities in (3.28) where $\hat{S}$ is a continuous relaxation of $\tilde{S}$; that is

$$\hat{S} = \left\{ -\sum_{u \in U} \sum_{f \in F_u} \beta_{uf} x_{uf} \leq \frac{d}{var(\tilde{v}^w)}, 0 \leq x \leq 1 \right\}. \tag{3.29}$$

Lemma 3. *Consider a minimum dependent set C, if $\exists \hat{x} \in \{0,1\}^{|F|}$ such that*

$$\sum_{j \in C} \hat{x}_j = |C| - 1, \text{ and} \tag{3.30}$$

$$\sum_{u \in U} \sum_{f \in F_u} \beta_{uf} \hat{x}_{uf} < \sqrt{\frac{d}{var(\tilde{v}^w)}}, \tag{3.31}$$

then $\hat{x} \in \hat{S}$ violates (3.28).

Proof. Consider $\hat{x}$ where

$$\hat{x}_j = \begin{cases} 1, & \text{if } j \in C \setminus \left\{\hat{j}\right\}; \\ \frac{\sqrt{\frac{d}{var(\tilde{v}^w)}} - \sum_{j \in C \setminus \{\hat{j}\}} \beta_j}{\beta_j}, & \text{if } j = \hat{j}\ ; \\ 0, & \text{otherwise.} \end{cases}$$

It is obvious that,

$$\sum_{j \in C} \hat{x}_j = |C| - 1 + \frac{\sqrt{\frac{d}{var(\tilde{v}^w)}} - \sum_{j \in C \setminus \{\hat{j}\}} \beta_j}{\beta_j} \qquad (> |C| - 1)$$

Hence $\hat{x} \in \hat{S}$ but violates (3.28). □

Observe that by the definition of a minimum dependent set there is always a binary solution that satisfies (3.30). From Lemma 3, it can be seen that the MDS constraints (3.28) cuts of the solution $\hat{x}$ while the constraint (3.25) does not. Observe that the coefficients of (3.25) are fractional, but the coefficients of (3.28) are integer, consequently the constraints in set (3.28) should encourage integrality in the solution. In the next section, we discuss the BCP methodology used to solve the URPRTW.

3.5 Branch-and-Cut-and-Price Methodology

In this section, we discuss the Branch-and-Cut-and-Price (BCP) methodology. The BCP methodology is an LP-based branch and bound algorithm for solving mixed integer linear problems. The mixed-integer linear problem is solved within a branch-and-bound tree to ensure integer solutions. Within each node in a branch-and-bound

tree, a continuous relaxation of the interested mixed-integer problem is solved, bounds are updated, and branching is performed. In this research, we implement the *Follow-on* branching as our branching logic. The Follow-on branching is a variant of *Ryan-Foster* branching [76] that fixes or deletes certain edges that represent connections between two consecutive targets. The Follow-on branching is very suitable to our column generation subproblem and will be discussed later.

3.5.1 Delayed Column Generation Algorithm (DCG)

As mentioned previously, the continuous relaxation problem is solved within each node of a branch-and-bound tree. In this section, we discuss the Delayed Column Generation Algorithm (DCG) as an algorithm to solve the continuous relaxation of URPRTW (CURPRTW) in a node of a branch-and-bound tree. The CURPRTW is as follows.

$$\min \sum_{u \in U} \sum_{f \in F_u} E\left[\tilde{e}_{uf}\right] x_{uf} \tag{3.32}$$

$$\text{s.t.} \sum_{u \in U} \sum_{f \in F_u} a_{kuf} x_{uf} = 1 \qquad \forall k \in K, \tag{3.33}$$

$$\sum_{f \in F_u} x_{uf} \leq n_u \qquad \forall u \in U, \tag{3.34}$$

$$-\sum_{u \in U} \sum_{f \in F_u} \beta_{uf} x_{uf} \leq \sqrt{\frac{d}{var(\tilde{v}^w)}} \tag{3.35}$$

$$\sum_{u \in U} \sum_{f \in F_u} a_{Cuf} x_{uf} \leq |C| - 1 \qquad \forall C \in \mathcal{D}, \tag{3.36}$$

$$0 \leq x_{uf} \leq 1 \qquad \forall u \in U, f \in F_u, \tag{3.37}$$

where

$$a_{Cuf} = \begin{cases} 1, & \text{if a variable } x_{uf} \in C; \\ 0, & \text{otherwise.} \end{cases}$$

Let π_k, π_u, ρ_w, and ρ_C be dual variables for constraints in set (3.33), (3.34), (3.35), and (3.36), respectively. Therefore, given the optimal solution $(x^*, \pi_k^*, \pi_u^*, \rho_w^*, \rho_C^*)$, the reduced cost $\bar{c}_{uf}$ for variable x_{uf} is

$$\bar{c}_{uf} = E\left[\tilde{e}_{uf}\right] - \sum_{k \in K} a_{kuf}\pi_k^* - \pi_u^* + \beta_{uf}\rho_w^* - \sum_{C \in \mathcal{D}} a_{Cuf}\rho_C^* \qquad \forall u \in U, \forall f \in F_u. \quad (3.38)$$

Given $E\left[\tilde{e}_{uf}\right] = \alpha_{uf}$, and relationships in (3.19) and (3.20), we can simplify the reduced cost for a variable x_{uf} in (3.38) to a *link based* reduced cost as

$$\bar{c}_{uf} = \sum_{l \in f}\left\{\alpha_l + \beta_l\rho_w^*\right\} - \sum_{k \in K}\pi_k^* - \pi_u^* - \sum_{C \in \mathcal{D}} a_{Cuf}\rho_C^*, \qquad \forall u \in U, \forall f \in F_u. \quad (3.39)$$

A simplified reduced cost in (3.39) implies that a cost of link l in a graph G is $\alpha_l + \beta_l\rho_w^* - \pi_k^*$, a cost of using a UAV u is $-\pi_u^*$, and a cost of existing in a minimum dependent set $C \in \mathcal{D}$ is ρ_C^*. Let a variable RC_{uf} be the total cost of using all links for a variable x_{uf}, and a variable ρ_{uf} be the total cost of existing in MDS constraints for a variable x_{uf}, that is

$$RC_{uf} = \sum_{l \in f}\left\{\alpha_l + \beta_l\rho_w^*\right\} a_{luf} - \sum_{k \in K}\pi_k^* - \pi_u^* \quad (3.40)$$

$$\rho_{uf} = \sum_{C \in \mathcal{D}} a_{Cuf}\rho_C^*. \quad (3.41)$$

The reduced cost $\bar{c}_{uf}$ in (3.39) can be rewritten as

$$\bar{c}_{uf} = RC_{uf} - \rho_{uf}. \quad (3.42)$$

We will discuss more the reduced cost and column generation in Section 3.5.2. In the following, we present the DCG algorithm that is used to solve the CURPRTW in a node of a branch-and-bound tree. In the DCG algorithm, shown in Algorithm 1, the RMP only includes a subset of all routes $\overline{F} \subset F$ and generate new routes only when they are needed. The CURPRTW is then solved to obtain an optimal solution x^*. In the cut generation step, the algorithm attempts to generate an MDS constraint $C \in \mathcal{D}$ to cut off the solution x^*, if x^* is fractional. However, if the MDS constraint

does not exist, the column generation step is invoked where a new variable with a negative reduced cost is added to the RMP. The steps are repeated until there is no variable to be added. We discuss the column generation step and the cut generation step in Section 3.5.2, and Section 3.5.3, respectively.

Algorithm 1 Delayed Column Generation Algorithm (DCG)

Initialization Step: Let $\mathcal{M} \leftarrow \emptyset$ be a subset of all possible minimum dependent sets for a polyhedron $\tilde{S}$ defined in (3.27), $\mathcal{M} \subseteq \mathcal{D}$. Generate a subset of routes $\overline{F}_u \subset F_u, \forall u \in U$.

Restricted Master Problem (RMP) Step: Solve CURLFV over the set of subsets $\overline{F} = \bigcup_{u \in U} \overline{F}_u$, and $\mathcal{M}$ to get a solution (x^*, π^*, ρ^*).

Cut Generation Step: Let $\overline{C}$ be a minimum dependent set for x^*.

if $\overline{C} \neq \emptyset$ **then**

Update the set $\mathcal{M} \leftarrow \mathcal{M} \cup \overline{C}$, and return to the RMP step.

else

Column Generation Step: Find a route $\overline{f} \in F_u \setminus \overline{F}_u$ that minimizes the reduced cost $\bar{c}_{u\overline{f}}$ from (3.39).

if $\bar{c}_{u\overline{f}} < 0$ **then**

Update the set of subsets $\overline{F} \leftarrow \overline{F} \cup \{\overline{f}\}$, and return to the RMP step.

else

Return the optimal solution (x^*, π^*, ρ^*).

end if

end if

3.5.2 Column Generation

In this section, we discuss a procedure used to generate new variables with negative reduced cost for the column generation step of the DCG algorithm. There

are two algorithms presented in this section. Both find a minimum cost path in an instance graph G, but there are distinctions and complications when they are implemented. The minimum cost path is a route that starts and ends at a depot with a minimum cost. An instance of a minimum cost path problem for the DCG column generation step is obtained by constructing an instance graph $G = (V, E)$ where V is a set of nodes and E is a set of directed edges. In our URPRTW problem, with n targets, the set of nodes $V = \{v_0, v_1, v_2, ..., v_n, v_{n+1}\}$, where node v_0, and node v_{n+1} are the source node and the terminal node, respectively. Although v_0 and v_{n+1} are two different nodes in the graph G, they represent a common depot that a UAV route starts and ends at. Without loss of generality, we assume that all target nodes in set V, $\forall v_i \in V \setminus \{v_0, v_{n+1}\}$, are sorted with respect to their starting service time, that is $o_{v_1} \leq o_{v_2} \leq \ldots \leq o_{v_n}$, where o_{v_i} denotes the starting service time of target i. An edge e_{ij} represents a link that connects between target i and target j, and the associated cost for the edge is c_{ij}. We discussed the cost of the edge from target i to target j in the previous section that is $\alpha_l + \beta_l \rho_w^* - \pi_k^*$. The c_{ij} is not restricted in sign, meaning the cost $\alpha_l + \beta_l \rho_w^* - \pi_k^*$ can be either positive or negative. This may lead to negative cost cycles. However traveling from target i to target j is unreachable when $i > j$ since it represents traveling back in time, hence $c_{ij} = \infty$ when $i > j$. This is an advantage for solving minimum cost path problem in the URPRTW since the unreachable edges eliminate the negative cost cycles in the minimum cost path solution. The advantage however does not apply to the URPR which is discussed in chapter 5. We can graphically show an instance of the minimum cost route problem for a URPRTW with 3 targets as in Figure 3.1.

To generate a new variable to be added to the RMP, we find the minimum cost path that starts at node v_0 and ends at node v_{n+1} with a negative reduced cost. We propose two algorithms to solve the minimum cost path problem represented by an instance graph G, the first is *Dynamic Programming Shortest Path* algorithm

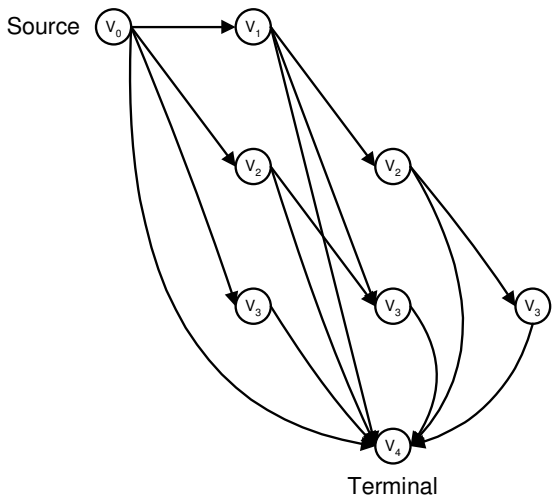

Figure 3.1. An instance graph of the URPRTW.

(DPSP) and the second is the *Integer Programming Shortest Path* algorithm (IPSP). Both methods are well known in the network literature, however the DPSP algorithm is superior over the IPSP algorithm on computational complexity and time. In the following, we discuss each method in details along with complications that may arise when implemented as a column generation engine in the DCG algorithm.

The DPSP algorithm is a shortest path algorithm that uses a dynamic programming fundamental called *divide and conquer* technique. The DPSP algorithm divides a problem into smaller subproblems. The smaller subproblems are solved recursively. The optimal solution of the problem can be constructed by the optimal solutions of the subproblems. In our case, the instance graph G is broken down to construct new subgraphs. The DPSP algorithm then identifies the shortest path within these subgraphs. Because these subgraphs are overlapping, the shortest path of the instance graph G can be constructed from the shortest paths of the subgraphs.

To implement the DPSP algorithm, let a recursive function $g(v_i)$ defines the minimum cost from node v_i to the terminal node v_{n+1} and a set $\delta^+_{v_i} = \{j : (i,j) \in E\}$

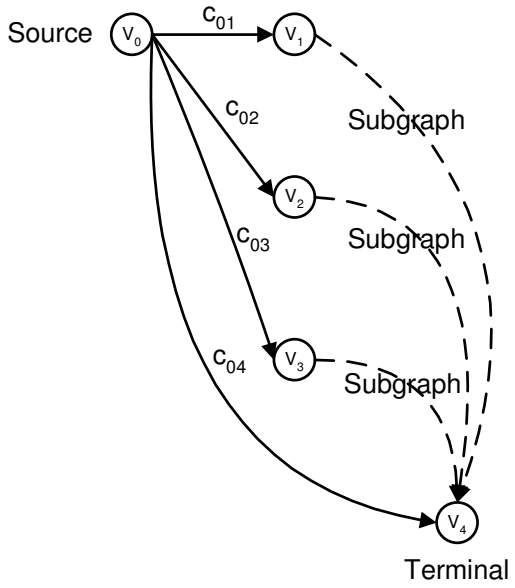

Figure 3.2. A DPSP algorithm.

be a set of all departing edges from node v_i in graph G. The recursive function is defined as

$$g(v_i) = \min_{(i,j)\in\delta^+_{v_i}} \{c_{ij} + g(v_j)\}. \tag{3.43}$$

To find the minimum cost of the instance graph G, one only need to evaluate the recursive function at the source node, that is $g(v_0)$. Although DPSP determines only the minimum cost of the instance, it is easy to record the path that has the minimum cost. Given the route f^* is the minimum cost path from the DPSP algorithm, the minimum cost of route f^* is the summation of costs of all links in f^*, that is $\sum_{l\in f^*} \{\alpha_l + \beta_l \rho^*_w - \pi^*_k\}$. We add the cost of using a UAV u, $-\pi^*_u$, to get the value in (3.40). To receive the reduced cost of the minimum cost route f^*, $\bar{c}_{uf^*}$, one must further subtract the cost of MDS constraints in (3.41). The route f^* will be added to the RMP only if the reduced cost $\bar{c}_{uf^*} < 0$.

A complication may arise when implementing the DPSP algorithm along with the cut generation step in the DCG algorithm where MDS cuts are generated. Consider the following example, let route f_1 and route f_2 be routes with negative costs

in an instance graph G. Let the cost of route f_1 be η_{f_1}, that is $RC_{vf_1} = \eta_{f_1}$, and let the cost of route f_2 be η_{f_2}, that is $RC_{vf_2} = \eta_{f_2}$, where $\eta_{f_1}, \eta_{f_2} < 0$. Given f_1 is the minimum cost route in an instance graph G, hence $\eta_{f_1} < \eta_{f_2}$. The complication arises when route f_1 was previously generated and added to the RMP and $a_{Cuf} = 1, \exists C \in \mathcal{D}$, hence $\rho_{uf_1} = \eta_{f_1}$. If the route f_2 has not been added to the RMP, hence $a_{Cuf} = 0, \forall C \in \mathcal{D}$. Therefore, the reduced costs for both routes are

$$\bar{c}_{uf_1} = RC_{uf_1} - \rho_{uf_1} = \eta_{f_1} - \eta_{f_1} = 0, \text{ and}$$
$$\bar{c}_{uf_2} = RC_{uf_2} - \rho_{uf_2} = \eta_{f_2} - 0 = \eta_{f_2} \qquad (< 0),$$

The DPSP algorithm returns route f_1 as the minimum cost route, since $\eta_{f_1} < \eta_{f_2}$. However, the column generation step of the DCG algorithm assumes that there are no more routes with negative reduced cost since DPSP returns route f_1 with nonnegative reduced cost, $\bar{c}_{uf_1} = 0$. The column generation in the DCG algorithm will stop and then proceed to branch for an integer solution. Instead, the DPSP algorithm should return route f_2 to be added to the RMP it has a negative reduced cost, that is $\bar{c}_{uf_2} = -\eta_{f_2}$. This situation only occurs when a previously generated route has a minimum cost in the graph and also exists in MDS constraints that have corresponding dual variables $\rho^*_C < 0$. Since we cannot embed the cost of existing in MDS constraints, ρ^*_C, in an instance graph G, implementing the DPSP algorithm in the DCG column generation step while implementing MDS constraints in the DCG cut generation step may lead to a suboptimal column generation and hence render a suboptimal solution. In order to overcome this difficulty, we need to consider a *constrained shortest path algorithm* which is much more difficult, and indeed the DPSP algorithm is not applicable. We can implement the DPSP algorithm in the column generation step only when the DCG algorithm does not include the cut generation step.

We propose another algorithm to find the *constrained minimum cost route* in an instance graph G when the DCG algorithm includes the cut generation step. The algorithm is *Integer Programming Shortest Path Algorithm (IPSP)*. In the IPSP

algorithm, we construct a new integer linear program called a *Minimum-Cost Network Flow* problem (MCNF) to find the minimum cost route in an instance graph G. The MCNF problem is then solved using an available mixed-integer solver software to obtain a minimum cost route. If the minimum cost route was already added to the RMP, a cut is generated and added to the MCNF problem to cut off the previously generated route. The MCNF problem is as follows.

$$\min \sum_{l \in L} \{\alpha_l + \beta_l \rho_w^* - \pi_k^*\} y_l \tag{3.44}$$

$$\text{s.t.} \sum_{l \in \delta_{v_i}^+} y_l - \sum_{l \in \delta_{v_i}^-} y_l = b(v_i) \qquad \forall v_i \in V, \tag{3.45}$$

$$y_l \in \{0,1\} \qquad \forall l \in L, \tag{3.46}$$

where $\delta_{v_i}^+ = \{j : (i,j) \in E\}$, $\delta_{v_i}^- = \{j : (j,i) \in E\}$. The constraints in set (3.45) are node conservation constraints, where $b(v_0) = 1$, $b(v_{n+1}) = -1$, and $b(v_i) = 0, \forall v_i \in V \backslash \{v_1, v_{n+1}\}$. If a solution y^* obtained from solving the MCNF problem is in fact a route $\tilde{f}$ that was previously added to the RMP, where $\tilde{f} \in \overline{C}$ and $\rho_{\overline{C}}^* < 0$, $\exists \overline{C} \in \mathcal{D}$, we would add the following *route elimination* constraint,

$$\sum_{l \in \tilde{f}} y_l \leq |\Omega(\tilde{f})|, \tag{3.47}$$

where $\Omega(\tilde{f})$ is a set of targets visited in route $\tilde{f}$. We show the IPSP algorithm in Algorithm 2.

When implemented, the route elimination constraints in (3.47) can be added to the MCNF problem a *priori* since there are only a finite number of routes that were added into the RMP and also exist in the MDS constraints with nonzero dual variables.

Algorithm 2 Integer Programming Shortest Path Algorithm (IPSP)

Initialization Step: Let $\mathcal{W} \subset \Re^{|E|}$ be a finite set.

Restricted Master Problem (RMP) Step: Solve the MCNF with $\mathcal{W}$ to obtain y^*.

if $y^* \in \overline{C}$ and $\rho^*_{\overline{C}} < 0$, $\exists \overline{C} \in \mathcal{D}$ **then**

Cut Generation Step: Add a route elimination constraint (3.47), and update $\mathcal{W} \leftarrow \mathcal{W} \cup \{y^*\}$ and return to the RMP Step.

else

Return the optimal solution y^*.

end if

3.5.3 Cut Generation

In the DCG algorithm, after solving the CURPRTW problem for a solution x^*, the DCG cut generation is invoked. We generate MDS constraints as in (3.36) to be added to the RMP. Usually x^* is a fractional solution, however the coefficients of generating MDS constraints are integer, which should encourage integrality in a solution. In each iteration, we attempt to find an MDS constraint that cuts off the fractional solution x^*. Consider the following integer programming problem.

$$\min \sum_{u \in U} \sum_{f \in F_u} (1 - x^*_{uf}) \tilde{x}_{uf} \tag{3.48}$$

$$\text{s.t.} \sum_{u \in U} \sum_{f \in F_u} \beta_{uf} \tilde{x}_{uf} \geq \sqrt{\frac{d}{var(\tilde{v}^w)}} + \varepsilon, \tag{3.49}$$

$$\tilde{x}_{uf} \in \{0, 1\}, \tag{3.50}$$

A feasible solution to the above integer program in which the objective value is less than ε implies that there exists an MDS constraint that cuts off the current fractional solution x^*. We then generate an MDS constraint as in (3.36) and add to the RMP. However, short of solving the integer program (3.48)-(3.50) using an integer

programming solver, we propose an *MDS generation heuristic*. The heuristic starts by sorting variable x_{uf} with respect to the term $\frac{1-x^*_{uf}}{\beta_{uf}}$ in an ascending order with higher β_{uf} to break ties. Let x^i_{uf} be the i^{th} order of the sorting sequence $i = 1, 2, ...N$, where N is the total number of variables in the RMP. We greedily add variable x^i_{uf} into a minimum dependent set C starting from $i = 1$ until finding a minimum dependent set C. The MDS generation heuristic is shown in Algorithm 3. In the next section, we discuss the computational experiments done on the URPRTW.

Algorithm 3 MDS Generation Heuristic

Initialization Step: Let a set $C \leftarrow \emptyset$ be a minimum dependent set, and $i = 1$.

Selection Step: Add a variable x^i_{uf} into a minimum dependent set, $C \leftarrow C \cup \{x^i_{uf}\}$, and $i \leftarrow i + 1$.

Quality Check Step:

if $i \leq N$ **then**

if $\sum_{u \in U} \sum_{f \in F_u} a_{Cuf} \beta_{uf} x^*_{uf} > \sqrt{\frac{d}{var(\tilde{v}^w)}}$ and $\sum_{u \in U} \sum_{f \in F_u} a_{Cuf} x^*_{uf} > |C| - 1$ **then**

Return the minimum dependent set C.

else

Return to the Selection Step.

end if

else

Return $\emptyset$.

end if

3.6 Computational Experiments

In this section, we present the computational experiments of the URPRTW. We implemented the DCG algorithm using the Computational Infrastructure for Operation Research (COIN-OR) that accommodates the BCP methodology. We used CPLEX 9.120 as a solver to solve the CURPRTW in each node of a branch-and-bound tree and to solve the MCNF problem in the DCG column generation step. The experiments were conducted on a Dual 3.06-Ghz Intel Xeon workstation. In our computational experiments, we implemented the DCG algorithm on the URPRTW. We created the URPRTW instances with the combination of 1 UAV group and 30, 60, and 90 targets. Locations, time windows, wind speed, wind direction, and wind variation were randomly generated. Each problem instance was solved with three different algorithms, which are the DCG-DPSP algorithm, the DCG-IPSP algorithm, and the DCG-IPSP-MDS algorithm. The DCG-DPSP algorithm uses the DPSP algorithm in the column generation step. We cannot include the cut generation step in the DCG-DPSP algorithm since it may result in a suboptimal column generation step. The DCG-IPSP algorithm uses the IPSP algorithm in the column generation step. The main purpose of implementing the DCG-IPSP is to compare the efficiencies of the DPSP algorithm and the IPSP algorithm as the column generation engine. Finally, the DCG-IPSP-MDS uses the IPSP algorithm in the column generation step, and the cut generation step generates MDS constraints. We discuss effects of generating MDS constraints in the cut generation step later in this section.

First, we solved each URPRTW instance without the variance constraint in (3.35) to receive an optimal solution. This case is referred to as the *deterministic* case and is shown in solution tables as a case with $d = 100\%$. The variance of the optimal solution from the deterministic case is recorded and used as a reference level for other values of d. We considered six levels of parameter d, which are 95%, 90%, 85%, 80%, 75%, and 70% of the standard deviation of the deterministic case. Note

that solving the URPRTW instances with different algorithms provided the same optimal solution, which are expected fuel burn, optimal routes, and variances. Table 3.1 shows the computational results of the URPRTW instances. The columns in Table 3.1 include the seven levels of the standard deviations with respect to that of the deterministic case, *d*, the actual value of the limited standard deviation, *MAX SD*, the value of standard deviation of the best found solution, *SD BS*, the different in percentage between the standard deviation of the best found solution and the standard deviation of the deterministic case, *SD (%)*, the expected fuel burn of the best found solution, *BS*, and the percentage of the expected fuel burn of the best found solution with respect to that of the deterministic case, *BS (%)*. It must be noted that all URPRTW instances were optimally solved before the 10 hours time limit is reached, hence received the same optimal solution from all three algorithms.

Table 3.1. Computation results of the URPRTW.

1 group, 30 targets					
d	MAX SD	SD BS	SD(%)	BS	BS(%)
70%	38.858	38.681	30.32	21128.362	125.54
75%	41.634	41.568	25.12	19197.895	114.07
80%	44.409	44.114	20.53	18587.199	110.44
85%	47.185	46.733	15.81	17641.524	104.82
90%	49.960	49.318	11.16	17504.803	104.01
95%	52.736	52.734	5.00	16861.065	100.18
100%	55.511	55.511	0.00	16830.159	100.00
1 group, 60 targets					
d	MAX SD	SD BS	SD(%)	BS	BS(%)
70%	65.761	65.640	30.13	32202.525	111.40
75%	70.459	70.429	25.03	31028.023	107.34
80%	75.156	75.027	20.14	29997.313	103.77
85%	79.853	79.499	15.38	29376.266	101.63
90%	84.550	84.510	10.04	29098.619	100.67
95%	89.248	88.775	5.50	28999.942	100.32
100%	93.945	93.945	0.00	28906.284	100.00
1 group, 90 targets					
d	MAX SD	SD BS	SD(%)	BS	BS(%)
70%	85.524	85.499	30.02	39806.898	109.30
75%	91.633	91.521	25.09	38634.260	106.08
80%	97.742	97.588	20.13	37823.394	103.86
85%	103.851	103.800	15.04	37082.975	101.82
90%	109.960	109.826	10.11	36736.543	100.87
95%	116.069	115.733	5.28	36454.951	100.10
100%	122.178	122.178	0.00	36418.983	100.00

It is intuitive that the reduction of the variation in the solution must be exchanged with the increase in the expected fuel burn. From Table 3.1, in the 1 UAV group and 30 targets instance, reducing the standard deviation by 5%, 11.16%, 15.81%, 20.53%, 25.12%, and 30.32% are accompanied with increasing of the expected fuel burn by

0.18%, 4.01%, 4.82%, 10.44%, 14.07%, and 25.54%, respectively. The variation in the deterministic case, which is 3081 ($= 55.551^2$), can be reduced to 2781 ($= 52.734^2$) with only a marginal increase in the expected fuel burn, however the ratio of the marginal benefit (reduction of variance) and the marginal cost (increase of expected fuel burn) changes when the reduction of the variation increases. The 60-target instance and the 90-target instance show similar results, and that we can reduce the variation by more than 30% while the expected fuel burn only increases about 10%. Figure 3.3 shows the relationship between the marginal benefit and the marginal cost. The efficient frontier depicted in the figure can help logistic planners to choose the best variation and expected fuel burn combination that is suitable for their risk preferences.

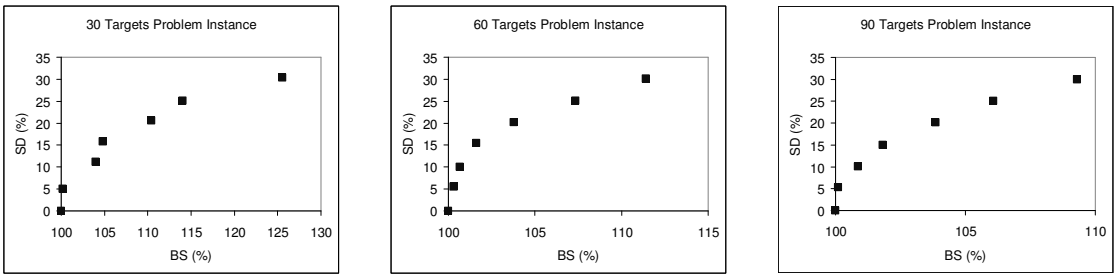

Figure 3.3. Efficient frontiers for 30, 60, and 90 targets problem instances.

Table 3.2 presents other characteristics of using the DCG-DPSP algorithm to solve the URPRTW. The columns in Table 3.2 include the seven levels of standard deviation with respect to the deterministic case, d, the time in seconds to find the best solution, *CPU BS*, the total number of variables generated by the DPSP algorithm in the column generation step, *VAR BS*, the time in seconds when the DCG algorithm terminates, *CPU*, and the total number of generated variables, *VARS*. It is obvious that the DCG-DPSP algorithm is very efficient for the URPRTW. The CPU time is very short. Especially in the 1 UAV group and 30 targets instance, when *CPU BS* for all cases are less than one second. It is expected that for cases with higher number of targets, the *CPU BS* increases. The DPSP algorithm is very suitable with our

shortest path problem represented in an instance graph G. The dynamic programming technique makes use of substructures which are subgraphs in the instance graph G, solves the subgraphs recursively, and then uses these optimal solutions of the subgraphs to construct the optimal solution which is a minimum cost route of the graph G. When the level of variation decreases, the total number of variables until the best solution found increases, since the DPSP algorithm must find more routes that generate less variation to the optimal solution.

Table 3.2. Computation results of the DCG-DPSP algorithm

1 group, 30 targets				
d	CPU BS	VARS BS	CPU	VARS
70%	0.33	311	0.86	870
75%	0.25	268	0.55	608
80%	0.26	287	0.74	770
85%	0.18	197	0.42	456
90%	0.25	318	0.26	348
95%	0.15	165	0.98	1045
100%	0.16	182	0.17	192
1 group, 60 targets				
d	CPU BS	VARS BS	CPU	VARS
70%	11.28	3960	33.43	11806
75%	7.13	2705	11.62	4006
80%	2.32	836	7.15	2807
85%	1.83	665	3.04	1182
90%	2.44	893	2.93	1068
95%	2.59	966	2.94	1014
100%	1.43	559	1.43	559
1 group, 90 targets				
d	CPU BS	VARS BS	CPU	VARS
70%	25.19	4299	139.66	28364
75%	52.69	9779	85.26	14187
80%	55.99	7553	148.31	22020
85%	25.68	3941	41.57	5404
90%	10.56	1529	13.73	2284
95%	8.99	1254	12.57	2046
100%	8.19	1182	8.26	1197

We can compare the efficiency of the DPSP algorithm and the IPSP algorithm as the column generation engine from the numbers in Table 3.2 and Table 3.3. The format of Table 3.3 is the same as in Table 3.2 where columns are *d*, *CPU BS*, *VARS BS*, *CPU*, and *VARS*, respectively. Generally, the numbers in Table 3.2 and Table 3.3 indicate that the DCG-DPSP algorithm performed better than the DCG-IPSP algorithm in computational time. The DCG-IPSP algorithm took longer time to find the optimal solution. This results from the fact that the DCG-DPSP algorithm is very suitable to our column generation subproblem. Also, when the IPSP algorithm is used as the column generation engine, a new integer programming problem is con-

structed to represent the MCNF problem every time the column generation step is invoked. This requires a lot of computational efforts in CPLEX, and this makes the IPSP algorithm even more time consuming. Although the total number of variables generated should be the same between the two algorithms, they are different. The two column generation algorithms break ties among the minimum cost routes differently. This is a reason why the numbers in column *VARS BS*, and *VARS* are not the same between Table 3.2 and Table 3.3. Nevertheless, the difference is insignificant. Obviously, the DCG-DPSP algorithm is a better algorithm than the DCG-IPSP algorithm for the URPRTW. However, as mentioned before, the DCG-DPSP algorithm cannot be implemented along with the cut generation step that generates MDS constraints.

Table 3.3. Computation results of the DCG-IPSP algorithm

1 group, 30 targets				
d	CPU BS	VARS BS	CPU	VARS
70%	1.90	293	5.70	868
75%	1.58	258	4.06	665
80%	4.85	708	5.57	803
85%	4.69	710	5.49	811
90%	1.90	327	2.11	362
95%	0.96	165	5.46	874
100%	1.05	184	1.12	196
1 group, 60 targets				
d	CPU BS	VARS BS	CPU	VARS
70%	137.77	4342	381.75	12277
75%	64.31	2024	93.44	2925
80%	21.29	717	74.57	2493
85%	19.00	654	38.32	1304
90%	20.36	655	29.73	951
95%	30.56	1038	32.21	1085
100%	16.03	555	16.09	556
1 group, 90 targets				
d	CPU BS	VARS BS	CPU	VARS
70%	410.58	4489	2980.6	33449
75%	943.77	10260	1271.60	13769
80%	413.11	4135	1442.50	14758
85%	459.16	4910	665.67	7094
90%	153.45	1592	207.88	2157
95%	110.14	1223	187.50	2081
100%	113.72	1182	116.85	1214

We implemented the DCG-IPSP-MDS algorithm to examine the effectiveness of generating MDS constraints in the cut generation step. For the previously discussed two algorithms, the DCG-DPSP algorithm, and the DCG-IPSP algorithm, the cut generation step was not included. Although the limited values of variance were specified but the simplified variance constraint (3.35) was added to the RMP *a priori*, the cut generation step is unnecessary for the DCG-DPSP algorithm, and

the DCG-IPSP algorithm. When the DCG-IPSP-MDS algorithm is implemented, the route elimination constraints in (3.47) are added to the MCNF problem in order to maintain optimality of the column generation step. In our implementation, we added the route elimination constraints in (3.47) for all routes that were in the MDS constraints with nonzero dual variables *a priori* to maintain the DCG column generation optimality. Therefore it makes no sense to report the total number of route elimination constraints in the results. We present the DCG-IPSP-MDS results in Table 3.4. The columns in Table 3.4 are the same labels as in Table 3.2 and Table 3.3 have the same meaning. However the additional columns labeled *CUTS BS*, and *CUTS* represent the total number of MDS constraints added before finding the best solution, and the total number of MDS constraints added, respectively.

Table 3.4. Computation results of the DCG-DPSP-MDS algorithm

1 group, 30 targets						
d	CPU BS	VARS BS	CUTS BS	CPU	VARS	CUTS
70%	1.64	262	1	6.04	924	9
75%	1.45	233	1	4.69	712	7
80%	6.13	819	5	6.92	895	11
85%	3.00	436	2	3.18	459	5
90%	2.03	327	0	2.26	362	0
95%	1.02	165	0	1.59	237	3
100%	1.13	184	0	1.20	196	0
1 group, 60 targets						
d	CPU BS	VARS BS	CUTS BS	CPU	VARS	CUTS
70%	127.68	3455	4	379.72	11359	45
75%	86.66	2305	5	127.51	3453	19
80%	22.31	717	0	81.82	2570	3
85%	19.84	654	0	40.96	1304	0
90%	21.40	655	0	33.87	1018	1
95%	31.80	959	1	33.74	1006	2
100%	16.83	555	0	16.89	556	0
1 group, 90 targets						
d	CPU BS	VARS BS	CUTS BS	CPU	VARS	CUTS
70%	412.46	3730	3	5167.48	54764	81
75%	879.06	8368	13	1317.27	12495	33
80%	451.24	3653	4	2538.03	25890	71
85%	329.17	2873	3	597.89	5298	19
90%	163.56	1592	0	223.92	2157	0
95%	123.92	1223	0	190.89	1913	2
100%	109.15	1182	0	112.42	1214	0

The optimal solutions from all three algorithms are the same and are shown in table 3.1. We want to point out that the CPU times of the DCG-IPSP-MDS algorithm are very competitive with those of the DCG-IPSP algorithm. In some cases, the *CPU BS*, and *CPU* of the DCG-IPSP-MDS algorithm are even lower than those of the DCG-IPSP algorithm. There were some cases that the MDS constraints could not

be found, however the problem instances in which MDS constraints were added show lower or competitive *CPU BS*, and *CPU*. Moreover, in most cases, the total number of the variables generated before finding the best solution in the DCG-IPSP-MDS algorithm is less or competitive with the other two algorithms. We conclude that the MDS constraints have a positive effect on reaching an integer solution faster since the structure of the MDS constraint encourages integrality in the solution. The MDS constraint should be more attractive when implemented with the URPR, a problem which is very difficult and generating a variable is a very time-consuming process. We discuss the URPR, the UAV routing problem without time windows in Chapter 5. In the next chapter, we will discuss a special case of the SPQC where a variation of a solution is different from that discussed in this chapter and hence the quadratic constraint is irreducible.

CHAPTER 4

SET-PARTITIONING PROBLEM WITH A MODIFIED QUADRATIC CONSTRAINT

4.1 Introduction

In this chapter, we discuss a special case of the SPQC with a different variation model (SPQCV). The SPQCV is a revised version of the SPQC defined by (3.1)-(3.4).

$$\min c^T x \tag{4.1}$$

$$\text{s.t. } Ax = 1, \tag{4.2}$$

$$x^T Q x + b^T x \leq d, \tag{4.3}$$

$$x \in \{0,1\}^n. \tag{4.4}$$

Let N be a set of variables in the SPQCV, and let M be a set of set-partitioning constraints, hence A is a constant 0-1 $|M| \times |N|$ matrix, c is a constant $|N| \times 1$ vector, 1 is an $|M| \times 1$ vector of ones, and x is a 0-1 $|N| \times 1$ vector. Q is an $|N| \times |N|$ symmetric positive semidefinite matrix in which its entry $[q_{ij}]$, where $i \in M, j \in N$, is the covariance of selecting variables x_i and x_j, simultaneously, and b is an $|N| \times 1$ vector in which its entry $[b_j]$, where $j \in N$ is a variance from selecting a variable x_j. Since the quadratic function $x^T Q x + bx$ is convex, by Kelley's cutting plane method [74], we can replace the quadratic constraint in (4.3) by an infinite set of first-order constraints given by

$$2w^T Q x + bx \leq d + w^T Q w \qquad \forall w \in \Re^{|N|}, \tag{4.5}$$

where w is a vector of constants of dimension $|N|$. The formulation represented by (4.1)-(4.4), and the one represented by (4.1),(4.2), (4.4), and (4.5) are known to be

equivalent in convex programming. Moreover, in an integer solution, x_j is binary, so $x_j^2 = x_j$. Hence, an alternative formulation replaces the quadratic constraint (4.3) with

$$x^T [Q + B] x \leq d, \tag{4.6}$$

where B is a diagonal $|N| \times |N|$ matrix with the values of vector b in the corresponding diagonal entry. Unfortunately, the function $x^T [Q + B] x$ of the revised formulation in (4.6) is not necessarily convex. However, in a special case where $b > 0$, which guarantees the convexity of $x^T [Q + B] x$, we can choose to solve the program represented by (4.1), (4.2), (4.4), and (4.6) instead of the program represented by (4.1)-(4.4).

4.2 Alternative Constraints

We refer to constraints in set (4.5) as *Type I Cuts*, and in this section we develop a new set of linear cuts. Let Q^+ be the Moore-Penrose inverse (Moore 1920, Penrose 1955), or *pseudoinverse*, of covariance matrix Q, and in this section, we use the following properties of Q^+.

Property 1 *Since matrix Q is an $|N| \times |N|$ symmetric positive semidefinite matrix, matrix Q^+ is also an $|N| \times |N|$ symmetric positive semidefinite matrix.*

Property 2 *Matrices Q and Q^+ have the same rank and column and row spaces.*

Property 3 *If an $|N|$-dimensional row vector b is in the row space of Q, then $bQQ^+ = bQ^+Q = b$. Hence, $QQ^+Q = Q$ and $Q^+QQ^+ = Q^+$.*

Property 4 *For an $|N|$-dimensional row vector b, $bQ^+ \in \arg\min_z ||zQ - b||^2$ and $bQ \in \arg\min_z ||zQ^+ - b||^2$.*

For any vector $w \in \Re^{|N|}$, let constant

$$h = \sqrt{\frac{bQ^+b^T + 4w^TQw + 4bQ^+Qw}{bQ^+b^T + 4d}} > 0,$$

and let constant vector $u = w + \frac{1-h}{2}Q^+b^T$. Consider the following new set of constraints

$$\left(2u^TQ + hb\right)x \leq \frac{1}{2}\sqrt{\left(bQ^+b^T + 4d\right)\left(bQ^+b^T + 4bQ^+Qw + 4w^TQw\right)} - \frac{1}{2}bQ^+b^T - bQ^+Qw \qquad \forall w \in \Re^{|N|}. \tag{4.7}$$

Theorem 4. *For all $w \in \Re^{|N|}$, the associated constraint in set (4.7) is a valid inequality.*

Proof. The constraint

$$\left(\frac{2}{h}u^TQ + b\right)x \leq d + \frac{1}{h^2}u^TQu$$

in the set (4.5) is a valid inequality. Substituting in for u on the right-hand side

$$\left(\frac{2}{h}u^TQ + b\right)x \leq d + \frac{(1-h)^2}{4h^2}bQ^+QQ^+b^T + \frac{1-h}{h^2}bQ^+Qw + \frac{1}{h^2}w^TQw.$$

By Property 3,

$$\left(\frac{2}{h}u^TQ + b\right)x \leq \frac{1}{4}bQ^+b^T + d - \frac{1}{h}\left[\frac{1}{2}bQ^+b^T + bQ^+Qw\right] + \frac{1}{h^2}\left[\frac{1}{4}bQ^+b^T + bQ^+Qw + w^TQw\right].$$

Since $h > 0$,

$$\left(2u^TQ + hb\right)x \leq \frac{h}{4}\left[bQ^+b^T + 4d\right] - \frac{1}{2}bQ^+b^T - bQ^+Qw + \frac{1}{4h}\left[bQ^+b^T + 4bQ^+Qw + 4w^TQw\right].$$

Substituting in for h,

$$\left(2u^TQ + hb\right)x \leq \frac{1}{2}\sqrt{\left(bQ^+b^T + 4d\right)\left(bQ^+b^T + 4bQ^+Qw + 4w^TQw\right)} - \frac{1}{2}bQ^+b^T - bQ^+Qw.$$

□

The primary advantage of using constraints in set (4.7) to solve CURPR instead of using those in set (4.5) is that the ones in (4.7) are often more restrictive than those in set (4.7). In Theorem 7, we show that if vector b is in the row space of matrix Q, then constraints in set (4.7) are at least as restrictive as those in set (4.7).

Lemma 5. *For any $w \in \Re^{|N|}$, the right-hand side of the associated constraint in set (4.7) is less than or equal to the right-hand side of the one in set (4.5).*

Proof. By the triangle inequality, for each real vector $w \in \Re^{|N|}$,

$$\begin{aligned}
&\left[\sqrt{\frac{1}{4}bQ^+b^T + d} - \sqrt{\frac{1}{4}bQ^+b^T + bQ^+Qw + w^TQw}\right] \geq 0 \\
\implies &\frac{1}{2}bQ^+b^T + d - \sqrt{\left(\frac{1}{4}bQ^+b^T + d\right)\left(\frac{1}{4}bQ^+b^T + bQ^+Qw + w^TQw\right)} \\
&\qquad + bQ^+Qw + w^TQw \geq 0 \\
\implies &d + w^TQw \geq \frac{1}{2}\sqrt{\left(bQ^+b^T + 4d\right)\left(bQ^+b^T + 4bQ^+Qw + 4w^TQw\right)} \\
&\qquad - \frac{1}{2}bQ^+b^T - bQ^+Qw,
\end{aligned}$$

so the right-hand side of the constraint in set (4.7) is less than or equal to the one in set (4.5). □

Lemma 6. *If vector b is in the row space of covariance matrix Q, then for any $w \in \Re^{|N|}$,*

$$2u^TQ + hb = 2w^TQ + b. \tag{4.8}$$

Proof. Substituting in for u,

$$2u^TQ + hb = 2w^TQ + (1-h)bQ^+Q + hb. \tag{4.9}$$

Suppose b is in the column space of Q. Then by Property 3, $b = bQ^+Q$, so equation (4.8) is true. □

Theorem 7. *If vector b is in the row space of covariance matrix Q, then for any $w \in \Re^{|N|}$, the associated constraint in set (4.7) is at least as restrictive as the associated one in set (4.5).*

Proof. The proof of Theorem 7 is immediate from Lemmas 5 and 6. □

Corollary 8. *If vector b is in the row space of covariance matrix Q, then for any $w \in \Re^{|N|}$, the vector w is infeasible with respect to the associated constraint in set (4.7).*

Corollary 8 immediate follows from Theorem 7. For all $w \in \Re^{|N|}$, the hyperplane defined by the associated constraint in set (4.7) is given by

$$H = \left\{ x | (2u^TQ + hb)x = \frac{1}{2}\sqrt{(bQ^+b^T + 4d)(bQ^+b^T + 4bQ^+Qw + 4w^TQw)} \right. \\ \left. -\frac{1}{2}bQ^+b^T - bQ^+Qw \right\}$$

The boundary of the quadratic constraint (4.3) is given $\mathcal{Q} = \left\{x | x^TQx + bx = d\right\}$. In Theorem 9, we prove that if vector b is in the row space of matrix Q, then the hyperplane H is tangent to the boundary of the quadratic constraint $\mathcal{Q}$, implying that constraints (4.7) cannot be further tightened.

Theorem 9. *If vector b is in the column space of covariance matrix Q, then for any $w \in \Re^{|N|}$, the hyperplane H is tangent to the boundary of the quadratic constraint $\mathcal{Q}$.*

Proof. Since (4.7) is a valid inequality by Theorem 4, it remains to be shown that at least one point x resides within the both H and $\mathcal{Q}$. Consider the vector $\frac{1}{h}u$.

$$\begin{aligned}\frac{1}{h^2}u^TQu+\frac{1}{h}bu =&\frac{1}{4}bQ^+b^T-\frac{1}{h}\left[\frac{1}{2}bQ^+b^T+bQ^+Qw\right]\\&+\frac{1}{h^2}\left[\frac{1}{4}bQ^+b^T+bQ^+Qw+w^TQw\right]+\frac{1}{h}\left[bw+\frac{1-h}{2}bQ^+b^T\right]\\=&-\frac{1}{4}bQ^+b^T-\frac{1}{h}\left[bQ^+Qw-bw\right]\\&+\frac{1}{4h^2}\left[bQ^+b^T+4bQ^+Qw+4w^TQw\right]\\=&-\frac{1}{4}bQ^+b^T-\frac{1}{h}\left[bQ^+Qw-bw\right]+\frac{bQ^+b^T+4d}{4}\\=&d-\frac{1}{h}\left[bQ^+Qw-bw\right]\end{aligned}$$

If b is in the row space of Q, then by Property 3, $\frac{1}{h}u\in\mathcal{Q}$. By Lemma 6,

$$\begin{aligned}\left(2u^TQ+hb\right)\frac{1}{h}u&=\left(2w^TQ+b\right)\frac{1}{h}u\\&=\frac{2}{h}w^TQw+\frac{1}{h}bw+\frac{1-h}{h}bQ^+Qw+\frac{1-h}{2h}bQ^+b^T\\&=\frac{1}{2h}\left[4w^TQw+2bw+2bQ^+Qw+bQ^+b^T\right]-bQ^+Qw-\frac{1}{2}bQ^+b^T.\end{aligned}$$

By Property 3, $b=bQ^+Q$, so

$$\begin{aligned}\left(2u^TQ+hb\right)\frac{1}{h}u=&\frac{1}{2h}\left[4w^TQw+4bQ^+Qw+bQ^+b^T\right]-bQ^+Qw-\frac{1}{2}bQ^+b^T\\=&\frac{1}{2}\sqrt{\left(bQ^+b^T+4d\right)\left(bQ^+b^T+4bQ^+Qw+4w^TQw\right)}\\&-bQ^+Qw-\frac{1}{2}bQ^+b^T.\end{aligned}$$

Consequently, $\frac{1}{h}u\in H$. □

Theorems 7 and 9 are only relevant if vector b is in the row space of matrix Q, which may not be true for certain problems. For a case with $b=0$, we show in Lemma **??** that the constraints in (4.7) reduce to the following set:

$$2w^TQx\le 2\sqrt{dw^TQw},\qquad\forall w\in\Re^{|N|},\tag{4.10}$$

so calculating Q^+ is unnecessary. In fact, one drawback of using constraints in set (4.7) for the general case in which $b \neq 0$ is that the matrix Q^+ can only be calculated when all of the columns have been generated, which almost never occurs when solving the SPQCV. To accommodate this difficulty, let $\overline{N}$ be a subset of schedules $\overline{N} \subset N$, and let the vector $w = \begin{bmatrix} w_1 & 0 \end{bmatrix}^T$, where $w_1 \in \Re^{|\overline{N}|}$, and here 0 is an $|N| - |\overline{N}|$-dimensional column vector of zeros. By decomposing vectors x and b and matrix Q into appropriately dimensioned subcomponents, the associated constraint in set (4.5) is given by

$$\begin{aligned} 2 \begin{bmatrix} w_1 & 0 \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \leq d + \begin{bmatrix} w_1 & 0 \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} w_1 \\ 0 \end{bmatrix}, \end{aligned}$$

which reduces to the following constraint

$$\left(2w_1^T Q_{11} + b_1\right) x_1 + \left(2w_1^T Q_{12} + b_2\right) x_2 \leq d + w_1^T Q_{11} w_1. \tag{4.11}$$

The structure of constraint (4.11) allows us to generate the variables in x_2 dynamically. We propose a set of constraints, referred to as *Type II Cuts*, in which we replace the constant h with

$$h' = \sqrt{\frac{b_1 Q_{11}^+ b_1^T + 4 w_1^T Q_{11} w_1 + 4 b_1 Q_{11}^+ Q_{11} w_1}{b_1 Q_{11}^+ b_1^T + 4d}}$$

and the vector u by $u' = \begin{bmatrix} u_1 & 0 \end{bmatrix}^T$, where $u_1 = w_1 + \frac{1-h'}{2} Q_{11}^+ b_1^T$. Using similar arguments as those to develop constraint (4.11) and prove Theorem 4, the Type II Cuts are given by

$$\begin{aligned}
&\left(2u_1^T Q_{11} + h' b_1\right) x_1 + \left(2u_1^T Q_{12} + h' b_2\right) x_2 \\
&\leq \frac{1}{2}\sqrt{\left(b_1 Q_{11}^+ b_1^T + 4d\right)\left(b_{11} Q_{11}^+ b_1^T + 4 b_1 Q_{11}^+ Q_{11} w_1 + 4 w_1^T Q_{11} w_1\right)} \\
&- \frac{1}{2} b_1 Q_{11}^+ b_1^T - b_1 Q_{11}^+ Q_{11} w_1 \qquad \forall w_1 \in \Re^{|\overline{F}|}. \qquad (4.12)
\end{aligned}$$

4.3 Delayed Columns-and-Cuts Generation Algorithm

In this section, we propose an algorithm to solve the SPQCV. Unlike the quadratic constraint in the URPR, the quadratic constraint (4.3) is irreducible, however it can be represented by an infinite number of first-order constraints in set (4.5). We develop a Branch-and-Cut-and-Price method (BCP) for the SPQCV. Using BCP, a continuous relaxation of SPQCV (CSPQCV) is solved in a branch-and-bound tree to ensure an integer solution. We propose the *Delayed Columns-and-Cuts Generation* Algorithm (DCCG) to solve the CSPQCV in a branch-and-bound node. Let π and ρ be dual vectors for constraint sets (4.2) and (4.5), respectively, and let $\bar{c}_j$ be a reduced cost of each variable x_j, the DCCG is shown in Algorithm 4.

When implemented, the DCCG cut generation step can follow the *cut selection strategy*:

- If the vector b is in the row space of matrix Q, then add a Type II Cut.
- Otherwise, add a Type I Cut.

For the computational experiments, we refer to [77] which implemented DCCG in a ship scheduling problem with limited risk. The computational experiments were done on three medium-size ship scheduling instances, each with a combination of six levels of limited risks. The results show that DCCG can efficiently find good ship scheduling solutions within a reasonable time limit. In 15 out of 18 instances, implementing DCCG with the cut selection strategy requires the same or fewer cuts to find the best known solution than adding the type I cut alone.

Algorithm 4 Delayed Column-and-Cut Generation Algorithm (DCCG)

Initialization Step: Let $\mathcal{W} \leftarrow \emptyset$ be a subset of linear constraints from (4.5). Generate a subset of variable $\overline{N} \subset N$

Restricted Master Problem (RMP) Step: Solve CSPQCV over the set of subsets $\overline{N}$, and first-order constraint set $\mathcal{W}$ to get a solution (x^*, π^*, ρ^*).

if $x^{*T}Qx + bx^* > d + \varepsilon$, **then**

Cut Generation Step: Update the constraint set $\mathcal{W} \leftarrow \mathcal{W} \cup \{x^*\}$ and return to the RMP step.

else

Find a variable $\bar{x}_j$ that minimizes the reduced cost $\bar{c}_j$.

if $\bar{c}_j \geq 0$, **then**

Return the optimal solution x^*.

else

Column Generation Step: $\overline{N} \leftarrow \overline{N} \cup \{\bar{x}\}$ and return to the RMP step.

end if

end if

CHAPTER 5

UAV ROUTING PROBLEM WITHOUT TIME WINDOWS

5.1 Introduction

In this chapter, we discuss the UAV routing problem with limited risk but without time windows (URPR). The URPR is very similar to the URPRTW in Chapter 3, that is a set of routes that visits all targets with a minimum expected fuel burn is determined and a variance of the expected fuel burn is required to be less than a predetermine amount, d. However, in the URPR, targets do not have time window requirements. A target can be visited by a UAV at any time in an operation time table. This relaxation increases the computational complexity of the URPR because the total number of possible routes exponentially increases as the number of targets in the problem increases. We propose the BCP methodology as a solution method to the URPR. The absence of the time windows greatly changes the column generation step of the BCP methodology. Because targets can be visited at any time, an instance graph G in the column generation step is very complex, and generating a variable with a negative reduced cost is very difficult. Nevertheless, the URPR can

be modeled as the same set-partitioning problem as in the URPRTW. The URPR is modeled as a set-partitioning problem as follows

$$\min \sum_{u \in U} \sum_{f \in F_u} E\left[\tilde{e}_{uf}\right] x_{uf} \tag{5.1}$$

$$\text{s.t.} \sum_{u \in U} \sum_{f \in F_u} a_{kuf} x_{uf} = 1 \qquad \forall k \in K, \tag{5.2}$$

$$\sum_{f \in F_u} x_{uf} \leq n_u \qquad \forall u \in U, \tag{5.3}$$

$$-\sum_{u \in U} \sum_{f \in F_u} \beta_{uf} x_{uf} \leq \sqrt{\frac{d}{var(\tilde{v}^w)}} \tag{5.4}$$

$$\sum_{u \in U} \sum_{f \in F_u} a_{Cuf} x_{uf} \leq |C| - 1 \qquad \forall C \in \mathcal{D}, \tag{5.5}$$

$$x_{uf} \in \{0, 1\} \qquad \forall u \in U, f \in F_u, \tag{5.6}$$

where

$$a_{Cuf} = \begin{cases} 1, & \text{if a variable } x_{uf} \in C; \\ 0, & \text{otherwise.} \end{cases}$$

The URPR in (5.1)-(5.6) is the same as the URPRTW, where the objective function (5.1) minimizes the expected fuel burn for UAVs to visit all targets. The set-partitioning constraints in set (5.2) require all targets must be visited exactly once. The constraints in set (5.3) imply that each UAV in a group is assigned to at most one route. The constraint in (5.4) is a linear variance constraint that maintains the variance of the fuel burn to be less than a predetermine amount d. Note that the linear variance constraint is simplified from a quadratic constraint where the simplification was discussed in Section 3.3. The constraints in set (5.5) are MDS constraints that are generated in the cut generation step in order to encourage integrality in a solution. Finally, constraints in set (5.6) are binary requirements for decision variables x_{uf}. Observe that the absence of the time windows does not affect the general model of the URPR; that is, the mathematical models of the URPR and the URPRTW

are the same. Consequently, we propose the BCP methodology to solve the URPR since the methodology was successfully implemented for the URPRTW in Chapter 3. In the next section, we discuss the implementation of the BCP methodology to the URPR. The discussion includes the column generation step, which is a major distinction between the implementations of the BCP methodology to the URPR and the URPRTW. Also, a simple path heuristic that quickly generate valid simple paths is discussed. However, we do not discuss the cut generation step as the step is the same as discussed in Section 3.5.3.

5.2 Branch-and-Cut-and-Price Methodology

In this section, we discuss the BCP methodology as a solution method to solve the URPR. Recall that the BCP methodology is an LP-based branch-and-bound algorithm for solving mixed integer programming problems. In our case, the considered integer program is the URPR, which is shown in (5.1)-(5.6). The integer programming problem is solved within a branch-and-bound tree to ensure an integer solution. In each node of the branch-and-bound tree, the RMP, which is a continuous relaxation of the URPR (CURPR), is solved. Originally, the RMP only includes a subset of variables to overcome the difficulty due to a large number of possible variables. Other variables are dynamically generated in the column generation step and added to the RMP only when they are needed. Bounds are updated, and branching is performed. As mentioned in Section 3.5, *follow-on* branching is used as our branching logic. Recall that the discussed algorithm is referred to as the DCG algorithm in

Chapter 3. The terminology is also used in this chapter. In the following, we present the CURPR as

$$\min \sum_{u \in U} \sum_{f \in F_u} E\left[\tilde{e}_{uf}\right] x_{uf} \tag{5.7}$$

$$\text{s.t.} \sum_{u \in U} \sum_{f \in F_u} a_{kuf} x_{uf} = 1 \qquad \forall k \in K, \tag{5.8}$$

$$\sum_{f \in F_u} x_{uf} \leq n_u \qquad \forall u \in U, \tag{5.9}$$

$$-\sum_{u \in U} \sum_{f \in F_u} \beta_{uf} x_{uf} \leq \sqrt{\frac{d}{var(\tilde{v}^w)}} \tag{5.10}$$

$$\sum_{u \in U} \sum_{f \in F_u} a_{Cuf} x_{uf} \leq |C| - 1 \qquad \forall C \in \mathcal{D}, \tag{5.11}$$

$$0 \leq x_{uf} \leq 1 \qquad \forall u \in U, f \in F_u, \tag{5.12}$$

where

$$a_{Cuf} = \begin{cases} 1, & \text{if a variable } x_{uf} \in C; \\ 0, & \text{otherwise.} \end{cases}$$

Given the RMP in (5.7)-(5.12) and the optimal dual vector $(\pi_k^*, \pi_u^*, \rho_w^*, \rho_c^*)$, where π_k, π_u, ρ_w, and ρ_c are dual variables of constraints in set (5.8), (5.9), (5.10), and (5.11), respectively, the reduced cost for a variable x_{uf} is

$$\bar{c}_{uf} = \sum_{l \in f} \{\alpha_l + \beta_l \rho_w^*\} - \sum_{k \in K} \pi_k^* - \pi_u^* - \sum_{C \in \mathcal{D}} a_{Cuf} \rho_C^*, \qquad \forall u \in U, \forall f \in F_u. \tag{5.13}$$

Recall that in Chapter 3, we let $RC_{uf} = \sum_{l \in f} \{\alpha_l + \beta_l \rho_w^*\} - \sum_{k \in K} \pi_k^* - \pi_u^*$, and $\rho_{uf} = \sum_{C \in \mathcal{D}} a_{Cuf} \rho_C^*$. Therefore the reduced cost can be simplify to $\bar{c}_{uf} = RC_{uf} - \rho_{uf}$. Since the URPR is a minimization problem, a variable must have a negative reduced cost, $\bar{c}_{uf} \leq 0$, to be added to the RMP. The column generation step in Section 5.2.1 discusses algorithms that generate a variable x_{uf} with a negative reduced cost. We also revisit a difficulty that may arise due to previously generated variable in Section 5.2.1 that is a path $\tilde{f}$ from the column generation step might have a minimum cost,

that is $RC_{u\tilde{f}} \leq RC_{uf}, \forall u \in U$ and $\forall f \in F_u$, but its reduced cost might not be minimal, that is $\exists u \in U$ and $\exists f \in F_u$ such that $RC_{u\tilde{f}} - \rho_{u\tilde{f}} > RC_{uf} - \rho_{uf}$. The DCG algorithm that follows the steps in the Algorithm 1 can be represented in a flow chart as in Figure 5.1.

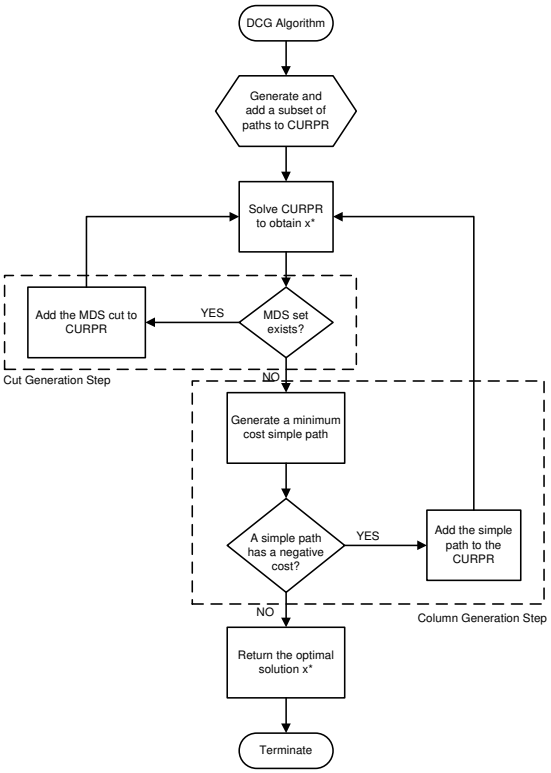

Figure 5.1. A DCG Algorithm.

The DCG algorithm is implemented within each node of a branch-and-bound tree. The algorithm starts with an initialization step, where a subset of routes are generated and added to the RMP. We solve the RMP to get a solution (x^*, π^*, ρ^*). If the solution (x^*, π^*, ρ^*) is fractional, the cut generation step attempts to generate an MDS constraint that eliminates the solution. However if an MDS constraint does not exist, the column generation step is invoked where a variable with a negative reduced cost is generated and added to the RMP. If there is no variable with a negative reduced cost, the column generation step is terminated. A solution of a node in a branch-

and-bound tree is then branched on to obtain an integer solution. As previously mentioned, the absence of time windows mainly affect the column generation step. We discuss the column generation step of the DCG algorithm in the following section.

5.2.1 Column Generation

In this section, we discuss a procedure to generate new variables to be added to the RMP. Because the URPR is a minimization problem, a new variable must have a negative reduced cost to be added to the RMP. In order to find variables with negative reduced costs, we construct an instance graph $G = (V, E)$ where $V = \{v_0, v_1, v_2, ..., v_n, v_{n+1}\}$ is a set of nodes, $E = \{e_{ij} = (v_i, v_j) : v_i, v_j \in V\}$ is a set of edges, and c_{ij} is a cost of traveling from node v_i to node v_j. In the graph G, nodes represent targets where a source node v_0 and a terminal node v_{n+1} represent a common depot, and edges represent traveling paths among targets. Let c_{ij} be equal to a reduced cost of visiting target v_j after visiting target v_i, that is $c_{ij} = \alpha_l + \beta_l \rho_w^* - \pi_k^*$. Therefore, to add a variable to the RMP is simply finding a path that starts from the source node v_0 and ends at the terminal node v_{n+1} that has a negative cost. Unlike the instance graph in the URPRTW where $c_{ij} = \infty$ if the traveling from node v_i to node v_j violates time windows, nodes in the instance graph G of the URPR can be accessed from all other nodes due to the absence of the time windows. This is a major distinction between the instance graph in the URPR, shown in Figure 5.2, and the instance graph in the URPRTW, shown in Figure 3.1.

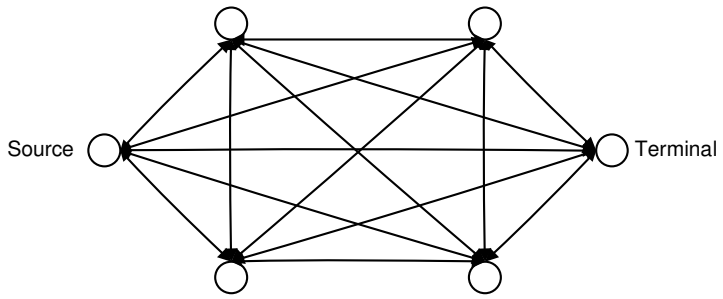

Figure 5.2. An instance graph of the URPR.

It must be pointed out that the total number of possible routes directly depends on the total number of targets in a problem. Even with a considerably small URPR, the number is usually very large. Figure 5.3 shows the relationship between the total number of possible routes and the total number of targets in the URPR. For example, there are 9,864,100 possible routes in the 10-target problem, 3.55E+12 possible routes in the 15-target problem, and 6.61E+18 possible routes in the 20-target problem. It is obvious that the total number of possible routes exponentially increase when the problem size increases. For this reason, the URPR is extremely difficult and almost impossible to solve with traditional methods. For example, consider the DPSP algorithm that was proposed in Chapter 3, the algorithm requires a labeling process to keep track of previously visited nodes in a path. The additional labeling process is equivalent to an explicit enumeration of all possible routes. Implementation of the DPSP algorithm is very slow due to extensive computational efforts and hence not suitable for the column generation step of the URPR.

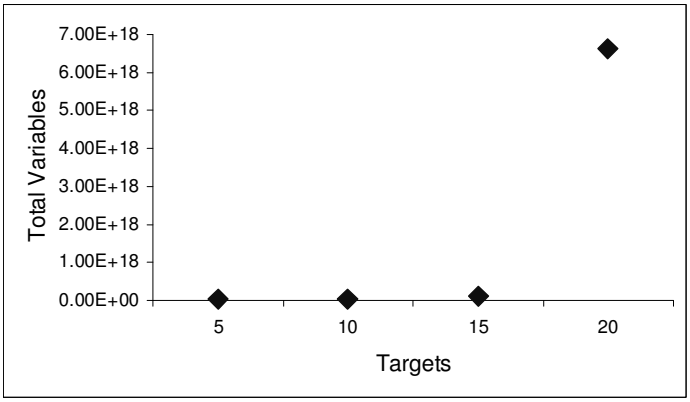

Figure 5.3. A total number of possible routes.

In this section, we propose the IPSP algorithm as an algorithm to find the minimum cost path in an instance graph G in the column generation step of the URPR. The IPSP algorithm constructs an integer program as a column generation subproblem called a *Minimum-Cost Network Flow* problem (MCNF). The MCNF

problem of the URPR is very similar to the MCNF problem of the URPRTW, but includes an additional set of constraints. The MCNF problem is as follows

$$\min \sum_{l \in L} \{\alpha_l + \beta_l \rho_w^* - \pi_k^*\} y_l, \tag{5.14}$$

$$\text{s.t.} \sum_{l \in \delta_{v_i}^+} y_l - \sum_{l \in \delta_{v_i}^-} y_l = b(v_i) \qquad \forall v_i \in V, \tag{5.15}$$

$$\sum_{l \in \delta_{v_i}^+} y_l \leq 1 \qquad \forall v_i \in V, \tag{5.16}$$

$$y_l \in \{0, 1\} \qquad \forall l \in L, \tag{5.17}$$

where $\delta_{v_i}^+ = \{j : (i,j) \in E\}$, and $\delta_{v_i}^- = \{j : (j,i) \in E\}$ are sets of edges that enter and leave node v_i, respectively. The objective function in (5.14) minimizes a total cost of traveling from the source node v_0 to the terminal node v_{n+1}. The node conservation constraints in set (5.15) maintain inflows and outflows of each node v_i, where $b(v_0) = 1, b(v_{n+1}) = -1$, and $b(v_i) = 0, \forall v_i \in V \setminus \{v_0, v_{n+1}\}$. The outflow limitation constraints in set (5.16) limit the outflow of each node to be at most one. Although the outflow limitation constraints are not essential to the MCNF problem, they reduce computational efforts due to negative cost cycles. Finally, binary requirements in (5.17) require a decision variable y_l to be a binary variable, that is $y_l = 1$ if the edge $e_l = (v_i, v_j) \in E$ is included in the MCNF solution, and $y_l = 0$ otherwise. A solution to the MCNF problem is a valid simple path that starts from a source node and ends at a terminal node with a minimum cost. Observe that the coefficient of a decision variable y_l in (5.14) is $\alpha_l + \beta_l \rho_w^* - \pi_k^*$. During execution, the MCNF problem is solved when an integer programming solver finds an optimal solution y_l^* with the total cost of $\sum_{l \in L} \{\alpha_l + \beta_l \rho_w^* - \pi_k^*\} y_l^*$. Assume that the optimal route y_l^* has not been added to the RMP, one must further add the total cost by a

cost of using a UAV u, $-\pi_u^*$, to receive an actual reduced cost of the optimal route y_l^*. The reduced cost of the optimal route f^*, is given by

$$\bar{c}_{uf^*} = \sum_{l \in L} \{\alpha_l + \beta_l \rho_w^* - \pi_k^*\} y_l^* - \pi_u^*. \tag{5.18}$$

The reduced cost in (5.18) is the same as (3.39) when the cost associated with MDS constraints is zero, $\sum_{C \in \mathcal{D}} a_{Cuf} \rho_C^* = 0$. By contrast, if the optimal route $\tilde{y}_l$ has a negative reduced cost but was previously added to the RMP, then the reduced cost of the optimal route $\tilde{f}$ is given by

$$\bar{c}_{u\tilde{f}} = \sum_{l \in L} \{\alpha_l + \beta_l \rho_w^* - \pi_k^*\} y_l^* - \pi_u^* - \sum_{C \in \mathcal{D}} a_{Cu\tilde{f}} \rho_C^* = 0, \tag{5.19}$$

where the summation of the dual variables associated with the MDS constraints are negative, $\sum_{C \in \mathcal{D}} a_{Cu\tilde{f}} \rho_C^* < 0$. According to the aforementioned difficulties in Section 3.5.2, to ensure optimality of the column generation step, a route elimination constraint will be added to the MCNF problem to eliminate the optimal route $\tilde{f}$, if $\tilde{f}$ was previously added to the RMP. The route elimination constraint is

$$\sum_{l \in \tilde{f}} y_l \leq |\Omega(\tilde{f})|, \tag{5.20}$$

where $\Omega(\tilde{f})$ is a set of targets visited in route $\tilde{f}$. We solve the MCNF problem and dynamically add route elimination constraints as in (5.20) until the minimum cost route that has not been added to the RMP is found. If an optimal route has a negative reduced cost, then it will be added to the RMP. However, if the optimal route has a non-negative reduced cost, then it can be concluded that there are no more variables to be added to the RMP, and the column generation step is terminated.

Moreover, since the cost of traveling on the edge e_{ij}, that is c_{ij}, is not restricted, it is a well-known problem that the optimal solution of the MCNF problem is usually embedded with negative cost cycles. This presents a great difficulty because the optimal solution with negative cost cycles is invalid to be added as a variable to the RMP.

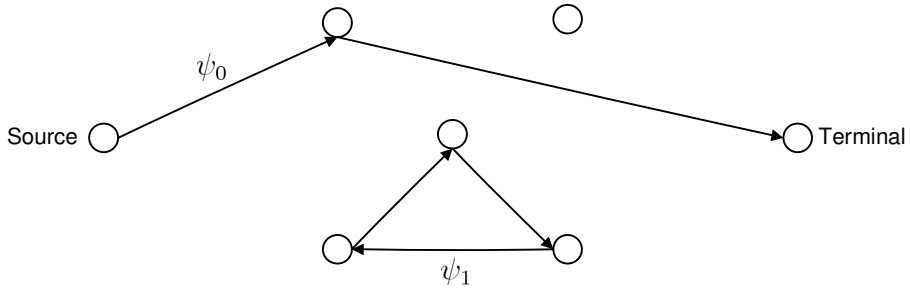

Figure 5.4. An MCNF solution with a cycle.

Consider the route in Figure 5.4. Although the route is a feasible solution for the MCNF problem, it includes an embedded negative cost cycle. The route can be decomposed into two parts. The first part is a simple path that leaves the source node and enters the terminal node, and the second part is a negative cost cycle. It is obvious that the cost of cycles are negative, however a cost of a simple path is ambiguous. That is the cost of a simple path can be positive, negative, or zero as long as when it is combined with the costs of cycles, they produce the minimum cost route in the graph G. In a graph theory context, both simple path and negative cost cycle are referred to as *walks*, ψ. Although a time consuming task, as widely suggested in literature, cuts should be generated to eliminate negative cost cycles. In this research, because of the characteristics of the instance graph G, it is likely that adding cuts that eliminate negative cost cycles alone is going to be inefficient. For example, given an optimal route f^*, in Figure 5.4, which is composed of a simple path ψ_0 and a cycle ψ_1. If the simple path ψ_0 has a negative cost, it alone can be added to the RMP even if the optimal route f^* includes the cycle ψ_1. In another case, if the simple path ψ_0 has a positive cost, it cannot be added to the RMP, and then a walk elimination constraint to eliminate a walk ψ_1 is added to the MCNF problem. A walk elimination constraint is

$$\sum_{l \in \psi} y_l \leq b(\psi), \tag{5.21}$$

where $b(\psi) = |\Omega(\psi)|$ if ψ is a simple path, and $b(\psi) = |\Omega(\psi)| - 1$ if ψ is a cycle, and $\Omega(\psi)$ is a set of targets visited in a walk ψ. The cut in (5.21) will be dynamically added, and the MCNF problem is resolved to obtain a new optimal route. However, it is likely that the same simple walk with positive cost ψ_0 may still be a part of the new optimal route $\tilde{f}$ but only accompanied with new cycles. The procedure is considered ineffective, and it struggled to quickly generate valid simple paths in our preliminary computational experiments. To overcome this difficulty, we consider both a simple path with a positive cost, and negative cost cycles as *invalid walks*. Therefore given an optimal route f^*, we will add walk elimination constraints as in (5.21) to eliminate all invalid walks within a given solution until a valid simple path is found. The Figure 5.5 is a complete representation of the column generation step.

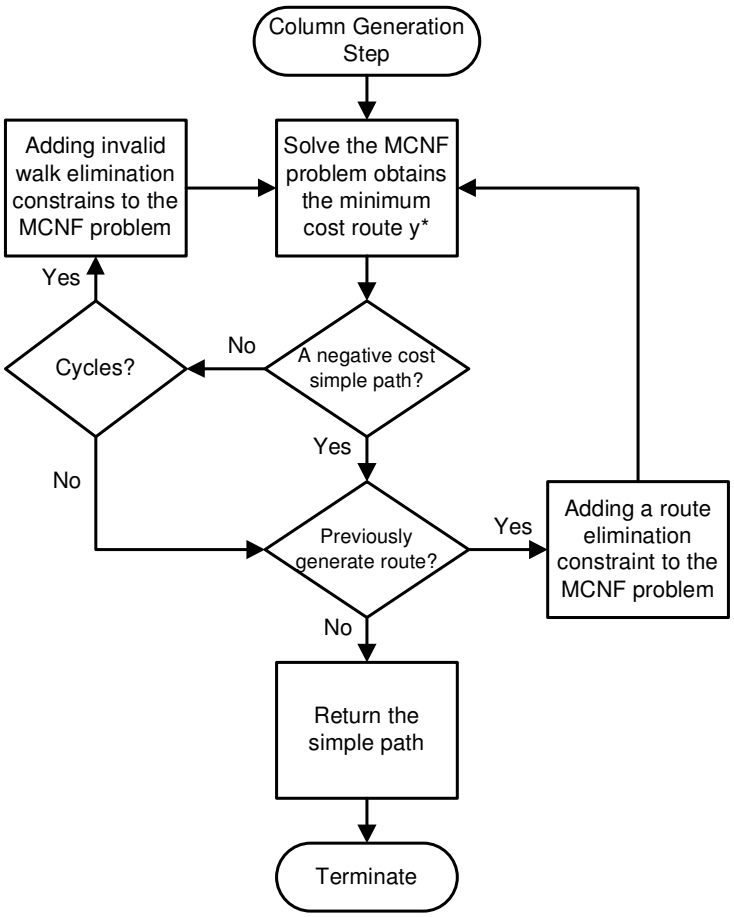

Figure 5.5. A column generation of the URPR.

The column generation step starts with constructing a column generation subproblem, which is the MCNF problem. The MCNF problem is solved by an integer solver to find an optimal route f^*. The optimal route f^* is then examined for a simple path with a negative cost. If it exists and has not been previously added, the simple path is added to RMP. However, if the simple path has a positive cost, walk elimination constraints are added to eliminate *all* invalid walks, including cycles, and the MCNF problem is resolved. The steps are repeated until a valid simple path is found. Recall that the cost of a walk is $\sum_{l\in\psi}(\alpha_l + \beta_l\rho_w^* - \pi_k^*)y_l$, hence the validity of a walk elimination constraint depends upon an RMP solution, (x^*, π^*, ρ^*). Therefore, a walk elimination constraint is valid only within the iteration that they were generated. Specifically, a simple path with a positive cost, ψ_0, may have a negative cost in the next iteration of the column generation step and hence a walk elimination constraint that eliminates the walk ψ_0 must be excluded in the next iteration. We explore the advantage of adding a walk elimination constraint that eliminates a simple path with a positive cost in Section 5.4. In the next section, we discuss a simple path heuristic, which is a procedure that generates simple paths from negative cost cycles in order to quickly provide an eligible variable to be added to the RMP.

5.2.2 Simple Path Heuristic

In this section, we discuss a procedure to generate simple paths from a negative cost cycle embedded in an MCNF solution. As mentioned in Section 5.2.1, a walk elimination constraint, as in (5.21), is added to eliminate an invalid walk, which is either a simple path with a positive cost or a negative cost cycle. The walk elimination constraints are dynamically added to the MCNF problem until a valid simple path is found. However, this procedure is very time consuming and inefficient with a larger problem in which an instance graph G is very complicate. In this section, we propose a heuristic that generates simple paths from a negative cost cycle. Since, the

simple paths that are generated from a negative cost cycle are likely to have negative costs, they can be added to the RMP. The proposed heuristic will prevent the column generation step from being stuck finding a minimum cost path in an instance graph G, but instead it generates an eligible path with a negative cost to be added to the RMP.

Consider a solution y_l^* from solving the MCNF problem in Figure 5.4. The solution includes a simple path ψ_0 and a negative cost cycle ψ_1. As mentioned in Section 5.2.1, if a walk ψ_0 has a negative reduced cost, it will be added to the RMP, hence no walk elimination constraint is needed. However, if the walk ψ_0 has a positive reduced cost, that is $\sum_{l\in\psi_0}(\alpha_l + \beta_l\rho_w^* - \pi_k^*) - \pi_u^* \geq 0$, and the walk ψ_1 has a negative reduced cost, that is $\sum_{l\in\psi_1}(\alpha_l + \beta_l\rho_w^* - \pi_k^*) - \pi_u^* < 0$, the heuristic will ignore the positive cost path ψ_0, and it will generate simple paths from the negative cost cycle ψ_1. The heuristic generates a simple path by deleting an edge $l \in \psi_1$, and adding two edges to construct a simple path. Given a link l is a part of the cycle ψ_1, where the link l connects node v_i and node v_j. If the link l is deleted in order to construct a simple path, a link from the source node to node v_j and a link from node v_i to the terminal node will be added. Figure 5.6 shows a simple path $\hat{\psi}$ generated from deleting a link l that connects between node v_i and node v_j, when a link l is a part of the cycle ψ_1.

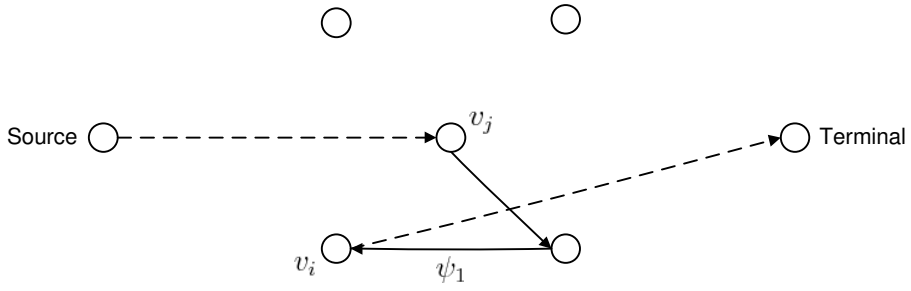

Figure 5.6. A heuristic route generated from a cycle.

The simple path $\hat{\psi}$, which was heuristically generated, starts at the source node, visits targets, and ends at terminal nodes. Assume that the simple path $\hat{\psi}$ has not been added to the RMP and the reduced cost is negative, that is $\sum_{l\in\hat{\psi}}(\alpha_l + \beta_l\rho^*_w - \pi^*_k) - \pi^*_u < 0$, the simple path $\hat{\psi}$ is a candidate to be added to the RMP. The heuristic will delete all edges included in a cycle to generate new simple paths. It is obvious from a negative cost cycle ψ_1, there are $|\Omega(\psi_1)|$ heuristic routes that can be generated. The heuristic route with the least negative reduced cost will be added to the RMP. In the next section, we present the implementation of our proposed solution method to the URPR.

5.3 Computational Experiments

In this section, we discuss an implementation of the BCP methodology to the URPR. We implemented the BCP methodology using COIN-OR on a Dual 3.06-Ghz Intel Xeon workstation, and CPLEX 9.120 was used as the LP solver and the IP solver to solve the CURPR and the MCNF problem, respectively. We created problem instances that are combinations of 1 UAV group with 10, 15, and 20 targets, where locations, wind speed, wind direction, and the variance of the wind were randomly generated. Observe that, sizes of the considered problem instances are different from those discussed in Chapter 3. In the URPRTW, larger problem instances were considered. They are 30, 60, and 90 targets, all of which were optimally solved. However, with the mentioned difficulty due to the absence of time windows, the problem instances of the URPR are much harder to solve, and problem instances that are larger than those presented in this section are very difficult. Nevertheless, problem instances with targets range from 10-20 targets are considered realistic in the focused military and scientific applications.

In the following computational experiments, each problem instance was solved using three algorithms, the DCG algorithm, the DCG-HEU algorithm, and the DCG-

HEU-MDS algorithm. The DCG algorithm follows the procedure discussed in Section 5.2 that the CURPR is solved within a branch-and-bound tree where the column generation step generates variables with negative reduced costs to be added to the RMP. The DCG-HEU algorithm heuristically generates simple paths if an MCNF solution includes negative cost cycles. Consequently, the heuristically generated path with the least negative cost is added to the RMP. Finally, the DCG-HEU-MDS algorithm attempts to generate MDS constraints in the cut generation step to encourage integrality in the RMP solution. For the computational experiments, we still follow the procedure discussed in Chapter 3 where the *deterministic* case was first solved without the variance constraint in (5.4). We then used the variance of the optimal solution in the deterministic case as a reference level for other values of d. The considered levels of d were 95%, 90%, 85%, 80%, 75%, and 70%. The computational results of the DCG algorithm are shown in Table 5.1. The columns include the percentage of standard deviation with respect to that of the deterministic case (d), the actual standard deviation ($MAX\ SD$), the standard deviation of the best found solution ($SD\ BS$), the difference in the percentage between the standard deviation of the best found solution and that of the deterministic case ($SD\ (\%)$), the expected fuel burn of the best found solution (BS), the percentage of the expected fuel burn of the best found solution with respect to the that of the deterministic case ($BS\ (\%)$), the total computational time until finding the best found solution ($CPU\ BS$), the total number of variables added to the RMP until receiving the best found solution ($Vars\ BS$), the total number of subproblem cuts added until finding the best found solution ($SP\ Cuts\ BS$), the total computational time until the algorithm terminates (CPU), the total number of variables added until the algorithm terminates ($Vars$), and the total number of subproblem cuts added until the algorithm terminates ($SP\ Cuts$). Note that all times are measured in CPU seconds, and the computational time limit was

10 hours. The *Time* in the column *CPU* means algorithms were terminated because the time limit was reached.

Table 5.1. Computation results of the DCG Algorithm

1 group, 10 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.999	59.274	31.98	33120	118.16	51.38	361	6431	84.14	901	12437
75%	65.356	61.907	28.96	31591	112.70	43.86	181	3842	67.25	359	6434
80%	69.713	69.371	20.39	29832	106.43	2.57	30	352	25.01	201	3088
85%	74.070	69.371	20.39	29832	106.43	5.97	43	650	18.60	165	2423
90%	78.427	69.371	20.39	29832	106.43	31.06	154	3255	42.48	368	5553
95%	82.784	69.371	20.39	29832	106.43	28.78	135	2922	47.56	566	6748
100%	87.141	87.141	0.00	28031	100.00	1.90	30	313	2.99	38	453
1 group, 15 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	63.116	60.007	33.45	34463	122.32	9259.61	112	24909	Time	281	65466
75%	67.625	-	NA	-	NA	-	-	-	Time	107	44998
80%	72.133	70.529	21.78	30118	106.90	1582.64	54	8409	Time	218	54574
85%	76.641	75.557	16.20	29231	103.75	7208.32	135	22240	28696.86	435	84816
90%	81.150	75.557	16.20	29231	103.75	1341.83	57	8204	Time	545	94545
95%	85.658	75.557	16.20	29231	103.75	1037.48	46	5691	Time	344	62096
100%	90.166	90.166	0.00	28174	100.00	46.15	58	1554	178.12	95	3373
1 group, 20 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.945	-	NA	-	NA	-	-	-	Time	7	7683
75%	65.298	-	NA	-	NA	-	-	-	Time	7	7683
80%	69.651	-	NA	-	NA	-	-	-	Time	7	7683
85%	74.004	-	NA	-	NA	-	-	-	Time	7	7683
90%	78.357	-	NA	-	NA	-	-	-	Time	7	7683
95%	82.711	-	NA	-	NA	-	-	-	Time	7	7683
100%	87.064	87.064	NA	32118	NA	463.90	100	4426	Time	130	11611

Consider the computational results in Table 5.1. The optimal solutions of the deterministic cases for the 10 targets, 15 targets, and 20 targets problem instances are 28031 lbs with 7594 variance ($= 87.141^2$), 28174 lbs with 8130 variance ($= 90.166^2$), and 32118 lbs with 7580 variance ($= 87.064^2$) of expected fuel burn, respectively. The reported standard deviations of the deterministic cases are reduced to 95%, 90%, 85%, 80%, 75%, and 70% to obtain six subcases. For example, in the 10-target problem instance, the variances can be reduced to 4812 ($= 69.371^2$), 3832 ($= 61.907^2$), and 3513 ($= 59.274^2$), while the expected fuel burns increase to 29832 lbs, 31591 lbs, and 33120 lbs, respectively. Except the 20 targets problem instance, the result in Table 5.1 indicates that as variance decreases, expected fuel burn increases. The behavior is intuitive since a route with less expected fuel burn has to be disregard if it generates higher variance. Moreover, using the DCG algorithm, only the 10-target problem instance can be optimally solved in all of variance limiting cases. The computational

time used in all of cases of the 10-target problem instance are considered reasonable, and each solved in less than two minutes. There are only two cases in the 15-target problem instance, and one case of the 20-target problem instance that were optimally solved. This emphasizes our previously mentioned difficulty that the computational complexity of the URPR exponentially increases as the problem size increases. However, in the 15-target problem instance, the DCG algorithm was able to produce good quality solutions, where in the 20-target problem instance the DCG algorithm struggled with generating enough variables to produce feasible solutions in six of the seven cases. We show efficient frontiers that represent the relationship between

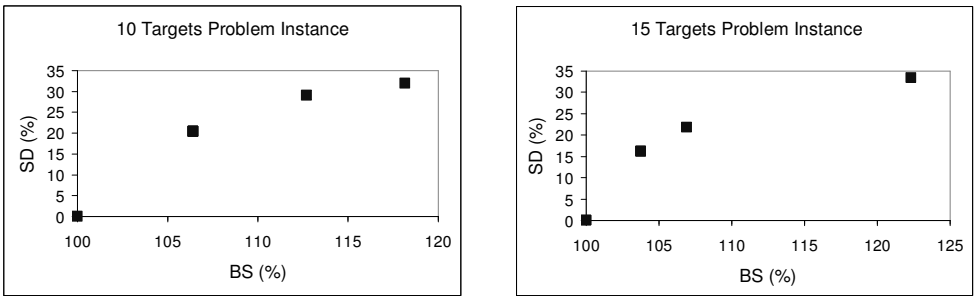

Figure 5.7. Efficient frontiers for 10, and 15 targets problem instances.

the reduction of the standard deviation and the increase of the expected fuel burn in Figure 5.7. The figure does not include the efficient frontier of the 20-target instance and the 75% case of the 15-target problem instance since they do not have feasible solutions. All combinations of *SD (%)* and *BS (%)* in the figure are non-dominated. Decision makers can utilize the efficient frontiers and choose the best combination of expected fuel burn and variance of fuel burn according to their risk preferences.

The results from implementing the DCG-HEU algorithm are shown in Table 5.2. As mentioned before, if a solution of the MCNF problem includes negative cost cycles, then the heuristically generated path with the least negative reduced cost will be added to the RMP. Consequently, the total number of generated variables, *Vars BS* and *Vars*, in the DCG-HEU algorithm are more that of the DCG algorithm. Also,

Table 5.2. Computation results of the DCG-HEU Algorithm

1 group, 10 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.999	59.274	31.98	33120	118.16	33.13	379	5750	56.15	867	10899
75%	65.356	63.947	26.62	31591	112.70	20.89	159	2505	38.94	326	5065
80%	69.713	69.371	20.39	29832	106.43	2.60	58	497	24.46	219	3413
85%	74.070	69.371	20.39	29832	106.43	4.64	80	867	16.83	223	2821
90%	78.427	69.371	20.39	29832	106.43	21.86	138	2501	29.06	384	4523
95%	82.784	69.371	20.39	29832	106.43	21.20	155	2698	42.78	728	7877
100%	87.141	87.141	0.00	28031	100.00	0.53	28	158	2.46	35	321
1 group, 15 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	63.116	60.007	33.45	34463	122.32	7033.98	153	8675	Time	243	24886
75%	67.625	67.577	25.05	31971	113.48	29914.59	319	33034	Time	773	59327
80%	72.133	70.529	21.78	30118	106.90	842.42	123	3089	Time	439	46305
85%	76.641	75.557	16.20	29231	103.75	16.89	189	2558	22535.58	531	37753
90%	81.150	75.557	16.20	29231	103.75	2046.91	214	8387	30479.51	702	58175
95%	85.658	75.557	16.20	29231	103.75	803.69	276	9694	Time	620	47525
100%	90.166	90.166	0.00	28174	100.00	35.38	94	1389	107.13	108	2337
1 group, 20 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.945	52.602	NA	56741	NA	794.45	168	5172	Time	277	12806
75%	65.298	56.765	NA	46347	NA	823.42	169	5575	Time	234	9422
80%	69.651	68.368	NA	53763	NA	705.24	163	5137	Time	264	11546
85%	74.004	-	NA	-	NA	-	-	-	Time	290	14973
90%	78.357	77.658	NA	42328	NA	1593.99	168	5518	Time	309	15784
95%	82.711	80.202	NA	64293	NA	18.9	99	1518	Time	285	12861
100%	87.064	87.064	NA	32118	NA	1277.31	110	2763	2952.25	129	4558

the total number of subproblem cuts, *SP Cuts BS* and *SP Cuts*, in the DCG-HEU algorithm are less than that of the DCG algorithm. This is because instead of repeatedly adding walk elimination constraints as in (5.21) to the MCNF problem until the minimum cost route without a negative cost cycle is found, the DCG-HEU algorithm heuristically generates simple paths and adds them to the RMP immediately. Although the added simple path might not be the minimum cost path in an instance graph G, it has a negative cost. It must be noted that the relationship between the reduction of the standard deviation and the increase of the expected fuel burn still holds. A significant improvement of the DCG-HEU algorithm over the DCG algorithm can be seen in the 20-target problem instance. Although the quality of the solutions in the 20-target problem instance is poor compared to the optimal solution in its deterministic case, the DCG-HEU algorithm found solutions in six out of seven cases compared to one out of seven with the DCG algorithm. The computational time improvements will be discuss later in this section.

In Table 5.3, the results from implementing the DCG-HEU-MDS algorithm are presented. In addition to the columns in Table 5.1 and Table 5.2, Table 5.3 includes

Table 5.3. Computation results of the DCG-HEU-MDS Algorithm

1 group, 10 targets													
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	MDS BS	CPU	Vars	SP Cuts	MDS
70%	60.999	59.274	31.98	33120	118.16	38.68	339	5810	12	59.71	756	10632	54
75%	65.356	63.947	26.62	31591	112.70	19.06	170	2623	3	32.78	328	4793	21
80%	69.713	69.371	20.39	29832	106.43	2.57	61	507	2	18.04	187	2733	9
85%	74.070	69.371	20.39	29832	106.43	4.28	70	791	4	14.06	187	2543	33
90%	78.427	69.371	20.39	29832	106.43	2.11	56	502	2	17.62	232	3262	44
95%	82.784	69.371	20.39	29832	106.43	9.32	67	1036	14	15.17	158	2298	35
100%	87.141	87.141	0.00	28031	100.00	0.51	28	158	0	2.47	35	321	0
1 group, 15 targets													
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	MDS BS	CPU	Vars	SP Cuts	MDS
70%	63.116	60.007	33.45	34463	122.32	7290.13	318	20149	3	Time	433	34670	5
75%	67.625	67.577	25.05	31971	113.48	5480.87	239	14768	2	Time	813	67450	44
80%	72.133	70.529	21.78	30118	106.90	17.74	133	1913	3	Time	406	30350	18
85%	76.641	75.557	16.20	29231	103.75	2703.55	206	9404	3	7076.17	458	28010	25
90%	81.150	75.557	16.20	29231	103.75	175.28	149	3194	2	6235.78	797	37466	52
95%	85.658	75.557	16.20	29231	103.75	210.96	133	2610	5	1468.23	478	17066	50
100%	90.166	90.166	0.00	28174	100.00	36.47	94	1389	0	107.29	108	2337	0
1 group, 20 targets													
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	MDS BS	CPU	Vars	SP Cuts	MDS
70%	60.945	-	NA	-	NA	-	-	-	-	Time	197	7787	3
75%	65.298	-	NA	-	NA	-	-	-	-	Time	190	9593	1
80%	69.651	-	NA	-	NA	-	-	-	-	Time	271	13069	1
85%	74.004	-	NA	-	NA	-	-	-	-	Time	241	12709	2
90%	78.357	-	NA	-	NA	-	-	-	-	Time	175	26002	2
95%	82.711	79.747	NA	36624	NA	27022.10	233	12587	3	Time	282	15266	3
100%	87.064	87.064	NA	32118	NA	1298.28	110	2763	0	3008.18	129	4558	0

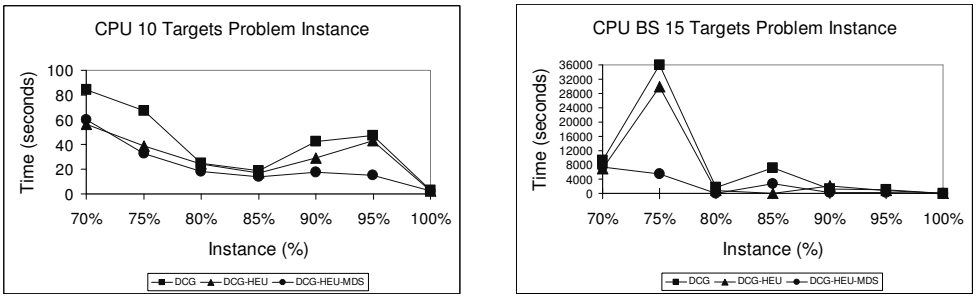

Figure 5.8. Computational times for 10, and 15 targets problem instances.

columns for the total number of MDS constraints generated until the best solution is found, *MDS BS*, and the total number of MDS constraints generated until the algorithm terminates, *MDS*. Generally, the results for the 10-target problem instance are the same as in Table 5.1 and Table 5.2. However, for the 15-target instance, there are four out of seven cases that were optimally solved, which is higher than those of the DCG algorithm and the DCG-HEU algorithm. Although the total number of cases with feasible solutions in the 20-target problem instance are less than those of the DCG-HEU algorithm, the quality of the solution of the 95% case of the DCG-HEU-MDS algorithm is much better than that of the DCG-HEU algorithm. To effectively compare the proposed algorithms, we plot their computational times in Figure 5.8. We ignore the computational time of the 20-target problem instance since they are insignificant. We consider the computational times until termination, *CPU*, for the 10-target problem instance since all of its cases were optimally solved, and the computational times until the best solution was found for the 15-target problem instance since not all of its cases were optimally solved. Considering *CPU* of the 10-target problem instance, the DCG-HEU-MDS algorithm performed better than the DCG algorithm and the DCG-HEU algorithm. This is because the DCG-HEU-MDS algorithm includes the heuristic route generation step and cut generation step. Both steps contribute in saving a lot of computational time for the DCG-HEU-MDS algorithm. The simple path heuristic prevents the DCG-HEU-MDS algorithm from being stuck in the column generation step looking for the best variable by generating a variable

with acceptable quality to be added to the RMP. The cut generation step generates MDS constraints that eliminate fractional RMP solutions. The MDS constraints encourage integrality and hence the algorithm should quickly converge. Consequently, the DCG-HEU algorithm also performed better than the DCG algorithm because of the simple path heuristic. Finally, consider the *CPU BS* of the 15 targets problem instance. Except the in 85% case, the DCG-HEU-MDS algorithm reaches the best found solutions faster than the DCG algorithm and the DCG-HEU algorithm. In the next section, we discuss a modification of the column generation step discussed in Section 5.2.1 and the computational experiments are presented.

5.4 Modified Column Generation

In this section, we discuss a modification of the column generation step in Section 5.2.1. As mentioned earlier, solving the MCNF problem usually obtains a minimum cost path that includes negative cost cycles. To overcome this difficulty, in Section 5.2.1, we suggested dynamically adding walk elimination constraints as in (5.21) to eliminate all invalid walks in the minimum cost path. The invalid walks include simple paths with a positive cost and negative cost cycles. Recall that the cost of an invalid walk ψ, which is $\sum_{l\in\psi}(\alpha_l+\beta_l\rho_w^*-\pi_k^*)y_l$, depends upon the RMP solution (x^*,π^*,ρ^*). In each iteration when the column generation step is invoked, the RMP solution is different. This implies that walk elimination constraints added in each iteration are local constraints and that they are only valid within the iteration in which they were generated. Consequently, walk elimination constraints were disregarded every time the column generation step is terminated. The implementation of the DCG algorithm, the DCG-HEU algorithm, and the DCG-HEU-MDS algorithm in Section 5.3 follows the procedure that all walk elimination constraints added to the MCNF problem were disregarded when the column generation step is terminated.

In this section, we modify the procedure suggested in Section 5.2.1 to reduce the computational time in the column generation step. Consider a walk elimination constraint that was generated to eliminate a negative cost cycle ψ_1 from a previous iteration with (x^*, π^*, ρ^*). When a new variable with negative cost is needed, the column generation is again invoked with a different RMP solution, $(\tilde{x}, \tilde{\pi}, \tilde{\rho})$. With the RMP solution $(\tilde{x}, \tilde{\pi}, \tilde{\rho})$, the cycle ψ_1 can have positive, negative, or zero cost. If the cycle ψ_1 has either a positive or zero cost, the cycle ψ_1 will not be part of a solution of the MCNF problem. However, if the cycle ψ_1 has a negative cost, the walk elimination constraint added in the previous iteration is still a cut to eliminate the cycle ψ_1 and can be reused. This implies that even with a different RMP solution, (x^*, π^*, ρ^*), walk elimination constraints that eliminate negative cost cycles can be considered as global constraints. By contrast, a positive cost simple path from the previous iteration may have a negative cost with a new RMP solution. Hence a walk elimination constraint that eliminates a positive cost simple path is not necessarily a valid inequality for other RMP solutions. For the modified column generation step, we suggest that a walk elimination constraint is added to the MCNF problem only to eliminate a negative cost cycle. The walk elimination constraint is then kept as a part of the MCNF problem, while a positive cost simple path is ignored since its cut will not be valid in the next iteration.

$$\min \sum_{l \in L} \{\alpha_l + \beta_l \rho_w^* - \pi_k^*\} y_l, \tag{5.22}$$

$$\text{s.t.} \sum_{l \in \delta_{v_i}^+} y_l - \sum_{l \in \delta_{v_i}^-} y_l = b(v_i) \qquad \forall v_i \in V, \tag{5.23}$$

$$\sum_{l \in \delta_{v_i}^+} y_l \leq 1 \qquad \forall v_i \in V, \tag{5.24}$$

$$\sum_{l \in \psi} y_l \leq |\Omega(\psi)| \qquad \forall \psi \in \bar{\Psi}, \tag{5.25}$$

$$y_l \in \{0, 1\} \qquad \forall l \in L, \tag{5.26}$$

The MCNF problem for the modified column generation step becomes (5.22)-(5.26), where $\bar{\Psi}$ is a subset cycles. Obviously, the MCNF problem become larger as the process continues because of walk elimination constraints in (5.25). To compare the column generation step in Section 5.2.1 and the column generation step suggested in this section, it is equivalent to a comparison between solving a smaller MCNF problem and regenerate a lot of walk elimination constraints, and solving a larger MCNF problem but generate *new* walk elimination constraints only when new negative cost cycles are found. However, the difficulty of solving the MCNF problem is not exponentially increase as the problem size increases. Therefore, if there are a lot of negative cost cycles embedded in an instance graph G, the modified column generation step is likely to be better than the column generation step in Section 5.2.1. Intuitively, in addition to the modified column generation step, it might be a better idea to add a walk elimination constraint to eliminate a positive cost simple path and removed at the end of each iteration. However, any improvements from doing so is overwhelmed by the amount of bookkeeping effort.

We implemented the BCP methodology using the modified column generation to the same problem instances as in Section 5.3. The 10-target, 15-target, and 20-target problem instances were solved using the three algorithms but now with a suffix, -M, to indicate the modification of the column generation, which are the DCG-M algorithm, the DCG-HEU-M algorithm, and the DCG-HEU-MDS-M algorithm.

The results from implementing the DCG-M algorithm are shown in Table 5.4. In the 10-target problem instance, all of its cases were optimally solved. A major improvement of the DCG-M algorithm is that all cases of the 15-target problem instance were optimally solved. This is from the fact that only two, three, and four out of seven cases were optimally solved in the DCG algorithm, the DCG-HEU algorithm, and the DCG-HEU-MDS algorithm, respectively. However, for the 20-target problem instance, the DCG-M algorithm struggled and provided similar results as from the

Table 5.4. Computation results of the DCG-M Algorithm

1 group, 10 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.999	59.274	31.98	33120	118.16	22.49	360	496	56.93	874	1065
75%	65.356	61.907	28.96	31591	112.70	14.93	179	336	22.08	309	479
80%	69.713	69.371	20.39	29832	106.43	1.55	30	95	10.50	192	314
85%	74.070	69.371	20.39	29832	106.43	2.26	42	119	7.34	167	267
90%	78.427	69.371	20.39	29832	106.43	10.87	134	261	21.84	345	539
95%	82.784	69.371	20.39	29832	106.43	16.55	161	331	47.47	672	926
100%	87.141	87.141	0.00	28031	100.00	1.23	29	91	2.65	41	124
1 group, 15 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	63.116	60.007	33.45	34463	122.32	6954.10	139	2449	19743.32	1601	5645
75%	67.625	67.577	25.05	31971	113.48	13929.82	303	2831	31647.89	1823	6495
80%	72.133	70.529	21.78	30118	106.90	1208.83	83	1864	6183.92	398	2907
85%	76.641	75.557	16.20	29231	103.75	2468.13	104	1878	8392.52	423	2908
90%	81.150	75.557	16.20	29231	103.75	1200.89	98	1870	18974.2	568	3827
95%	85.658	75.557	16.20	29231	103.75	4269.04	242	2691	10086.72	803	3855
100%	90.166	90.166	0.00	28174	100.00	73.33	38	515	102.58	76	583
1 group, 20 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.945	-	NA	-	NA	-	-	-	Time	7	2282
75%	65.298	-	NA	-	NA	-	-	-	Time	7	2282
80%	69.651	-	NA	-	NA	-	-	-	Time	7	2282
85%	74.004	-	NA	-	NA	-	-	-	Time	7	2282
90%	78.357	-	NA	-	NA	-	-	-	Time	7	2282
95%	82.711	-	NA	-	NA	-	-	-	Time	7	2282
100%	87.064	87.064	NA	32118	100	258.05	100	842	1103.96	142	967

DCG algorithm. The size of the problem and the cycles embedded in the instance graph G of the 20-target problem instance are too complex even with the DCG-M algorithm. In the 10-target, and the 15-target problem instances, the total number of subproblem cuts, *SP Cuts BS* and *SP Cuts*, reduced tremendously from that of the DCG algorithm. This is because walk elimination constraints are kept in the MCNF problem and generate new ones only when they are needed. Keeping the walk elimination constraints in the MCNF problem is a benefit. The results for using the DCG-HEU-M algorithm and the DCG-HEU-MDS-M algorithm are similar to the previous conclusions and are shown in Table 5.5 and Table 5.6, respectively.

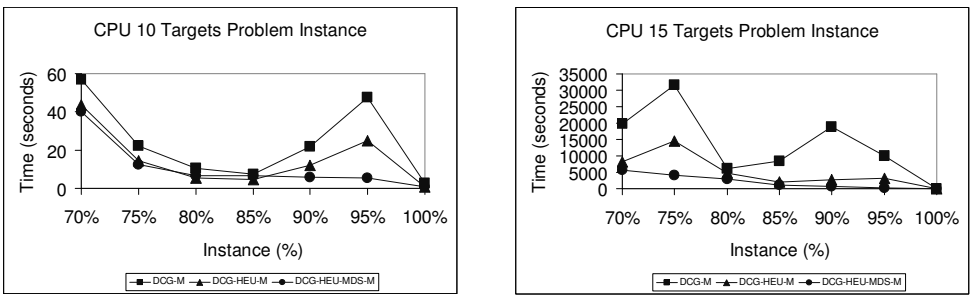

Figure 5.9. Computational times for 10, and 15 targets problem instances with the modified column generation.

Table 5.5. Computation results of the DCG-HEU-M Algorithm

1 group, 10 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.999	59.274	31.98	33120	118.16	18.69	381	438	43.48	763	853
75%	65.356	63.947	26.62	31591	112.70	6.36	150	189	14.44	298	349
80%	69.713	69.371	20.39	29832	106.43	1.76	61	86	5.53	157	187
85%	74.070	69.371	20.39	29832	106.43	0.99	51	78	4.52	142	172
90%	78.427	69.371	20.39	29832	106.43	5.74	116	151	11.95	310	350
95%	82.784	69.371	20.39	29832	106.43	6.43	161	199	24.96	616	673
100%	87.141	87.141	0.00	28031	100.00	0.52	30	45	0.76	38	54
1 group, 15 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	63.116	60.007	33.45	34463	122.32	2967.39	723	2317	8124.88	1555	3825
75%	67.625	67.577	25.05	31971	113.48	6786.42	297	1728	14636.41	1648	4333
80%	72.133	70.529	21.78	30118	106.90	49.51	159	382	4710.43	490	2349
85%	76.641	75.557	16.20	29231	103.75	133.41	198	592	2039.56	484	1932
90%	81.150	75.557	16.20	29231	103.75	150.86	201	653	2833.36	601	2107
95%	85.658	75.557	16.20	29231	103.75	458.25	210	858	3102.84	669	2129
100%	90.166	90.166	0.00	28174	100.00	2.95	40	83	60.82	75	215
1 group, 20 targets											
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	CPU	Vars	SP Cuts
70%	60.945	-	NA	-	NA	-	-	-	Time	181	1095
75%	65.298	64.098	NA	59508	NA	79.06	92	239	Time	206	1161
80%	69.651	-	NA	-	NA	-	-	-	Time	225	1325
85%	74.004	-	NA	-	NA	-	-	-	Time	255	1032
90%	78.357	-	NA	-	NA	-	-	-	Time	180	957
95%	82.711	-	NA	-	NA	-	-	-	Time	231	1291
100%	87.064	87.064	NA	32118	NA	376.60	104	268	872.28	139	452

Figure 5.9 shows the computational time for the 10-target and the 15-target instances. Since all cases of both problem instances were optimally solved, we plot the computational time until algorithms terminate, *CPU*. In most cases, except the 85% and the 80% cases of the 10-target problem instance, the DCG-HEU-MDS-M algorithm performed better than the other two algorithms. In the 10-target problem instance, all of its cases can be optimally solved within one minute, and five of those cases can be optimally solved within 10 seconds by the DCG-HEU-MDS-M algorithm. The reduction of computational times was impressive, especially in the 15-target problem instance. The computational time in the 95% case decreased from almost three hours in the DCG-M algorithm to about five minutes in the DCG-HEU-MDS-M algorithm. The computational time in the 90% case decreased from more than five hours in the DCG-M algorithm to about 13 minutes in the DCG-HEU-MDS-M algorithm. Finally, the computational time in the 75% case decreased from almost nine hours in the DCG-M algorithm to about one hour in the DCG-HEU-MDS-M algorithm. These show remarkable improvements of the DCG-HEU-MDS-M algorithm over the other two algorithms. In order to directly compare a development of the modified column

Table 5.6. Computation results of the DCG-HEU-MDS-M Algorithm

1 group, 10 targets													
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	MDS BS	CPU	Vars	SP Cuts	MDS
70%	60.999	59.274	31.98	33120	118.16	14.23	332	381	10	40.30	752	824	58
75%	65.356	63.947	26.62	31591	112.70	5.94	140	177	3	12.46	283	325	19
80%	69.713	69.371	20.39	29832	106.43	1.28	56	80	2	6.80	176	211	9
85%	74.070	69.371	20.39	29832	106.43	2.07	64	93	4	6.80	166	199	30
90%	78.427	69.371	20.39	29832	106.43	1.39	57	84	3	5.86	176	205	42
95%	82.784	69.371	20.39	29832	106.43	1.40	59	86	8	5.33	152	184	37
100%	87.141	87.141	0.00	28031	100.00	0.40	30	45	0	0.63	38	54	0
1 group, 15 targets													
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	MDS BS	CPU	Vars	SP Cuts	MDS
70%	63.116	60.007	33.45	34463	122.32	3246.88	263	1369	3	5573.56	887	2622	98
75%	67.625	67.577	25.05	31971	113.48	896.59	211	1084	6	4184.06	777	2376	88
80%	72.133	70.529	21.78	30118	106.90	104.83	162	429	5	2901.75	417	1972	39
85%	76.641	75.557	16.20	29231	103.75	615.92	134	780	1	1098.21	388	1245	30
90%	81.150	75.557	16.20	29231	103.75	376.56	144	481	3	786.52	470	1077	90
95%	85.658	75.557	16.20	29231	103.75	11.16	109	237	1	321.85	456	972	64
100%	90.166	90.166	0.00	28174	100.00	2.94	40	83	0	61.09	75	215	0
1 group, 20 targets													
d	MAX SD	SD BS	SD(%)	BS	BS(%)	CPU BS	Vars BS	SP Cuts BS	MDS BS	CPU	Vars	SP Cuts	MDS
70%	60.945	-	NA	-	NA	-	-	-	-	Time	187	705	3
75%	65.298	62.996	NA	52345	NA	365.28	118	319	1	Time	171	885	1
80%	69.651	-	NA	-	NA	-	-	-	-	Time	209	853	1
85%	74.004	71.643	NA	58440	NA	84.78	95	210	2	Time	187	1016	4
90%	78.357	67.762	NA	54314	NA	21.83	103	210	1	Time	176	916	1
95%	82.711	81.884	NA	42649	NA	170.96	119	280	1	Time	235	1066	3
100%	87.064	87.064	NA	32118	NA	378.73	104	268	0	880.13	139	452	0

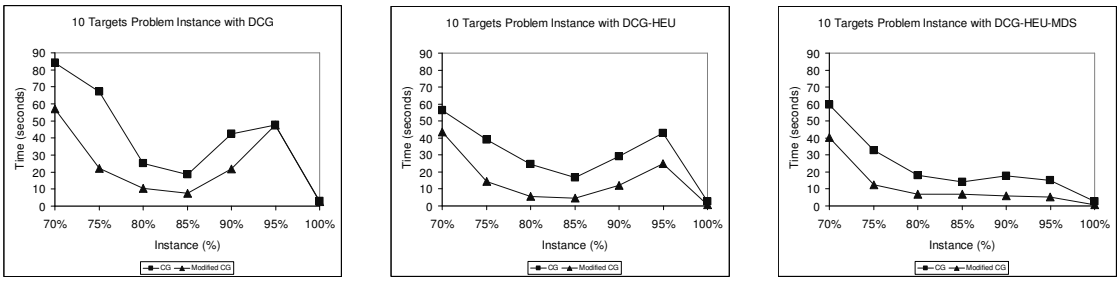

Figure 5.10. Comparisons of the computation times between the original and the modified column generations in the 10-target problem instance.

generation and the column generation in Section 5.2.1, we show their computational times until the algorithms terminated, *CPU*, in Figure 5.10.

Note that the figure shows only the computational times of the 10-target problem instance, since not all cases the 15-target and the 20-target problem instances were optimally solved. It is obvious from Figure 5.10 that every algorithm performed better when implemented with the modified column generation in all cases.

CHAPTER 6

CONCLUSION AND FUTURE RESEARCH

In this research, we presented a study on the UAV routing problem with limited risk (URPR). The considered risk in this study is the variation of the fuel burn caused by wind variation. Due to UAV size and speed, changes in wind can significantly affect the amount of fuel burn for a route. A feasible route at the beginning of an operation can become infeasible because of an unexpected stronger headwind or a lesser tailwind. To discuss this challenge, we modeled the URPR as a set-partitioning problem with a quadratic constraint (SPQC). The quadratic constraint, which is usually ignored in the literature, controls variations in a UAV operation to a predetermined amount. The SPQC model is very difficult to solve because of the enormous number of possible variables and the additional quadratic variance constraint. We constructed a variance model to quantify the variation of the expected fuel burn. However, given a route starts and ends at the same depot, we can show that the quadratic constraint is reducible to a linear knapsack constraint. This improvement greatly reduces computational efforts required to solve the URPR since the problem is simplified to a linear integer program. Although the URPR is very similar to the VRP, most research on the URPR focuses on heuristic and simulation techniques that quickly generate a static route that ignores variability in UAV operations. In this research, we developed and implemented exact solution methods to solve the URPR based upon branch-and-cut-and-price methodology. The considered URPR includes both the URPR with time windows (URPRTW) and the URPR without time windows (URPR).

The methodology includes a column generation step that generates variables to the RMP and a cut generation step that generates a new class of valid inequalities

called minimum dependent set constraints (MDS) derived from the linear variance constraint. The MDS constraint is proved to cut off fractional solutions that the linear variance constraint does not. Furthermore, the coefficients of MDS constraints are positive integers, which should encourage integrality of the URPR solution. We conducted computational experiments on both the URPR and the URPRTW. The results showed that the methodology is very efficient as an exact solution method to the URPR and the URPRTW. The methodology was able to optimally solve medium-sized URPRTW instances and small-sized URPR instances in seconds. Since the MDS constraints are not essential to the URPR, they were included only to encourage integrality. We implemented the methodology both with and without the MDS constraint to illustrate effectiveness of the MDS constraint. Although the efficiency of the methodology was not improve much by adding the MDS constraint in the URPRTW, adding the MDS constraints greatly improve the computational time in solving the URPR. This is expected since the absence of time window requirements increases the number of variables in the URPR. Implementing the MDS constraints encourages integrality and hence improves the overall performance of the methodology. Moreover, for the URPR, we developed a simple path heuristic that generates simple paths from a negative cost cycle. Heuristics also have a positive effect on the efficiency of the methodology. Since without the heuristics, the column generation step has to generate cycle elimination constraints to eliminate negative cost cycles until a valid simple path is found. This process is usually time consuming and very inefficient for a larger problem.

Finally, for future extensions of this study, we will consider implementing a lifting process to the MDS constraints. The lifting technique that lifts an MDS constraint into a higher dimensional space will tighten the constraint. We believe that a tighter constraint will have a positive effect on the computational effort to solve the URPR. We also would like to relax the assumptions that we made to simplify

the URPR. Without the assumptions, the URPR becomes more realistic. However a more powerful algorithm is required. Finally, we would like to generalize our solution method to other applications.

REFERENCES

[1] [Online]. Available: www.wired.com/science/discoveries/news/2003/11/

[2] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, pp. 80–91, 1959.

[3] G. Laporte and I. H. Osman, "Routing problems: A bibliography," *Annals of Operations Research*, vol. 6, pp. 227–262, 1995.

[4] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, pp. 345–358, 1992.

[5] G. Laporte and Y. Nobert, "Exact algorithms for the vehicle routing problem," in *Surveys in Combinatorial Optimization*, S. Martello, G. Laporte, M. Minoux, and C. Ribeiro, Eds. Amsterdam: North-Holland, 1987, pp. 147–184.

[6] G. Laporte, H. Mercure, and Y. Nobert, "An exact algorithm for the asymmetrical capacitated vehicle routing problem," *Networks*, vol. 16, pp. 33–46, 1986.

[7] N. Christofides, A. Mingozzi, and P. Toth, "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations," *Mathematical Programming*, vol. 20, pp. 255–282, 1981.

[8] S. Eilon, C. D. T. Watson-Gandy, and N. Christofides, *Distribution Management: Mathematical Modelling and Practical Analysis*. London: Griffin.

[9] N. Christofides, "Vehicle scheduling and routing," *Presented at the 12th International Symposium on Mathematical Programming, Cambridge, MA.*, 1985.

[10] M. Balinski and R. Quandt, "On an integer program for a delivery problem," *Operations Research*, vol. 12, pp. 300–304, 1964.

[11] B. Foster and D. Ryan, "An integer programming approach to the vehicle scheduling problem," *Operatons Research Quarterly*, vol. 27, pp. 367–384, 1976.

[12] Y. Agarwal, K. Mathur, and H. M. Salkin, "A set-partitioning-based algorithm for the vehicle routing problem," *Networks*, vol. 19, pp. 731–750, 1989.

[13] J. Desrosiers, F. Soumis, and M. Desrochers, "Routing with time windows by column generation," *Networks*, vol. 14, pp. 545–565, 1984.

[14] M. Desrochers, J. K. Lenstra, and M. W. P. Savelsbergh, "A classification scheme for vehicle routing and scheduling problems," *European Journal of Operation Research*, vol. 46, pp. 322–332, 1990.

[15] A. S. Alfa, S. S. Heragu, and M. Chen, "A 3-opt based simulated annealing algorithm for the vehicle routing problem," *Computer and Industrial Engineering*, vol. 21, pp. 635–639, 1991.

[16] B. L. Golden and C. C. Skiscim, "Using simulated annealing to solve routing and location problems," *Naval Research Logistics Quaterly*, vol. 33, pp. 261–279, 1986.

[17] D. T. Hiquebran, A. S. Alfa, J. A. Shapiro, and D. H. Gittoes, "A revised simulated annealing and cluster-1st route-2nd algorithm applied to the vehicle-routing problem," *Engineering Optimization*, vol. 22, pp. 77–107, 1994.

[18] I. H. Osman, "Metastrategy simulated annealing and tabu search algorithms for combinatorial optimization problems," *Ph.D. Thesis, Imperial College, The Management School, University of London*, 1991.

[19] ——, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of Operations Research*, vol. 41, pp. 421–451, 1993.

[20] G. M. Buxey, "The vehicle scheduling problem and monte-carlo simulation," *Journal of Operation Research Society*, vol. 30, pp. 563–573, 1979.

[21] M. Gendreau, A. Hertz, and G. Laporte, "A tabu search heuristic for the vehicle routing problem," *Management Sciences*, vol. 40, pp. 1276–1290, 1994.

[22] V. M. Pureza and P. M. Franca, "Vehicle routing problems via tabu search metaheuristic," *Publication CRT-747, Centre de recherche sur les transports, Montreal*, 1991.

[23] G. A. P. Kinderwater and M. W. P. Savelsbergh, *Local search in physical distribution in: Local Search Algorithm*, J. K. Lenstra and E. H. L. Aarts, Eds. Chichester: Wiley, 1996.

[24] M. W. P. Savelsbergh, "An efficient implementation of local search algorithm for constrained routing problems," *European Journal of Operational Research*, vol. 47, pp. 75–85, 1990.

[25] P. M. Thompson, "Local search algorithms for vehicle routing and other combinatorial problems," *Ph.D. Thesis, Massachusetts Institute of technology*, 1988.

[26] H. Kopfer, G. Pankratz, and E. Erkens, "Development of a hybrid genetic algorithm for vehicle-routing," *Operations Research Spektrum*, vol. 16, p. 21, 1994.

[27] M. W. P. Savelsbergh, "Local search in routing problems with time windows," *Annals of Operations Research*, vol. 4, pp. 285–305, 1985.

[28] J. F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis, "The vrp with time windows," *Society for Industrial and Applied Mathematics*, pp. 157–193, 2001.

[29] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problem with time windows constraints," *Operations Research*, vol. 35, pp. 254–265, 1987.

[30] ——, "On the worst-case performance of some heurisitics for the vehicle routing and scheduling problem with time window constraints," *Networks*, vol. 16, pp. 161–174, 1986.

[31] J. Y. Potvin and J. M. Rousseau, "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows," *European Journal of Operation Research*, vol. 66, pp. 331–340, 1993.

[32] E. Baker and J. Schaffer, "Computational experience with branch exchange heuristic for vehicle routing problems with time window constraints," *American Journal of Mathematical and Management Sciences*, vol. 6, pp. 261–300, 1986.

[33] R. A. Russell, "An effective heuristic for the m-tour traveling salesman problem with some side conditions," *Operations Research*, vol. 25, pp. 517–524, 1977.

[34] M. W. P. Savelsbergh, "The vehicle routing problem with time windows: Minimizing route duration," *ORSA Journal on Computing*, vol. 4, pp. 146–154, 1992.

[35] G. A. P. Kinderwater and M. W. P. Savelsbergh, *Local search in combinatorial optimization*, J. K. Lenstra and E. H. L. Aarts, Eds. Chichester: Wiley, 1997.

[36] G. Kontoravdis and J. F. Bard, "A grasp for the vehicle routing problem with time windows," *ORSA Journal on Computing*, vol. 7, pp. 10–23, 1995.

[37] R. A. Russell, "Hybrid heuristics for the vehicle routing problem with time windows," *Transportation Science*, vol. 29, pp. 156–166, 1995.

[38] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Y. Potvin, "A tabu search heuristic for the vehicle routing problem with soft time windows," *Transportation Science*, vol. 31, pp. 170–186, 1997.

[39] Y. Rochat and E. Taillard, "Probabilistic diversification and intensification in local search for vehicle routing," *Journal of Heuristics*, vol. 1, pp. 147–167, 1995.

[40] E. Taillard, "Parallel iterative search methods for vehicle routing problems," *Networks*, vol. 23, pp. 661–673, 1993.

[41] J. Homberger and H. Gehring, "Two evolutionary metaheuristics for the vehicle routing problem with time windows," *INFOR*, vol. 37, pp. 297–318, 1999.

[42] J. Y. Potvin and S. Bengio, "The vehicle routing problem with time windows - part ii: Genetic search," *INFORMS Journal on Computing*, vol. 8, pp. 165–172, 1996.

[43] J. L. Blanton and R. L. Wainwright, "Multiple vehicle routing with time and capacity constraints using genetic algorithms," *In Proceeding of the 5th International Conference on Genetic Algorithms, San Mateo, CA.*, pp. 452–459, 1993.

[44] G. Laporte and F. V. Louveaux, "The integer l-shape method for stochastic integer problems with complete recourse," *Operations Research Letters*, vol. 13, pp. 133–142, 1993.

[45] W. R. Stewart and B. L. Golden, "Stochastic vehicle routing: A comprehensive approach," *European Journal of Operation Research*, vol. 14, pp. 371–385, 1983.

[46] G. Laporte, F. V. Louveaux, and H. Mercure, "Models and exact solutions for a class of stochastic location-routing problems," *European Journal of Operational Research*, vol. 39, pp. 71–78, 1989.

[47] H. Vladimirou and S. Zenios, "Stochastic linear programs with restricted recourse," *European Journal of Operational Research*, vol. 101, pp. 177–192, 1997.

[48] D. P. Morton and R. K. Wood, "Restricted-recourse bounds for stochastic linear programming," *Operations Research*, vol. 47, pp. 943–956, 1999.

[49] P. Beraldi, L. Grandinetti, R. Musmanno, and C. Triki, "Parallel algorithms to solve two-stage stochastic linear programs with robustness constraints," *Parallel Computing*, vol. 26, pp. 1189–1908, 2000.

[50] H. D. Sherali and B. Fraticelli, "A modification of benders' decomposition algorithm for discrete subproblems: An approach for stochastic programs with integer recourse," *Journal of Global Optimization*, vol. 22, pp. 319–342, 2002.

[51] D. J. Bertsimas, "A vehicle routing problem with stochastic demand," *Operations Research*, vol. 40, pp. 574–585, 1992.

[52] D. J. Bertsimas, P. Jaillet, and A. R. Odoni, "A priori optimization," *Operations Research*, vol. 38, pp. 1019–1033, 1990.

[53] M. Gendreau, G. Laporte, and R. Seguin, "Stochastic vehicle routing," *European Journal of Operational Research*, vol. 88, pp. 3–12, 1996.

[54] M. R. Sisson, "Applying tabu heuristic to wind influenced, minimum risk and maximum expected coverage route," in *MS Thesis*, February 1997.

[55] K. P. O'Rourke, T. G. Bailey, R. R. Hill, and W. B. Carltons, "Dynamic routing of unmanned aerial vehicles using reactive tabu search," *Military Operations Research: A Journal of the Military Operations Research Society*, vol. 6, no. 1, pp. 5–30, 2001.

[56] J. L. Ryan, "Embedding a reactive tabu search heuristic in unmanned aerial vehicle simulation," *MS Thesis, Air Force Institute of Technology*, February 1998.

[57] M. A. Russell and G. B. Lamont, "A genetic algorithm for unmanned aerial vehicle routing," *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 1523–1530, 2005.

[58] J. J. Corner and G. B. Lamont, "Parallel simulation of uav swarm senarios," *Proceeding of the 2004 Winter Simulation Conference*, pp. 355–363, 2004.

[59] V. K. Shetty, M. Sudit, and R. Nagi, "Priority-based assignment and routing of a fleet of unmanned combat aerial vehicles," *Computers and Operations Research*, vol. 35, no. 6, pp. 1813–1828, 2008.

[60] S. Rathinam and R. Sengupta, "Lower and upper bounds for a multiple depot uav routing problem," *Proceeding of the 45th IEEE Conference on Decision and Control*, pp. 5287–5292, 2006.

[61] P. Chandler and M. Pachter, "Research issues in autonomous control of tactical uavs," *American Control Conference*, pp. 394–398, 1998.

[62] P. Chandler, S. Rasmussen, and M. Pachter, "Uav cooperative path planning," *Proceedings of the GNC*, pp. 1255–1265, 2000.

[63] P. Chandler and M. Pachter, "Hierarchical control of autonomous control of tactical uavs," *Proceedings of GNC*, pp. 632–642, 2001.

[64] P. Chandler, S. Rasmussen, and M. Pachter, "Uav cooperative control," *American Control Conference*, 2001.

[65] T. Maddula, A. A. Minai, and M. M. Polycarpou, *Multi-target assignment and path planning for groups of UAVs*, S. Butenko, R. Murphey, and P. Pardalos, Eds. Kluwer Academic Publishers, December 2002.

[66] J. Bellingham, M. Tellerson, A. Richards, and J. P. How, "Multi-task allocation and trajectory design for cooperating uavs," *Cooperative Control: Models, Applications and Algorithms at the Conference on Coordination, Control and Optimization*, November 2001.

[67] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "On the solution of traveling salesman problems," *Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung, International Congress of Mathematicians*, p. 645, 1998.

[68] M. Eso, D. L. Jensen, and L. Ladanyi, "Bid evaluation for fcc auction 31 using column generation," *Presentation at the INFORMS Annual Meeting, San Jose*, 2002.

[69] ——, "Solving lexicographic multiobjective mip with column generation," *Presentation at the INFORMS Annual Meeting, San Jose*, 2002.

[70] M. Grotschel, M. Junger, and G. Reinelt, "A cutting plane algorithm for the linear ordering problem," *Operations Research*, vol. 32, pp. 1195–1220, 1984.

[71] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale traveling salesman problems," *SIAM Review*, vol. 33, pp. 60–100, 1991.

[72] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for hugh integer programs," *Operations Research*, vol. 46, pp. 316–329, 1998.

[73] M. W. P. Savelsbergh, *Branch-and-price: Integer programming with column generation*, ser. In Encyclopedia of Optimization, C. Floudas and P. Pardalos, Eds., 2001.

[74] J. E. Kelley, "The cutting-plane method for solving convex program," *SIAM Journal of Applied Mathematics*, vol. 8, pp. 703–712, 1960.

[75] G. L. Nemhauser and L. A. Wolsey, *Interger and combinatorial optimization*. Wiley-Interscience, 1988.

[76] D. M. Ryan and B. A. Foster, *An integer programming approach to scheduling*, ser. Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling, A. Wren, Ed. North-Holland, Amsterdam, 1981.

[77] H. S. Hwang, S. Visoldilokpun, and J. M. Rosenberger, "A branch-and-cut-and-price method for ship scheduling with limited risk," *Transportation science*, vol. 42, no. 3, pp. 336–351, 2008.

BIOGRAPHICAL STATEMENT

Siriwat Visoldilokpun was born in 1979 in Bangkok, Thailand. He received his B.S. degree in Industrial and Manufacturing Systems Engineering from the Sirindhon International Institute of Technology at the Thammasart University in 1999, his M.S. degree in Industrial and Manufacturing Systems Engineering from the Colorado State University at Pueblo in 2002, and his Ph.D. in Industrial Engineering from the University of Texas at Arlington in 2008. During his study at UTA, he also worked as a research assistant for Dr. Rosenberger on several projects and as a teaching assistant for many faculty members at the IMSE department. His current research interest is in the areas of Operations Research, Optimization, and Integer Programming.