SCALABLE ALGORITHMS FOR TWO-STAGE STOCHASTIC PROGRAM OPTIMIZATIONS

BY

AKHIL LANGER

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Laxmikant V. Kalé
Professor Udatta Palekar
Professor David Padua
Professor Rob Rutenbar
Dr. Steven Baker, MITRE Corporation

Abstract

We develop scalable algorithms for two-stage stochastic program optimizations. We propose performance optimizations such as cut-window mechanism in Stage 1 and scenario clustering in Stage 2 of benders method for solving two-stage stochastic programs. A naïve implementation of benders method has slow convergence rate and does not scale well to large number of processors especially when the problem size is large. We propose two decomposition schemes namely the Split-and-Merge (SAM) method and the Lagrangean Decomposition and Merge (LDAM) method that significantly increase the convergence rate of Bender's decomposition. SAM method gives up to 74% reduction in solution time while also giving significantly higher parallel speedups as compared to the naïve benders method. LDAM method, on the other hand, has made it possible to solve otherwise intractable stochastic programs. When mixed-integer variables are present in the Stage 1 of the stochastic program, these methods do not scale very well because of the increased size of the Stage 1 bottleneck. Parallelization of stochastic integer programs pose very unique characteristics that make them very challenging to parallelize. We develop a Parallel Stochastic Integer Program Solver (PSIPS) that exploits nested parallelism by exploring the branch-and-bound tree vertices in parallel along with scenario parallelization. PSIPS has been shown to have high parallel efficiency of greater than 40% at 120 cores which is significantly greater than the parallel efficiency of state-of-the-art mixed-integer program solvers. We further provide a computational engine for many real-time and dynamic problems faced by US Air Mobility Command (AMC). We first propose a stochastic programming solution to the military aircraft allocation problem

with consideration for disaster management. Then, we study US AMC's dynamic mission replanning problem and propose a mathematical formulation that is computationally feasible and leads to significant savings in cost as compared to myopic and deterministic optimization. It is expected that this work will provide the springboard for more robust problem solving with HPC in many logistics and planning problems

To my teachers, friends and parents

List of Symbols

This work was partially supported by the following sources:

- **MITRE**. This research was partially supported by MITRE Research Agreement Number 81990 with UIUC.
- **CSE Resources**. The research used the parallel computing resources provided by the Computational Science and Engineering Program at the University of Illinois. The CSE computing resource, provided as part of the Taub cluster, is devoted to high performance computing in engineering and science.

Acknowledgments

Table of Contents

List of Figures x

List of Tables xii

CHAPTER 1 Introduction 1
1.1 Stochastic Optimization 2
1.2 Parallel Computing 6
1.3 Thesis Organization 7
1.4 Literature Review 8

CHAPTER 2 Performance Optimizations for Two-stage Stochastic Linear Program Optimizations 11
2.1 Introduction 11
2.2 Related Work 12
2.3 Model Formulation & Approach 14
2.4 Parallel Stochastic Linear Program Solver (PSLPS) Design 16
2.5 Optimizing Stage 1 19
2.6 Optimizing Stage 2 26
2.7 PSLPS Scalability 29
2.8 Summary 30

CHAPTER 3 Parallel Branch-and-Bound for Two-stage Stochastic Integer Programs 31
3.1 Introduction 32
3.2 Two-stage Stochastic Integer Optimization 33
3.3 Case Study: Military Aircraft Allocation 33
3.4 Prior Work 35
3.5 Parallelization Approach 36
3.6 Design Considerations 39
3.7 Parallel Programming Model 45
3.8 Design A: Each Branch-and-Bound (BnB) Vertex is an Isolated Two-Stage LP 47

3.9 Design B: BnB Vertices Share Cut Constraints, Tree Explorers and *Scenario Evaluators* . . . 49
3.10 Performance and Analysis of PSIPS . . . 52
3.11 Summary . . . 59

CHAPTER 4 Split-and-Merge Method for Accelerating Convergence of Stochastic Linear Programs . . . 61
4.1 Introduction . . . 61
4.2 Related Work . . . 62
4.3 Split and Merge Algorithm . . . 64
4.4 Parallel Design of SAM . . . 67
4.5 Experimental Setup . . . 68
4.6 Results . . . 69
4.7 Summary . . . 77

CHAPTER 5 Accelerating Two-stage Stochastic Linear Programs Using Lagrangean Decomposition . . . 78
5.1 Introduction . . . 78
5.2 Introduction to Lagrangean Decomposition . . . 79
5.3 jetAlloc: The Military Aircraft Allocation Problem . . . 82
5.4 Lagrangean Decomposition for Two-Stage Stochastic Problems . . . 82
5.5 Lagrangean Decomposition for Military Aircraft Allocation Problem . . . 90
5.6 Parallel Design of LDAM and Experimental Setup . . . 92
5.7 Results . . . 94
5.8 Summary . . . 98

CHAPTER 6 Stochastic Optimization for Military Aircraft Allocation with Consideration for Disaster Management . . . 99
6.1 Introduction . . . 100
6.2 Literature Review . . . 101
6.3 Stochastic Model for Aircraft Allocation Problem . . . 103
6.4 Results . . . 106
6.5 Summary . . . 112

CHAPTER 7 Responding to disruptive events at execution time: a stochastic integer programming approach to Dynamic Mission Replanning by the Air Mobility Command of the Department of Defense . . . 113
7.1 Introduction . . . 113
7.2 Related Work . . . 115
7.3 Terminology . . . 116
7.4 Dynamic Mission Replanning . . . 118
7.5 Model Features . . . 120
7.6 Modeling Approach . . . 123
7.7 Model Formulation . . . 127
7.8 Implementation Details . . . 141

7.9 Experimental Setup 142
7.10 Results 148

CHAPTER 8 Conclusion and Future Work 150

APPENDIX A Stochastic Formulation of Military Aircraft Allocation Problem . . . 152
A.1 Indices and Index Sets 152
A.2 Input Data 154
A.3 Variables 156
A.4 Stage 1 Linear Program 157
A.5 Stage 2 Linear Program 158

APPENDIX B A small example of Dynamic Mission Replanning (DMR) 160
B.1 Demands and chosen itineraries 160
B.2 A sample scenario 162

REFERENCES 166

List of Figures

1.1 Benders decomposition and multicut L-shaped method for two-stage stochastic programs. . . . 4
1.2 Schematic of a distributed system . . . 5

2.1 Solution time for solving extensive formulation of stochastic programs . . . 13
2.2 Parallel design for two-stage Bender's decomposition . . . 18
2.3 Stage 1 LP solve times with and without advanced start . . . 19
2.4 Stage 1 memory usage and Stage 1 LP solve times . . . 20
2.5 Impact of memory bandwidth on LP solve time . . . 21
2.6 Cut usage rate in Stage 1 . . . 22
2.7 Stage 1 LP solve times and memory usage with cut-window . . . 23
2.8 Number of rounds to convergence and time to solution with different cut-windows . . . 24
2.9 Comparison of different cut-window strategies . . . 25
2.10 Variation across identical runs of parallel Bender's program . . . 26
2.11 Improvement in Stage 2 solve times due to scenario clustering . . . 28

3.1 Benders decomposition for 2-stage stochastic integer programs . . . 34
3.2 Exploiting nested prallelism in stochastic integer programs . . . 37
3.3 Scalability limitations of parallel naïve Bender's method implementation . . . 38
3.4 Variability and unpredictability in LP solve times of stochastic programs . . . 40
3.5 Schematics of the two design variants of PSIPS . . . 46
3.6 Strong scaling and performance variability of PSIPS with different number of Stage 1 Tree Explorers . . . 51
3.7 Widely different BnB trees from two identically configured executions . . . 54
3.8 Diagnozing the slower performance of Design A as compared to Design B . . . 55
3.9 Strong scaling of PSIPS for various datasets . . . 58

4.1 Schematic of the Splt-and-Merge (SAM) method . . . 66
4.2 Parallel design for Split-and-Merge (SAM) method . . . 67
4.3 Scenario and cut activity for multicut Bender's method . . . 69
4.4 Scenario and cut activity for SAM method . . . 70
4.5 Schematic of the Split and Hierarchical Merge (SAHM) method . . . 72

4.6 Scenario and cut activity for SAHM method 72
4.7 Timeline view of parallel naïve Bender's method and SAHM method execution 73
4.8 Parallel scalability of SAHM and its comparison with naïve Bender's method 77

5.1 Stage 1 and Stage 2 solve time comparison 80
5.2 Structure of two-stage stochastic program suitable for LDAM method 84
5.3 LDAM approach applied to 30t military aircraft allocation problem 86
5.4 Parallel design of the bootstrap phase of the LDAM method 93
5.5 Timeline view of processors for the naïve Bender's and LDAM method . . . 95

6.1 Hedging against future uncertainty can play a significant cost-saving role. In this 6-scenario example with an imminent disaster, squares correspond to each scenario's cost using deterministic optimization of expected demands; the horizontal average is depicted as a dashed line. In contrast, the circles correspond to stochastic optimization costs for each scenario (with the dashed-line average). Stochastic optimization yields only slightly more costly solutions when actual demands are low (scenarios 1, 4, and 5), but are much less costly when demands are elevated due to the disaster (scenarios 2, 3, and 6). The average cost savings in this example is approximately 35 percent. 107
6.2 For randomly generated scenarios with no imminent disaster, hedging against future uncertainty usually leads to cost saving. In this 120-scenario example, the stochastic programming solution has a lower cost in all scenarios. The average cost savings in this example are 3 percent 108
6.3 Time to solution of the extensive formulation of the stochastic program for different number of scenarios. The extensive formulation ILP is solved with Gurobi parallel integer solver running on 12 processors. Total time is broken down in to 3 components: *Generation time* - time it takes to generate the extensive formulation ILP from individual scenarios, *Root Solve Time*- time taken by Gurobi to solve the root relaxation of the extensive formulation ILP, *Optimization time After Root Solve* - time after the root solve to obtain the integer solution that is within 1% of the optimal. Optimization of the problem with 240 scenarios was terminated after walltime limit of 14000 seconds. 110
6.4 Time to solution using our distributed solver for the aircraft allocation problem with 240 scenarios. 111

7.1 TACC duty officer 118
7.2 Different types of disruption events and scenario formation 122
7.3 An example of an itinerary and its recourses for a weather disruption event . 143
7.4 Simulation setup for comparing various optimization approaches for the Dynamic Mission Replanning Problem 147

B.1 A sample itinerary $\mathcal{I}$ 164
B.2 Itinerary $\mathcal{I}$'s recourses for the given scenario 165

List of Tables

2.1 Size of stochastic linear program datasets (each with 120 scenarios) 16

3.1 Size of Stage 1 IP and Stage 2 lp of stochastic integer program models . . . 34
3.2 Average Stage 1 Linear Program (LP) solve time comparison between Design A and Design B of PSIPS 53
3.3 Cumulative distribution of trials of Design B for a target minimum parallel efficiencies on 3t-120 (baseline: 3 cores) 57
3.4 Cumulative distribution of trials of Design B for a target minimum parallel efficiency on 5t-120 (baseline: 3 cores) 57

4.1 Stage 1 LP and Stage 2 LP sizes of the stochastic linear program datasets . . 68
4.2 Size of stochastic linear program datasets 68
4.3 Solution time of 8t120s model with Naïve Bender's and SAHM method with different number of split-phase rounds 74
4.4 Solution time of 8t1000s model with Naïve Bender's and SAHM method with different number of split-phase rounds 75
4.5 Solution time of 10t1000s model with Naïve Bender's and SAHM method with different number of split-phase rounds 76

5.1 Steps in lagrangean decomposition applied to a sample linear program 81
5.2 Size of Stage 1 and Stage 2 LP in the stochastic linear program datasets . . 94
5.3 Size of the stochastic linear program datasets 94
5.4 Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 8t120s dataset 96
5.5 Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 10t120s dataset 96
5.6 Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 12t120s dataset 97
5.7 Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 15t120s dataset 97
5.8 Stage 2 variables, constraints count and optimization times for various LDPs and the original Stage 2 LP 98

7.12 Comparison of various KPIs across different optimization approaches 149

B.1 A sameple set of demands to be carried by an execution schedule 160

List of Algorithms

1 The Scenario Clustering Approach 28

2 Split-and-Merge (SAM) 65

3 Pseudocode for generating Lagrangean Decomposed Problem (LDP) 91

4 Pseduocode for generating objective function of the subproblems 92

List of Symbols

LP	Linear Program
IP	Integer Program
ILP	Integer Linear Program
SLP	Stochastic Linear Program
SIP	Stochastic Integer Program
MIP	Mixed-Integer Program
BnB	Branch-and-Bound
PSIPS	Parallel Stochastic Integer Program Solver
PH	Progressive Hedging
SAM	Split And Merge
LDAM	Lagrangean Decompose And Merge
LDP	Lagrangean Decomposed Problem
NB	Naïve Benders
DoD	Department of Defense
AMC	Air Mobility Command
TACC	Tanker Airlift Control Center
DMR	Dynamic Mission Replanning
CES	Current Event Set

CuIS	Current Itinerary Set
CaIS	Candidate Itinerary Set
SIS	Selected Itinerary Set
MyOp	Myopic Optimization
DetOp	Deterministic Optimization
StOp	Stochastic Optimization

CHAPTER 1

Introduction

In many real world situations, future outcomes such as weather in agriculture, product demands in the manufacturing industry, stock prices for an investor, etc. are dependent on myriad different factors that cannot be deterministically predicted. However, resource allocation has to take place before the actual realization of these unknown parameters. When resource use has to be optimized under such conditions, the resulting problem is called stochastic optimization. Stochastic optimization provides a means of coping with the uncertainty inherent in real- world systems. Unlike deterministic programming, stochastic programming explicitly incorporates uncertain parameters by assuming a probabilistic distribution to make a more rational decision for optimal resource allocation. Stochastic optimization algorithms have applications in statistics, science, engineering, and business. Examples include making investment decisions in order to increase profit (financial modeling), transportation (planning and scheduling logistics), design-space exploration in product design, supply chain and scheduling, environmental and pollution control, economic dispatch and unit commitment problem for power supply, etc. There are other applications in agriculture, energy, telecommunications, military, medicine, water management etc.

1.1 Stochastic Optimization

Equation 1.1 gives a standard representation of a stochastic program.

$$\begin{aligned} \min \quad & cx + \sum_s p_s(q_s y_s) \\ s.t. \quad & Ax \leq b \\ \forall s, \quad & W_s y_s + T_s x \leq h_s \end{aligned} \tag{1.1}$$

where, x corresponds to the strategic decisions corresponding to the known parameters that are to be taken now, and y_s corresponds to the operational decisions that will be taken when the scenario s is realized, and p_s is the probability that scenario s will occur. The objective function is sum of the costs of strategic decisions and the weighted average of the cost of operational decisions for all scenarios.

In multi-stage stochastic programs, decisions are made in multiple stages. For example, in portfolio management, a fixed amount of cash available at time t_0 has to be invested across times $t_1, t_2, ..., t_n$. Decisions taken at time t_i will depend on the decisions/outcomes from time t_{i-1}. Unlike the case of portfolio management in which the unknown parameters are realized over a sequence of stages, in two-stage stochastic programs, all the unknown parameters are realized in a single stage (as there are only two stages). In the first stage, strategic decisions are made (the known resources are allocated to the different fields of activities) and in the second stage operational decisions are made for every scenario. A specific instantiation of the unknown parameters is called a scenario. Most applications can be formulated as two-stage programs. We propose our work on two-stage stochastic programs but the strategy is easily generalizable to multi-stage stochastic programs. Moreover, multistage stochastic programs can be solved as a sequence of two-stage stochastic programs. Equation 1.2 and 1.3 shows the first and second stage programs of the two-stage stochastic program, respectively.

Stage 1 Program:

$$\begin{aligned} \min \quad & cx + \sum_s p_s Q_s(x, y_s) \\ s.t. \quad & Ax \leq b \end{aligned} \tag{1.2}$$

Stage 2 Program:

$$\begin{aligned} \min \qquad & Q_s(x, y_s) \\ & W_s y_s \leq h_s - T_s x \end{aligned} \tag{1.3}$$

In this work, we focus on the two-stage stochastic programs whose Stage 1 can be a linear/integer/mixed-integer program and Stage 2 is a linear program. The usual method of solving such stochastic linear program uses Bender's decomposition [1]. In this method, a candidate Stage 1 solution is obtained by optimizing the Stage 1 program. The candidate Stage 1 solution is evaluated against all the scenarios in Stage 2. Stage 2 optimization gives the scenario costs for the given Stage 1 solution, and optimality/feasibility cuts that are fed back to Stage 1. Stage 1 is re-optimized with the addition of new set of cuts to obtain another candidate solution. The iterative Stage 1-Stage 2 optimization continues until the optimal solution is found which is determined by a convergence criteria. A Stage 1 optimization followed by a Stage 2 optimization is called an iteration.

There are two variants of the Bender's approach. In one, a single cut using a weighted combination of Stage 2 dual objective function is added to the Stage 1 in each iteration. This method is called the L-shaped method [2]. In the other, in an iteration a cut constraint is added to Stage 1 for each scenario. This multicut method [3] has the advantage that the set of cuts in each iteration dominates a single L-shaped cut. However, the number of cuts can become very large quickly, particularly for problems with large number of scenarios.

In most real world applications, the number of uncertain parameters are large, and therefore the number of scenarios are also very large. In addition to the complexity of the focal application, factors such as the number of Stage 2 evaluations, number of rounds it takes to converge to optimality (within the user-specified convergence criteria), and the size of the Stage 1 linear program which increases with the increase in number of scenarios, add to the computational complexity of the stochastic programs.

Our research focuses on multicut Bender's method. Equation 1.4 shows the Stage 1

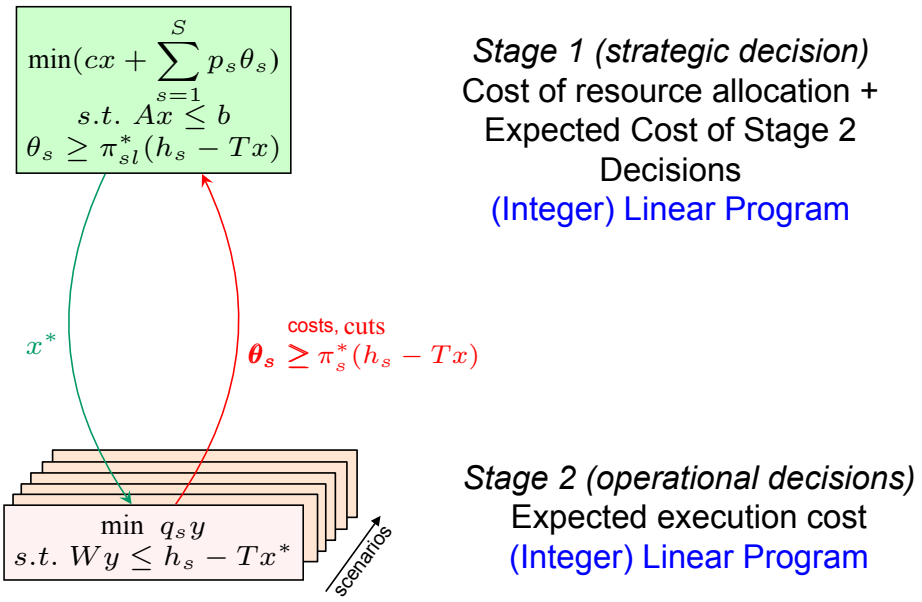


Figure 1.1: Benders decomposition and multicut L-shaped method for two-stage stochastic programs.

program after r rounds.

$$\begin{aligned} \min \quad & cx + \sum_s p_s\theta_s \\ s.t. \quad & Ax \leq B \\ \forall s \text{ and } l \in [1, r], \quad & E_{sl}x + \theta_s \leq e_{sl} \end{aligned} \tag{1.4}$$

where, $E_{sl} + \theta_s \leq e_{sl}$ are the cut constraints obtained from Stage 2 optimization and θ_s is the cost of scenario s.

Because of the large number of cuts in the multicut method, it is imperative that the cuts generated in each round are strong cuts in the sense that they allow the Bender's program to converge quickly. The multicut method is depicted in Figure 1.1.

Because of its heavy computational demands, stochastic optimization has been typically restricted to a relatively small number of scenarios. However, to model an application with fidelity requires hedging against hundreds to several thousands or more scenarios. This makes it a computationally challenging problem and hence the need for scalable algorithms and parallelization. Parallelization of stochastic optimization presents difficult, interesting

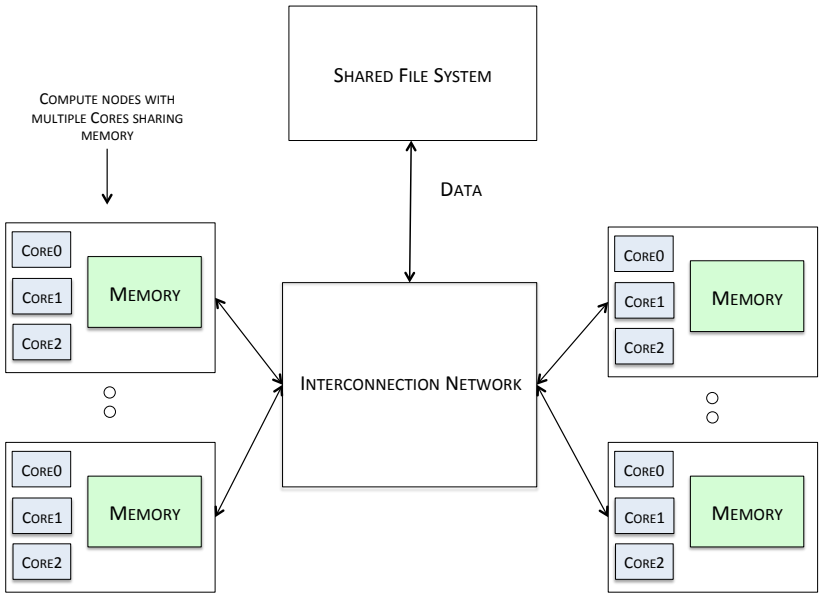


Figure 1.2: A rough sketch of a distributed system. Multiple compute nodes (each with possibly multiple cores sharing same memory) are connected to each other through an interconnection network. The compute nodes also have a shared file system for storing data files.

and unique challenges, which is probably why its extreme-scale parallelization has largely remained an unchartered territory. A naive parallelization of the Bender's decomposition can be done using a simple master-worker design which comprises of a master process that optimizes the Stage 1 linear program and multiple worker processes which will evaluate the scenarios in parallel. This design is very far from being either efficient or scalable. We discuss the scalability challenges by dividing our discussion over two classes of stochastic programs. They are classified based on the types of variables - Stochastic Linear Programs (Chapter 2, 4, 5) with integer variables in Stage 1 and Stage 2, and Stochastic Integer Programs (Chapter 3) with mixed-integer variables in Stage 1 and linear variables in Stage 2.

1.2 Parallel Computing

In parallel computing, multiple compute nodes are connected to each other through a network. The processes on the compute nodes communicate with each other by passing messages to each other through the network. The interconnection network could be a 1D/2D/3D/4D mesh/torus, a ring, a fat-tree, a fully-connected topology and so on. Each of the compute nodes itself could be a multi-processor system, in which the processors share the same memory. Figure 1.2 shows a diagram of a distributed system, and its various components. The various compute nodes can run jobs in parallel and exchange information if required either via message passing or memory sharing (when processes are on the same compute node). The total available memory in a distributed system is the sum of the memories of the individual nodes, and hence required number of nodes can be added to the system to have sufficient memory to store the stochastic program in memory.

There are several parallel programming models available to write parallel programs for distributed systems, such as, MPI [4], Charm++ [5], etc. Multiple processes are launched on the system, and each process is assigned a global rank. Processes can communicate messages amongst each other by specifying the rank of the sending/receiving process. The parallel programming model handles the delivery of message to the target rank.

Stochastic optimization algorithms have been growing rapidly in popularity over the last decade or two. Given the complexity of the systems and the scale at which the system parameters change, the time to solve stochastic models is very critical. It is therefore natural to develop techniques to utilize parallel computing resources (in the form of multi-core desktops and servers, clusters, super-computers) to cut down the solution times. An interesting development in this context, is the widespread availability of cloud computing platforms which offer computational resources and optimization solvers as services. Since users pay for the time they use these resources/services, it is critical to optimize the application. We strongly believe that this area has great potential for research in parallel computing community.

1.3 Thesis Organization

This thesis is divided into two major parts. Part 1 covers the proposed computational methods for solving large scale stochastic optimization problems. It is divided into four chapters - Chapter 2 to Chapter 5. Chapter 2 describes our parallel design of the stochastic program solver, along with various proposed optimizations to improve the performance of the solver. The design proposed in Chapter 2 has limited scalability for stochastic programs that have large number of integer variables in Stage 1 and/ or large number of Stage 2 scenarios. In Chapter 3, we propose a highly scalable BnB based solver design for solving large stochastic programs with mixed-integer variables in Stage 1. We call our solver as Parallel Stochastic Integer Program Solver or PSIPS. PSIPS shows strong scaling on up to 960 cores of a cluster, with parallel efficiency above 40% with very high probability. On the contrary, commercial state-of-the-art integer program solvers like Gurobi are known to have very poor parallel efficiency. Chapter 4, and Chapter 5 propose two decomposition schemes, namely the Split-And-Merge method and the lagrangean decomposition method, for accelerating the convergence of Bender's multicut method for stochastic program optimizations.

In Part 2 of the thesis, we propose models for scheduling military airlift assets. Unlike commercial air carriers, military airlift faces demands that are highly uncertain because they are subject to rapidly changing worldwide tensions and commitments of the military. Because of these changes, more than 90% of airlift missions have to be changed during either planning, or execution, or both. Our work proposes stochastic models that can significantly improve the efficiency of this problem. In Chapter 6, we discuss a military aircraft allocation problem, in which the aircraft are to be allocated to various missions and wings of the military one month in advance, when their demands are now known with certainty. We propose a stochastic optimization approach for obtaining robust solutions, and show its superiority over solutions obtained from deterministic optimization of real data obtained from US Air Mobility Command. In Chapter 7, we study the dynamic mission replanning problem of the US Air Mobility Command, in which a currently executing schedule has to be replanned because of dynamic disruptions such as weather events, aircraft breakdown, etc. We demonstrate that our stochastic formulation gives significantly superior solutions as

compared to myopic and deterministic optimization.

We conclude the thesis by presenting thesis contributions and future work in Chapter 8.

1.4 Literature Review

In this section, we do a literature review on several aspects of stochastic optimization. We first review the research and algorithmic advancement in the theoretical aspects of stochastic optimization in the last few decades. This is followed by a literaturre review of the applications of stochastic optimization. Finally, we do a survey of the related work on computational methods for stochastic optimization. In addition to this, chapters in the thesis also contain the relevant literature review wherever required.

1.4.1 Stochastic Optimization Theory

Stochastic programming was first introduced by George Dantzig [6]. He proposed the decomposition of stochastic programs into two or more stages and proposed the use recourse in the form of dual constraints to solve such stochastic programs [7]. Benders [] proposed the paritioning procedure for mixed-variable problems, and therefore it is also known as the Bender's method. Later, L-shaped [8] and multicut methods [3] were introduced to solve stochastic programs that need not have complete recourse, that is Stage 1 decisions are not necessary to be feasible for every Stage 2 scenario. A detailed introduction to stochastic programming can be dound in Birge and Louveaux [9].

Two-stage stochastic integer programs with mixed-integer variables in Stage 2 are computationally very hard problems and beyond the scope of this thesis. More about them can be found in [10–12]. In particular, for two-stage stochastic integer programs an excellent introduction is [13]. Sougie and van der Vlerk [14] discuss approximation algorithms and Ahmed [15] presents a comprehensive algorithmic perspective for stochastic integer programs. In our work, we deal with large scale parallelization of stochastic programs that have integer variables only in Stage 1. Stochastic integer programs with integer variables in Stage 2 are computationally much harder and are beyond the scope of this work.

1.4.2 Applications of Stochastic Optimization

There are a variety of applications that can be formulated as two-stage stochastic integer programs, for example, manufacturing [16], energy planning [17], logistics [18], etc. Gangammanavar et al [19] propose a stochastic programming framework for economic dispatch problem to address integration of renewable energy resources into power systems. Munoz et al [20] propose an approach for solving stochastic transmission and generation investment planning problem in which the reduce the number of scenarios by clustering the scenarios and using a representative scneario (centroid) from each cluster. Yue et al [21] show that stochastic programming model to schedule adaptive signal timing plans at oversaturated traffic signals outperforms deterministic linear programming in total vehicle delay. Park et al [22] propose a two-stage stochastic integer model for least-cost generation capacity expansion model to control carbon dioxide (CO2) emissions. Ahmed et al [23] propose a multi-stage stochastic integer programming approach for the problem of capacity expansion under uncertainty. Kim and Mehrotra [24] employed a two-stage stochastic integer programming approach for integrated staffing and scheduling problem with application to nurse management. Ariyawansa et al [25] have given free web access to a collection of stochastic programming test problems. SIPLIB [26] is another collection of test problems to facilitate computational and algorithmic research in stochastic integer programming. Depth and breadth of applications of stochastic optimization can be found in [27].

1.4.3 Computational Methods for Stochastic Optimizations

Guo et al [28] integrate progressive hedging [29] and dual decomposition [30] to accelerate the convergence of dual decomposition in stochastic integer program optimization. Eric et al [31] study of stage- and scenario-wise Fenchel decomposition for two-stage SIPs with special structure. Kawas et al [32] have developed the *Uncertainty Toolkit* for decision optimization under uncertainty. This is a user-friendly toolkit that solicits information on the uncertain data, automatically generates models that incorporate the uncertainty, and includes visual analytics for comparing outcomes. Becker discusses decomposition methods for stochastic and robust optimization problems for large-scale real world applications in this thesis [33].

Parallel processing for stochastic programming was proposed by Dantzig and Glynn [34], and has since been employed by Gondzio and Kouwenberg [35] and [36–38] among others. Ryan et al [39] propose a strategy for parallelizing Progressive Hedging (PH) to solve stochastic unit commitment problem. Anthony et al [40] use high performance computing for solving stochastic unit commitment subject to uncertainty in renewable power supply and generator and transmission line failures.

CHAPTER 2

Performance Optimizations for Two-stage Stochastic Linear Program Optimizations

2.1 Introduction

This chapter explores the parallelization of two-stage stochastic linear programs for resource allocation problems that seek an optimal solution in the first stage, while accounting for sudden changes in resource requirements by evaluating multiple possible scenarios in the second stage. Unlike typical scientific computing algorithms, linear programs (which are the individual grains of computation in our parallel design) have unpredictable and long execution times. This confounds both a priori load distribution as well as persistence-based dynamic load balancing techniques. We present a master-worker decomposition coupled with a pull-based work assignment scheme for load balance. We discuss some of the challenges encountered in optimizing both the master and the worker portions of the computations, and techniques to address them. Of note are cut retirement schemes for balancing memory requirements with duplicated worker computation, and scenario clustering for accelerating the evaluation of similar scenarios.

We base our work in the context of a real application: the optimization of US military aircraft allocation to various cargo and personnel movement missions in the face of uncertain demands. We demonstrate scaling up to 122 cores of an Intel®64 cluster; even for very small,

but representative datasets. Our decision to eschew problem-specific decompositions has resulted in a parallel infrastructure that should be easily adapted to other similar problems. Similarly, we believe the techniques developed in this chapter will be generally applicable to other contexts that require quick solutions to stochastic optimization problems.

We describe our design for a parallel program to solve a 2-stage stochastic linear optimization model for an aircraft planning problem. We present our parallel decomposition and some interesting considerations in dealing with computation-communication granularity, responsiveness, and the lack of persistence of work loads in an iterative setting. Related work is summarized in Section 2.2. In Section 2.3 we briefly describe the aircraft allocation problem and its formulation as a two-stage stochastic program. In Section 2.4 we discuss our parallel program design for the Benders decomposition approach. In Section 2.5, we present challenges and strategies for optimizing the Stage 1 component of the computations while in Section 2.6 we present our study of the Stage 2 computations. Scalability results are presented in Section 2.7.

2.2 Related Work

Stochastic linear programs can be solved using the extensive formulation(EF) [9]. Extensive formulation of a stochastic program is its deterministic equivalent program in which constraints from all the scenarios are put together in a single large scale linear program. e.g. the extensive formulation for a stochastic program corresponding to Stage 1 given in equations 2.1, 2.2, 2.3 and Stage 2 in equations 2.4, 2.5 can be written as:

$$\begin{aligned} \min \quad & c^T x + \textstyle\sum_{k=1}^{K} p_k q_k^T y_k \\ s.t. \quad & Ax = b, \\ & T_k x + W_{y_k} = h_k, \qquad k = 1, ..., K \\ & x \geq 0, y_k \geq 0, \qquad k = 1...., K \end{aligned}$$

EF results in a large linear program that quickly becomes too large to be solved by a single computer. Figure 2.1 shows the solution time for the extensive formulation of the problems of our concern using the Simplex and Interior Point Method (IPM) available in Gurobi [41]. Solution time increases rapidly with increase in the number of scenarios and hence this is not a practical way of solving the stochastic programs when the number of scenarios are large in number.

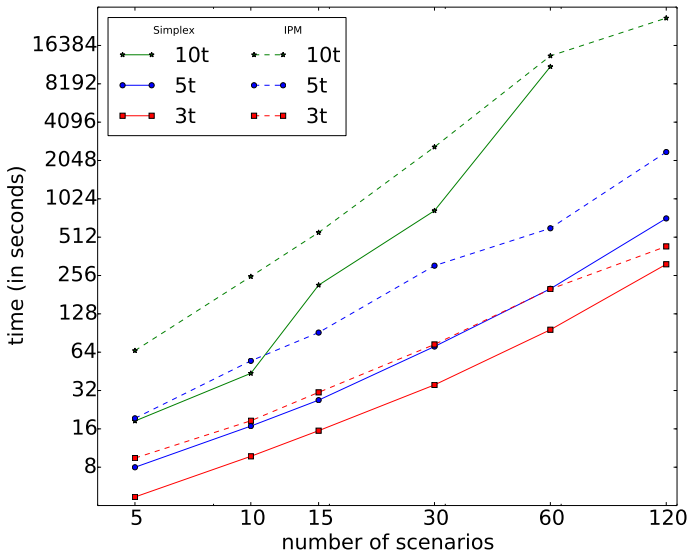


Figure 2.1: Time to solution using the Simplex and IPM methods in Gurobi for solving the extensive formulation of the stochastic programs.

Liner program solvers are hard to parallelize, and other parallelization approaches become necessary. Recently, there has been some work on parallelization of the simplex algorithms for linear programs with dual block-angular structure [42]. Lubin et al [43] demonstrated how emerging HPC architectures can be used to solve certain classes of power grid problems, namely, energy dispatch problems. Their PIPS solver is based on the interior-point method and uses a Schur's complement to obtain a scenario-based decomposition of the linear algebra. However, in our work we choose not to decompose the LP solves, but instead delegate them to a LP solver library. This reuses domain expertise encapsulated in the library and allows performance specialists to focus just on parallel performance. Using a library also allows the implementation to remain more general with the ability to use it for

other problems.

Linderoth, et. al. [38] have studied the performance of two-stage stochastic linear optimizations using the L-shaped algorithm on distributed grids. Unlike modern supercomputers, grids have high communication latencies and availability of nodes is sporadic. Hence, their work focuses on performance of an asynchronous approach to the Benders decomposition. In contrast, our work is based on a synchronous approach where a new iteration is initiated only after completion of all the scenario solves from the previous iteration.

2.3 Model Formulation & Approach

The United States Air Mobility Command (AMC) [1] manages a fleet of over 1300 aircraft [44] that operate globally under uncertain and rapidly changing demands. Aircraft are allocated at different bases in anticipation of the demands for several missions to be conducted over an upcoming time period (typically, fifteen days to one month). Causes of changes include demand variation, aircraft breakdown, weather, natural disaster, conflict, etc. The purpose of a stochastic formulation is to optimally allocate aircraft to each mission such that subsequent disruptions are minimized.

ACC (Tanker Airlift Control Center)[2] is responsible for allocating aircraft to three of the primary mission types flown by AMC: 1) Channel missions - regularly scheduled missions between the US and overseas locations, 2) Contingency missions - which are irregularly scheduled missions that deliver cargo to an international "hot spot," and 3) Special assignment airlift missions (SAAMs) - chartered by military units for a specific purpose. Aircraft are allocated by aircraft type, airlift wing, mission type and day. In situations when self-owned military aircraft are not sufficient for outstanding missions, civilian aircraft are leased. The cost of renting civilian aircraft procured in advance for the entire planning cycle is lower than the rent of civilian aircraft leased at short notice. Therefore, a good prediction of the aircraft demand prior to the schedule execution reduces the execution cost.

We model the allocation process as a two-stage stochastic linear program (LP) with Stage 1

[1] `http://www.amc.af.mil/`
[2] `http://www.618tacc.amc.af.mil`

generating candidate allocations and Stage 2 evaluating the allocations over many scenarios. This iterative method developed by Benders [45] has been widely applied to Stochastic Programming. Note that our formulation of the aircraft allocation model has complete recourse (i.e. all candidate allocations generated are feasible) because any demand (in a particular scenario) that cannot be satisfied by a candidate allocation is met by short term leasing of civilian aircraft at a high cost while evaluating that scenario.

In *Stage 1*, before a realization of the demands are known, decisions about long-term leasing of civilian aircraft are made, and the allocations of aircraft to different missions at each base location are also decided.

$$\min \quad Cx + \sum_{k=1}^{K} p_k \theta_k \tag{2.1}$$

$$s.t. \qquad Ax \leq b, \tag{2.2}$$

$$E_l x + \theta \leq e_l \tag{2.3}$$

In the objective function(2.1), x corresponds to the allocations by the aircraft type, location, mission and time. C is the cost of allocating military aircraft and leasing civilian aircraft. $\theta = \{\theta_k | k = 1, ..., k\}$ is the vector of Stage 2 costs for the k scenarios, p_k are the probability of occurrence of scenario k, l corresponds to the iteration in which the constraint was generated and $E_l(e_l)$ are the coefficients (right hand sides) of the corresponding constraints. Constraints in (2.2) are the feasibility constraints, while constraints in (2.3) are cuts which represents an outer linearization of the recourse function.

In *Stage 2*, the expected cost of an allocation for each scenario in a collection of possible scenarios is computed by solving LPs for that scenario.

$$\min \qquad {q_k}^T y \tag{2.4}$$

$$s.t. \quad Wy \leq h_k - T_k x \tag{2.5}$$

The second stage optimization helps Stage 1 to take the recourse action of increasing the capacity for satisfying an unmet demand by providing feedback in the form of additional constraints (cuts) on the Stage 1 LP (2.6). Here, π_k are the dual multipliers obtained

from Stage 2 optimization and x^* is the allocation vector obtained from the last Stage 1 optimization.

$$\theta_k \leq \pi_k * (h_k - T_k x^*) - \pi_k T_k (x - x^*) \tag{2.6}$$

A detailed description of our model and the potential cost benefits of stochastic vs deterministic models is discussed in Chapter 6. To illustrate the size of the datasets of interest, Table 2.1 lists the sizes of various airlift fleet assignment models. $3t$ corresponds to an execution period of 3 days, $5t$ for 5 days, and so on.

Table 2.1: Size of stochastic linear program datasets (each with 120 scenarios)

Model Name	Num Stage 1 variables	Num Stage 2 variables	Num Stage 2 constraints
3t	255	1076400	668640
5t	345	1663440	1064280
10t	570	3068760	1988640
15t	795	4157040	2805000
30t	1470	7956480	5573400

2.4 Parallel Stochastic Linear Program Solver (PSLPS) Design

In this section, we discuss the various design aspects of our parallel stochastic linear program sovler, called as PSLPS.

2.4.1 Parallel Programming Model

We have implemented the program in Charm++ [46, 47], which is a message-driven, object-oriented parallel programming framework with an adaptive run-time system. It allows expressing the computations in terms of interacting collections of objects and also implicitly overlaps computation with communication. Messaging is one-sided and computation

is asynchronous, sender-driven; facilitating the expression of control flow which is not bulk synchronous (SPMD) in nature.

2.4.2 Coarse Grained Decomposition

To exploit the state of the craft in LP solvers, our design delegates the individual LP solves to a library (Gurobi [41]). This allows us to build atop the domain expertise required to tune these numerically intensive algorithms. However, the same decision also causes a very coarse-grain of computation as the individual solves are not decomposed further. Parallel programs usually benefit from a medium or fine-grained decomposition as it permits a better overlap of computation with communication. In Charm++ programs, medium-sized grains allow the runtime system to be more responsive and give it more flexibility in balancing load. Adopting a coarse-grained decomposition motivates other mitigating design decisions described here. It also emphasizes any sequential bottlenecks and has been causative of some of our efforts in optimizing solve times.

2.4.3 Two-stage Design

Since the unit of sequential computation is an LP solve, the two-stage formulation maps readily onto a two-stage parallel design, with the first stage generating candidate allocations, and the second stage evaluating these allocations over a spectrum of scenarios that are of interest. Feedback cuts from the second stage LPs guides the generation of a new candidate allocation. There are many such iterations (rounds) until an optimal allocation is found. We express this as a master-worker design in Charm++ with two types (C++ classes) of compute objects. An *Allocation Generator* object acts as the master and generates allocations, while a collection of *Scenario Evaluator* objects are responsible for the evaluation of all the scenarios.

2.4.4 Unpredictable Grain Sizes

Experiments show that LP solves for different scenarios take different amounts of time. Hence, an a priori static distribution of scenarios across all the *Scenario Evaluators* will not

achieve a good load balance. Unlike typical algorithms in parallel, scientific computing, the time taken for an individual grain of computation (LP solve) is also devoid of any persistence across different iterations (rounds). This precludes the use of any persistence-based dynamic load balancers available in Charm++. To tackle this fundamental unpredictability in the time taken for a unit of computation we adopt a work-request or pull-based mechanism to ensure load-balance. We create a separate work management entity, *Work Allocator* object(*Comm* in Figure 2.2), that is responsible for doling out work units as needed. As soon as a *Scenario Evaluator* becomes idle, it sends a work request to the *Work Allocator* which assigns it an unevaluated scenario. Figure 2.2 is a schematic representing our design.

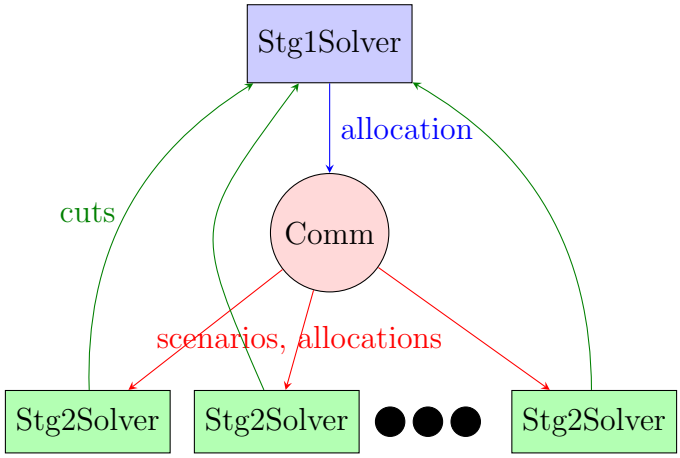


Figure 2.2: Parallel design schematic for two-stage Bender's decomposition

2.4.5 Maintaining Responsiveness

A pull-based mechanism to achieve load balance requires support from a very responsive *Work Allocator*. Charm++ provides flexibility in the placement of compute objects on processors. We use this to place the *Allocation Generator* and the *Work Allocator* objects on dedicated processors. This ensures a responsive *Work Allocator* object and allows fast handling of work requests from the *Scenario Evaluators*; unimpeded by the long, coarse-grained solves that would otherwise be executing.

2.5 Optimizing Stage 1

Advanced Starts The two-stage design yields an allocation that is iteratively evolved towards the optimal. Typically, this results in LPs that are only incrementally different from the corresponding LPs in the previous round as only a few additional constraints may be added every round. LP solvers can exploit such situations by maintaining internal state from a call so that a later call may start its search for an optimum from the previous solution. This is called advanced start (or warm start), and can significantly reduce the time required to find a solution to an LP. We enabled advanced starts for the Stage 1 LP and observed sizable performance benefits (Figure 2.3).

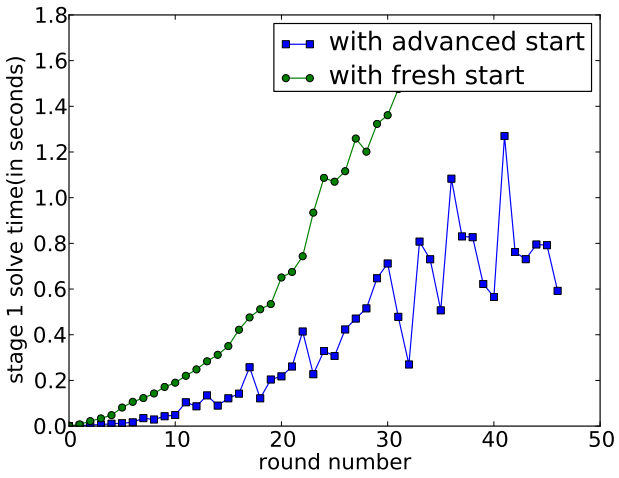


Figure 2.3: Stage 1 LP solve times with and without advanced start on 2.67 GHZ Dual Westmere Xeon

Memory Footprint and Bandwidth An observation from Figure 2.3 is that the Stage 1 solve time increases steadily with the round number irrespective of the use of advanced starts. Our investigation pointed to an increasing solver memory footprint as the cause for such behavior.

During each round, the *Allocation Generator* incorporates feedback from the evaluation of each scenario into the Stage 1 model. This feedback is in the form of constraints (cuts) which are additional rows added to a matrix maintained internally by the library. The number of cuts added to the model grows monotonically with the number of rounds; requiring an

increasing amount of memory to store and solve an LP. Figure 2.4 captures this trend by plotting memory utilization for the *Allocation Generator* object (which includes LP library memory footprint) and the time taken for the Stage 1 solves by round number. The memory usage is as high as 5 GB and the solve time for a single grain of Stage 1 computation can reach 100s.

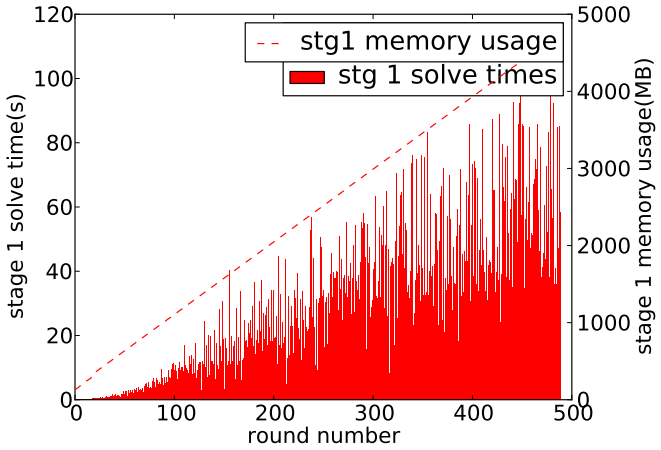


Figure 2.4: Stage 1 memory usage and LP solve times for 15 time period model on Dell 2.6 GHz Lisbon Opteron 4180

To improve the characterization of the LP solves, we designed an experiment that artificially limits the memory bandwidth available to a single LP solver instance by simultaneously running multiple, independent LP solver instances on a multicore node. Our results (Figure 2.5) show that for the same problem size, the time to solution of an LP is increased substantially by limiting the available memory bandwidth per core. As the Stage 1 model grows larger every round, it becomes increasingly limited by the memory subsystem and experiences dilated times for LP solves.

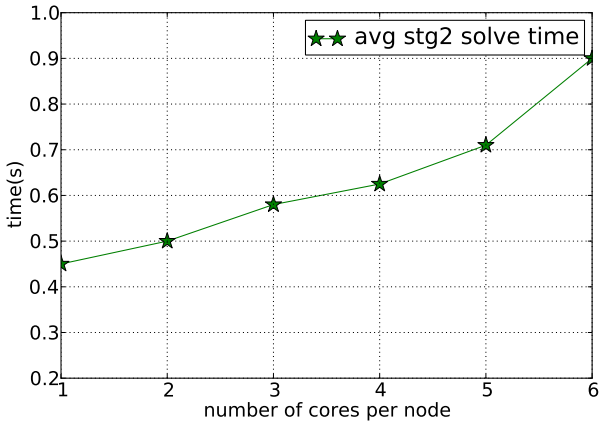


Figure 2.5: The impact of artificially constraining memory bandwidth available for an LP solve (10 time period model) on a system with Intel 64(Clovertown) 2.33 GHz dual socket quad core processor with 1333MHz front size bus (per socket), 2x4MB L2 cache and 2 GB/core memory.

2.5.1 Curbing Solver Memory Footprint

For large Stage 1 problems, which take many iterations to converge, the increasing Stage 1 solve times and the increasing memory demands exacerbate the serial bottleneck at the *Allocation Generator*, and pose a threat to the very tractability of the Benders approach. However, an important observation in this context is that not all the cuts added to a Stage 1 problem may actually constrain the feasible space in which the optimum solution is found. As new cuts are added, older cuts may no longer be binding or active. They may become active again in a later round or maybe rendered redundant if they are dominated by newer cuts. Such cuts simply add to the size of the Stage 1 model and its solve time, and can be safely discarded. Figure 2.6 plots a histogram of the cut usage rate (defined by equation 2.7) for the cuts generated during the course of convergence of a 5 time period model. Most of the cuts have very low usage rates while a significant number of the cuts are not used at all. This suggests that the size of the Stage 1 problem may be reduced noticeably without diluting the description of the feasible space for the LP solution.

$$\text{Cut Usage Rate} = \frac{\text{num rounds in which cut is active}}{\text{num rounds since its generation}} \tag{2.7}$$

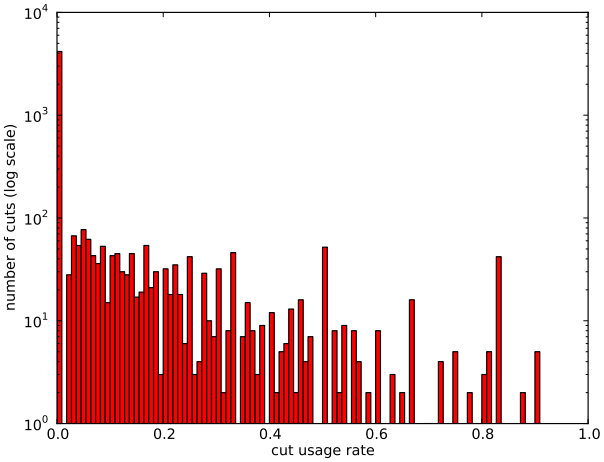


Figure 2.6: Cut usage rate for $5t$ model

We therefore implemented a cut retirement scheme that discards/retires cuts whenever the total number of cuts in the Stage 1 model exceeds a configurable threshold. After every round of the Benders method, the cut score is updated based on it's activity in that round. Cuts with small usage rates (defined by Equation 2.7) are discarded. The desired number of lowest scoring cuts can be determined using a partial sort that runs in linear time.

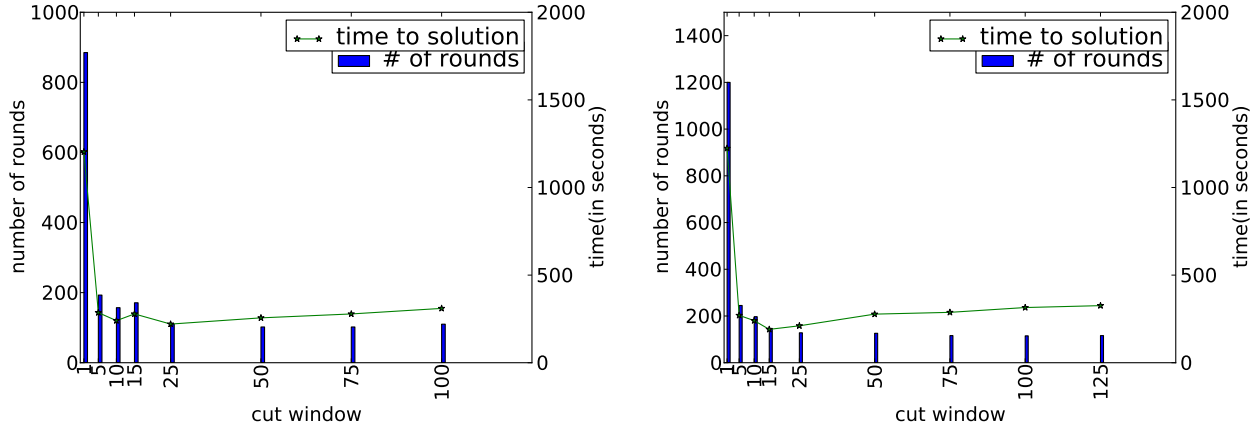
Discarding a cut that may be required during a later round only results in some repeated work. This is because the Benders approach will cause any necessary cuts to be regenerated via scenario evaluations in future rounds. This approach could increase the number of rounds required to reach convergence, but lowers execution times for each Stage 1 LP solve by limiting the required memory and access bandwidth. Figure 2.7 demonstrates these effects and shows the benefit of cut management on the Stage 1 memory usage and solve times of the 15 time period model solved to 1% convergence tolerance. The time to solution reduced from 19025s without cut retirement to 8184s with cut retirement - a 57% improvement.

Figure 2.7: Stage 1 LP solve times and memory usage for the 15 time period model solved to 1% convergence with *Cut Window* of 75 (run on 8 cores of 2.6 GHz Lisbon Opteron 4180)

We define a *Cut Window* as the upper limit on the number of cuts allowed in the Stage 1 model, expressed as the maximum number of cuts divided by the number of scenarios. Figure 2.8a and 2.8b describe the effect of different *Cut Window*s on the time and number of rounds to convergence. Smaller *Cut Window*s reduce the individual Stage 1 solve times, leading to an overall improvement in the time to solution even though it takes more rounds to converge. However, decreasing the *Cut Window* beyond a certain limit, leads to a significant increase in the number of rounds because several useful cuts are discarded and have to be regenerated in later rounds. Further reducing the *Cut Window* makes it impossible to converge because the collection of cuts is no longer sufficient. These experiments demonstrate the need to make an informed choice of the *Cut Window* to get the shortest time to solution, e.g. for the 5 time period model with 120 scenarios, an optimal *Cut Window* size is close to 25 while for the 10 time period model with 120 scenarios it is close to 15.

2.5.2 Evaluating Cut-Retirement Strategies

We investigate cut management further to study it's performance with different cut scoring schemes. Three cut scoring schemes are discussed here namely, the least frequently used, the least recently used and the least recently/frequently used. Each of these are briefly discussed



(a) 5 time period model (solved to 0.1% convergence on 8 cores of 2.26 GHz Dual Nehalem)

(b) 10 time period model (solved to 1% convergence on 32 cores of 2.67 GHz Intel Xeon hex-core processors)

Figure 2.8: Performance of $5t$ and $10t$ with different *Cut Windows*

here:

- *Least Frequently Used (LFU)* A cut is scored based on it's rate of activity since it's generation (equation2.7). This scoring method was used for results presented in Figure 2.8a and 2.8b.

- *Least Recently Used (LRU)* - In this scheme, the recently used cuts are scored higher. Therefore, a cut's score is simply the last round in which it was active.

$$LRU\_Score = \text{Last active round for the cut}$$

- *Least Recently/Frequently Used (LRFU)* This scheme takes both the recency and frequency of cut activity into account. Each round in which the cut was active contributes to the cut score. The contribution is determined by a weighing function $\mathcal{F}(x)$, where x is the time span from the activity in the past to current time.

$$LRFU\_Score = \sum_{i=1}^{k} \mathcal{F}(t_{base} - t_i)$$

where $t_1, t_2, ..., t_k$ are the active rounds of the cut and $t_1 < t_2 < ... < t_k \leq t_{base}$. This

policy can demand a large amount of memory if each reference to every cut has to be maintained and also demands considerable computation every time the cut retirement decisions are made. Lee, et. al. [48] have proposed a weighing function $\mathcal{F}(x) = (\frac{1}{p})^{\lambda x}$ $(p \geq 2)$ which reduces the storage and computational needs drastically. They tested it for cache replacement policies and obtained competitive results. With this weighing function, the cut score can be calculated as follows:

$$S_{t_k} = \mathcal{F}(0) + \mathcal{F}(\delta) S_{t_{k-1}},$$

where S_{t_k} is the cut score at the kth reference to the cut, $S_{t_{k-1}}$ was the cut score at the $(k-1)$th reference and $\delta = t_k - t_{k-1}$. For more details and proofs for the weighing function refer to [48]. We use $p = 2$ and $\lambda = 0.5$ for our experiments.

Figure 2.9 compares the result of these strategies. LRFU gives the best performance of the three. The *cut windows* used for these experiments were the optimal values obtained from experiments in Figure 2.8a and 2.8b.

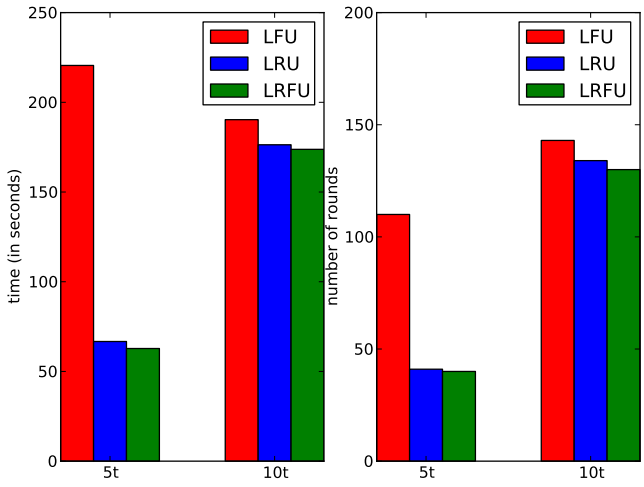


Figure 2.9: Performance of different cut scoring strategies for the 5 time period model(8 cores, cut-window=25, 0.1% convergence) and the 10 time period model(32 cores, cut-window=15, 1% convergence)

2.6 Optimizing Stage 2

2.6.1 Advanced Starts

In every iteration, there are as many Stage 2 LP solves as there are scenarios. This constitutes the major volume of the computation involved in the Benders approach because of the large number of scenarios in practical applications. Even a small reduction in the number of rounds or average Stage 2 solve times can have sizable payoffs. In this section, we analyze different strategies to reduce the amount of time spent in Stage 2 work.

2.6.2 Variability Across Runs

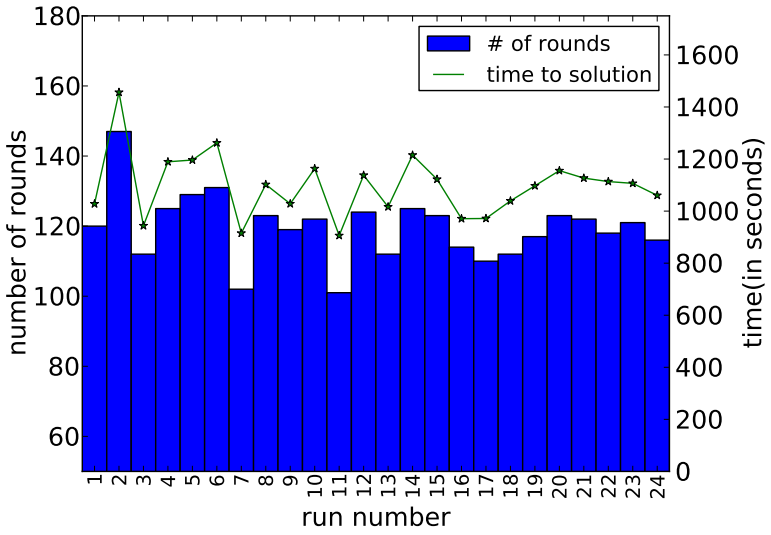


Figure 2.10: Variation across runs with advanced-start and their comparison with fresh start ($10t$ model) on 8 cores of 2.67 GHz Dual Nehalem

Figure 2.10 also shows the number of rounds and time to solution for 25 runs on the same model. An interesting note is the variability across various runs of the same program.

Scenarios are assigned to *Scenario Evaluators* in the order in which work requests are received. This varies across different runs because of variable message latencies and variable LP solve times. With advanced starts, this results in different LP library internal states as

starting points for a given scenario evaluation; yielding different cuts for identical scenario evaluations across different runs. This variation in cuts affects the next generated allocation from Stage 1 and the very course of convergence of the execution.

Variation across different runs make it difficult to measure the effect of different optimization strategies. Additionally in some situations, obtaining an optimal solution in predictable time can be more important than obtaining it in the shortest possible time. Therefore, mitigating the variability can be an important consideration.

Note that to verify that multiple solutions are not the artifact of a loose termination criteria, we solved the problems to very tight convergence criteria (up to 0.00001%). Identical runs resulted in different solutions implying that the problem is degenerate.

2.6.3 Clustering Similar Scenarios

Turning off the advanced start feature can significantly increase the time to solution and hence is not a viable approach. However, the scenario evaluation order can be pre-determined by assigning a fixed set of scenarios to each solver. This approach can potentially decrease the efficiency of the work-request mechanism at balancing Stage 2 load because work is now assigned in larger clusters of scenarios.

However, since some scenarios may exhibit similarities, it may be possible to group similar scenarios together to increase the benefits of advanced starts. It may be beneficial to trade coarser units of work-assignment (poorer load balance) for reduced computation grain sizes. We explore this by implementing scenario clustering schemes and cluster-based work assignment. Similarity between scenarios can be determined either by using the demands in each scenario, the dual variable values returned by them, or by a hybrid of demands and duals. Our current work has used the demands to cluster scenarios because they are known a priori– before the stochastic optimization process begins. We use a k-means [49] algorithm for clustering scenarios. Since, the clusters returned from k-means can be unequal in size, we use a simple approach (described in Algorithm 1) to migrate some scenarios from over-sized clusters to the under-sized clusters. We also implement random clustering for reference.

Algorithm 1: The Scenario Clustering Approach

Input

D_i- Demand set for scenario i $(i = 1, 2, ..., n)$

k - number of clusters

Output

k equally sized clusters of scenarios

Algorithm

{label, centroids} = kMeans({D_1, D_2, D_3, ..., D_n}, k)

IdealClusterSize = $\frac{n}{k}$

$size_i$ = size of cluster i

{Identify Oversized clusters}

$\mathcal{O} = \{c \in Clusters \mid size_c > IdealClusterSize\}$

{Identify Undersized clusters}

$\mathcal{U} = \{c \in Clusters \mid size_c < IdealClusterSize\}$

$\mathcal{S}$: set of adjustable points

for $c \in \mathcal{O}$ **do**

  Find $(size_i - IdealClusterSize)$ points in cluster c that are farthest from $centroid_c$ and add them to the set $\mathcal{S}$

end for

while $size(\mathcal{S}) > 0$ **do**

  Find the closest pair of cluster $c \in (U)$ and point $p \in \mathcal{S}$

  Add p to cluster c

  Remove p from $\mathcal{S}$

  if $size_c == IdealClusterSize$ **then**

    Remove c from $\mathcal{U}$

  end if

end while

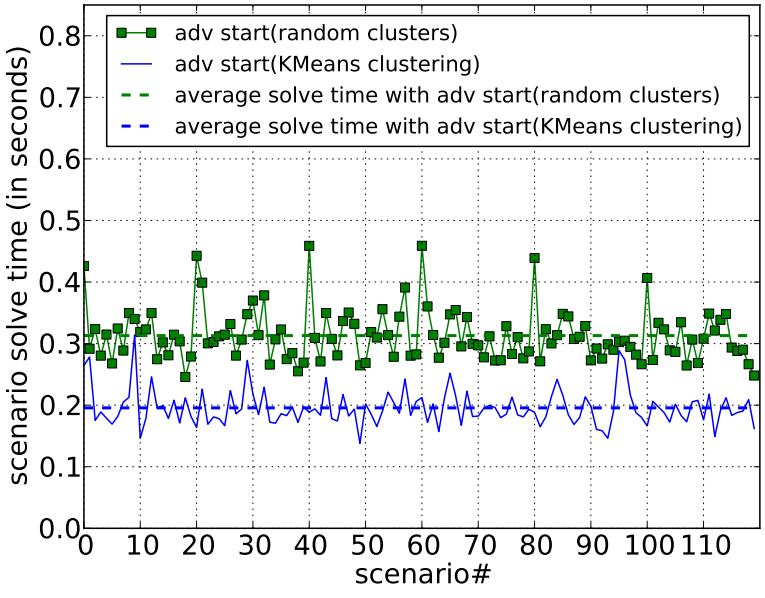
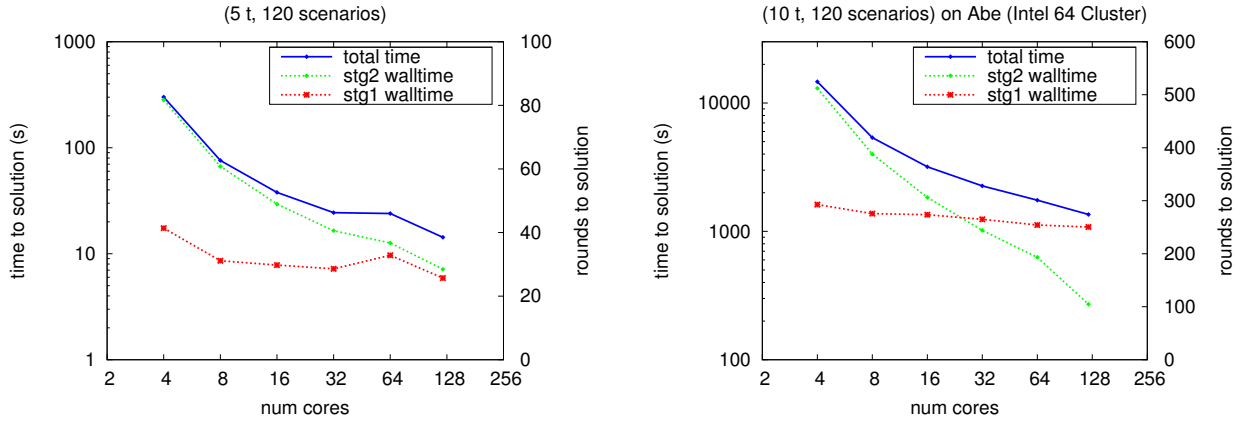


Figure 2.11: Comparison of average Stage 2 solve time between Stage 2 fresh start, advanced start with clustering and advanced start without clustering on 2.6 GHz AMD Lisbon Opteron



(a) 5 time period problem solved to 0.1% convergence (b) 10 time period problem solved to 0.1% convergence

Figure 2.12: Parallel scalability for $5t$ and $10t$ models using optimized Bender's decomposition

Figure 2.11 compares the improvement in average Stage 2 solve times when scenarios are clustered using Algorithm 1.

2.7 PSLPS Scalability

With the optimizations described above, we were able to scale medium-sized problems up to 122 cores of an Intel-64 Clovertwon (2.33 GHz) cluster with 8 cores per node. For 120 scenarios, an execution that uses 122 processors represents the limit of parallel decomposition using the described approach: one Stage 1 object, one *Work Allocator* object, and 120 *Scenario Evaluators* that each solve one scenario. Figure 2.12a and 2.12b show the scalability plots with Stage 1 and Stage 2 wall time breakdown. The plots also demonstrate Amdahl's effect as the maximum parallelism available is proportional to the number of scenarios that can be solved in parallel, and scaling is limited by the sequential Stage 1 computations. It must be noted that real-world problems may involve several hundreds or thousands of scenarios, and our current design should yield significant speedups because of Stage 2 parallelization.

2.8 Summary

Most stochastic programs incorporate a large number of scenarios to hedge against many possible uncertainties. Therefore, Stage 2 work constitutes a significant portion of the total work done in stochastic optimizations. For stochastic optimization with Benders approach, the vast bulk of computation can be parallelized using a master-worker design described in this chapter. We have presented experiments, diagnoses and techniques that aim to improve the performance of each of the two stages of computation.

We presented an *LRFU* based cut management scheme, that completely eliminates the memory bottleneck and significantly reduces the Stage 1 solve time, thus making the optimization of large scale problems tractable. We analyzed different aspects of the Stage 2 optimization and have presented some interesting avenues for further studies in improving Stage 2 performance. With our techniques, we were able to obtain a speedup of about 21 and 11 for the 5 and 10 time period problems, respectively with 120 scenarios each as we scaled from 4 cores to 122 cores. Much higher speedups can be obtained for real-world problems which present much more Stage 2 computational loads. In our current design, Stage 1 still presents a serial bottleneck that inhibits the efficiency of any parallel implementation. We are currently exploring methods such as Lagrangean decomposition to alleivate this. We believe that some of our strategies can be applied to other stochastic programs too; and that this work will be of benefit to a larger class of large, commercially relevant, high impact stochastic problems.

CHAPTER 3

Parallel Branch-and-Bound for Two-stage Stochastic Integer Programs

Many real-world planning problems require searching for an optimal integer solution in the face of uncertain input. If integer solutions are required, then branch-and-bound techniques are the accepted norm. However, there has been little prior work in parallelizing and scaling branch-and-bound algorithms for stochastic optimization problems.

In this chapter, we explore the parallelization of a two-stage stochastic integer program solved using branch-and-bound. We present a range of factors that influence the parallel design for such problems. Unlike typical, iterative scientific applications, we encounter several interesting characteristics that make it challenging to realize a scalable design. We present two design variations that navigate some of these challenges. Our designs seek to increase the exposed parallelism while delegating sequential linear program solves to existing libraries.

We evaluate the scalability of our designs using sample aircraft allocation problems for the US airfleet. It is important that these problems be solved quickly while evaluating large number of scenarios. Our attempts result in strong scaling to hundreds of cores for these datasets. We believe similar results are not common in literature, and that our experiences will feed usefully into further research on this topic.

3.1 Introduction

This chapter presents our parallel algorithms for scalable stochastic integer optimization. Specifically, we are interested in problems with integer solutions, and hence, in BnB approaches. Although BnB is a well-studied method, there has been little prior work in parallelizing or scaling two-stage, stochastic Integer Programs (IPs). Unlike typical, iterative scientific applications, we encounter some very interesting characteristics that make it challenging to realize a scalable design. The total amount of computation required to find optima is not constant across multiple runs. This challenges traditional thinking about scalability and parallel efficiency. It also implies that reducing idle time does not imply quicker runs. The sequential grains of computation are quite coarse. They display a wide variation and unpredictability in sizes. The structure of the branch-and-bound search tree is sensitive to several factors, any of which can cause significantly alter the search tree causing longer times to solution. We explore the causes for this fragility and evaluate the trade-offs between scalability and repeatability.

We structure this chapter to expose the design influences on parallel solutions of stochastic IPs. Once past the introductory sections (3.2–3.4), we present our approach to parallelizing stochastic IPs (3.5), and discuss the factors we considered while designing a parallel BnB for such optimization problems (3.6). This section presents some of the challenges that set this problem apart from typical parallel computational science applications. We pick a programming model that enables the expression and management of the available parallelism in section 3.7. Finally, we present two primary design variations (3.8 and 3.9), and analyze their performance in section 3.10.

The context for our work is a US fleet management problem where aircraft are allocated to cargo movement missions under uncertain demands (3.3). However, the design discussions and parallelization techniques are not specific to it.

3.2 Two-stage Stochastic Integer Optimization

As in the case of stochastic linear programs, two-stage stochastic integer optimization with integer variables only in Stage 1 is commonly solved using Benders decomposition [1], where candidate solutions are generated in Stage 1 and are evaluated in Stage 2 for every scenario (Figure 3.1). Stage 1 (Eq.3.1) gets *feedback* from Stage 2 (Eq.3.2) in the form of *cuts* (Eq.3.3), which are used by the Stage 1 to improve the candidate integer solution. The process iterates until no improvement can be made.

$$\min \quad cx + \sum_{s=1}^{s} p_s \theta_s \quad s.t.\ Ax \leq b \tag{3.1}$$

$$\theta_s = \quad \min({q_s}^T y) \quad s.t.\ Wy \leq h_s - T_s x \tag{3.2}$$

$$\theta_s \geq \quad \pi_s^*(h_s - Tx^*) \tag{3.3}$$

where, x is the candidate integer solution, c is the cost coefficient vector, $\theta = \{\theta_s | s = 1..S\}$ are the Stage 2 costs for the S scenarios, p_s is the probability of occurrence of scenario s, π_s^* is the optimal dual solution vector for Stage 2 LP of scenario s. This method is also called the multicut L-shaped method [3] in which one cut per scenario is added to the Stage 1 in every iteration/round.

We restrict our work to the problems in which Stage 2 has only linear variables. When only linear variables are present in Stage 1 also, we call it a stochastic LP. And when Stage 1 has integer variables, we call it a stochastic IP or a stochastic Mixed Integer Program (MIP). Louveaux and Schultz [50], Sahinidis [51], in the broader context of decision-making under uncertainty, give excellent overviews of stochastic IP problems.

3.3 Case Study: Military Aircraft Allocation

As in Chapter 2, we use stochastic integer program datasets that model the military aircraft allocation problem. Integer solutions are required because aircraft need to be dedicated

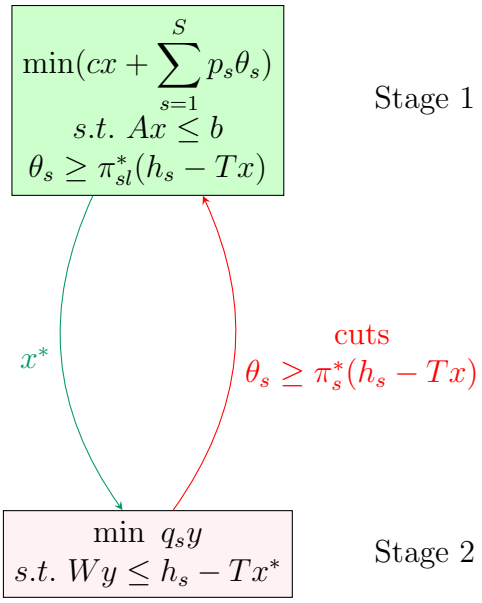


Figure 3.1: Benders decomposition for 2-stage stochastic integer programs

Table 3.1: Size of Stage 1 IP and Stage 2 lp of stochastic integer program models

Test	1st Stage		2nd-Stage Scenario		Nonzero Elements		
Problem	Vars.	Constrs.	Vars.	Constrs.	A	W_i	T_i
2t	54	36	6681	4039	114	20670	84
3t	81	54	8970	5572	171	27991	88
4t	108	72	11642	7216	228	36422	140
5t	135	90	13862	8669	285	43518	168
8t	216	144	20944	13378	456	66881	252
10t	270	180	25573	16572	570	82797	308

completely to individual missions. These models are classified based on the number of time periods (days) in the planning window and the number of possible scenarios that need to be evaluated to account for the uncertainty. The sizes of the LPs are given in Table 3.1. For e.g., the 5t-120 dataset has approximately 135 integer variables in the Stage 1 IP, $1.6M$ variables in the Stage 2 LP, and about $1M$ Stage 2 constraints when evaluating 120 Stage 2 scenarios. Similarly, 3t-240 stands for the 3t model with 240 scenarios, and so on. These models can be downloaded in SMPS[1] format from our website[2].

[1] http://myweb.dal.ca/gassmann/smps2.htm
[2] http://ppl.cs.illinois.edu/jetAlloc/

3.4 Prior Work

Parallelizing IP optimizations using BnB is in itself a challenging problem. Large scale solvers for Mixed Integer Programs (MIPs) have been studied before [52, 53]. The difficulty in achieving high efficiencies has been documented. Kale et al [54] have studied the challenges of dynamic load balancing in parallel tree search implementations. Gurobi [41] has a state-of-the art mixed integer program solver that exploits multi-core architectures. However, Koch et al in [53] observe that Gurobi suffers from poor efficiency (typically about 0.1) as it scales from 1 to 32 threads, the reason being that the number of BnB vertices needed to solve an instance varies substantially with different number of threads.

Our work involves optimization of stochastic IPs, which have decomposable program structure and large size. It presents further challenges that make it even harder to parallelize than just IPs. Examples of the uses of stochastic integer programming can be found in literature. Bitran et al [55] model production planning of style goods as a stochastic mixed IP. Dempster et al [16] consider heuristic solutions for a stochastic hierarchical scheduling problems. A comprehensive listing of work on stochastic IPs can be found here [56].

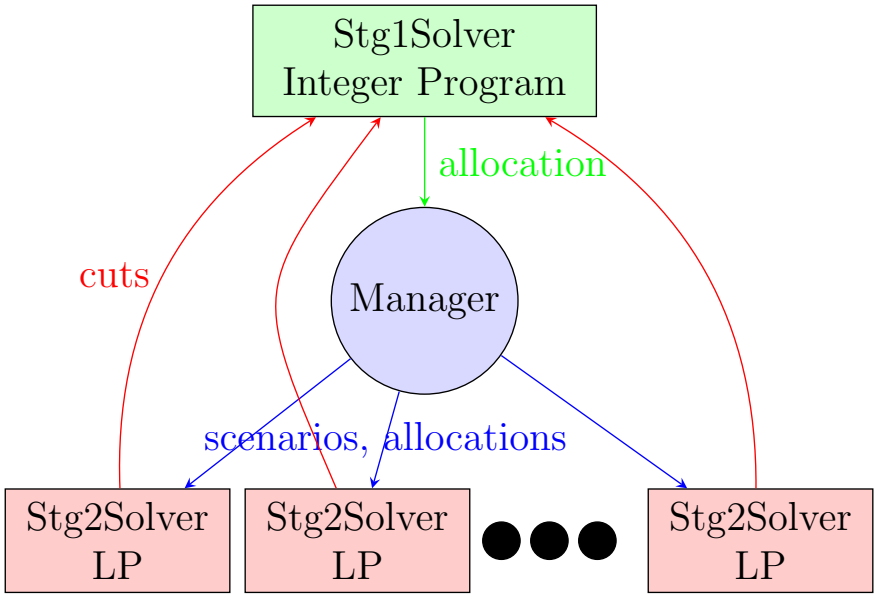
A stochastic program can be solved using its extensive formulation, which is its deterministic equivalent in which variables and constraints from all the scenarios are combined together in a single large LP. This LP can then be fed to any of the several open or commercial LP/IP solvers. However, Escudero et al [57] note that MIP solvers such as CPLEX [58] do not provide solution for even toy instances of two stochastic IPs in a viable amount of time.

We have not found systematic studies of large-scale stochastic integer optimization in literature. PySP [59, 60] is a generic decomposition-based solver for large-scale multistage stochastic MIPs. It provides a Python based programming framework for developing stochastic optimization models. For the solution of the stochastic programs, it comes with parallel implementations of algorithms such as Rockafellar and Wets' progressive hedging [29]. The basic idea of Progressive Hedging (PH) approach is to obtain the solution for every scenario independently. Every scenario will possibly give a different solution for Stage 1 variables, and hence is not an implementable solution. Therefore, penalty terms corresponding to Stage

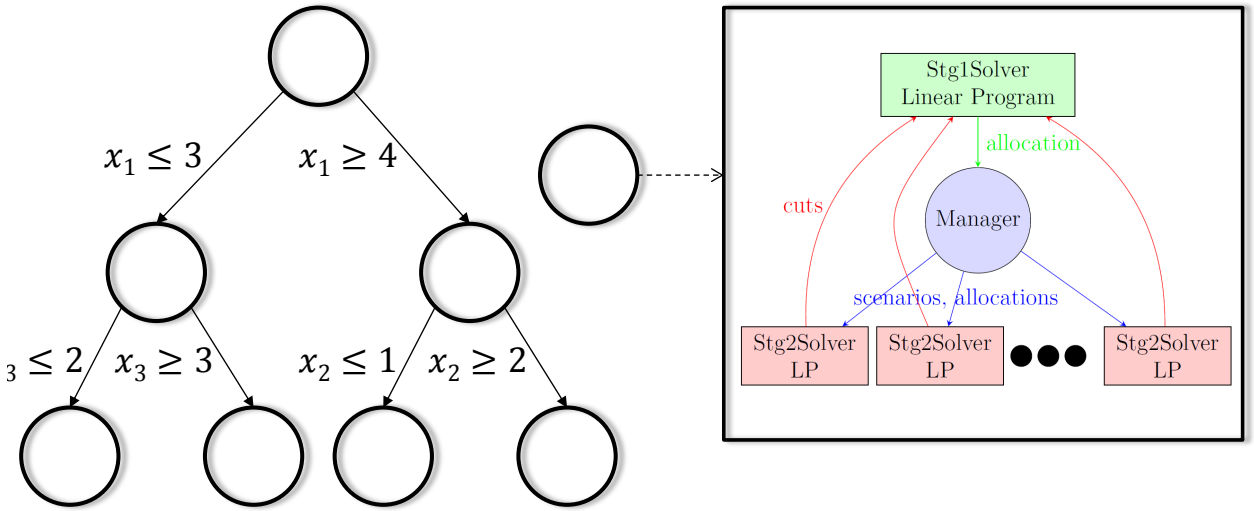
1 variables are added to the objective function for violating the lack of implementability. These penalty terms are the lagrangean multipliers that are iteratively updated by using a subgradient method. The iterations continue until an implementable solution is obtained. This method tends to be a heuristic method and require significant parameter tuning by the users. To the extent of our knowledge, the computational and scaling behavior of this framework have not been explored and the solver suffers from poor parallel efficiency because of MIP solve times. Recent work of Lubin et al [61] is based on parallelizing the dual decomposition of Stage 1 integer program by using interior-point solvers. Their study is limited to 32 cores and the approach suffers from load imbalance.

3.5 Parallelization Approach

Two-stage stochastic optimization problems have a natural expression in a two-stage software structure. The first stage proposes candidate solutions and the second stage evaluates multiple scenarios that helps refine the solution from the first stage. In Chapter 2 on stochastic LP, we focused on an iterative, two-stage master-worker design for the solution of stochastic linear programs. This tapped the readily available parallelism in Stage 2 by evaluating multiple possible scenarios simultaneously (Figure 3.2a). Although such a design captured much of the low-hanging, easily exploitable parallelism, it was quickly limited by the serial bottleneck of performing Stage 1 computations (Figure 3.3).



(a) Naive parallelisation with Benders decomposition



(b) Nested parallelism with Branch-and-Bound and Benders decomposition

Figure 3.2: Exploiting nested prallelism in stochastic integer programs

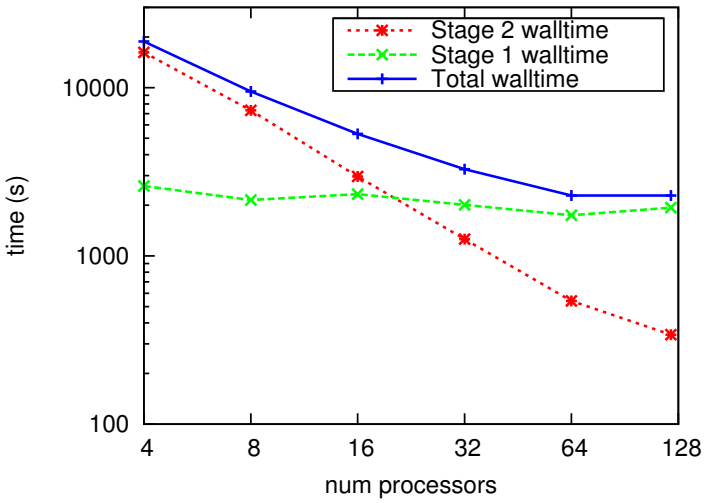


Figure 3.3: Scaling limited by Amdahl's law in a master-worker design for stochastic linear optimization. Results for 10t-1000 model obtained on Abe (dual quad-core 2.33GHz Intel Clovertown nodes with GigE)

In contrast to earlier work, this chapter focuses on the solution of stochastic integer programs (IP), which requires that Stage 1 solve an IP for every iteration. Since solving an IP is much more computationally expensive than an LP, this will magnify the serial bottleneck of the master-worker design such that it becomes completely untenable. Thus, it is imperative to reduce and hide this sequential bottleneck by exposing more parallelism.

Our approach to parallelizing stochastic IPs is by using BnB to obtain integer solutions to Stage 1 variables. We start by relaxing the integrality constraints in Stage 1 and solve the stochastic LP. BnB proceeds by branching on fractional parts of a solution obtained from the stochastic LP and restricting each branch to disjoint portions of the search space until gradually all variables in the solution become integral. This yields a tree where each vertex has one additional constraint imposed on the feasible space of solutions than its parent. We find a solution to this additionally constrained two-stage stochastic LP at this vertex, and then continue to branch. Therefore, each vertex in our BnB tree is a stochastic LP. The stochastic LP at each vertex permits evaluating each of the multiple scenarios in parallel. Additionally, the BnB search for integer solutions permits exploring the disjoint portions of the search space (i.e. the tree vertices) in parallel. Thus there are two sources of parallelism - simultaneous evaluation of Stage 2 scenarios and the simultaneous exploration of BnB

tree vertices. This nested parallelism (Figure 3.2b) has to be exploited for any reasonable scalability.

A relevant observation that influences processor utilization is the mutual exclusivity of the two stages of the stochastic programs. For a given vertex, Stage 1 cannot proceed while it is waiting for feedback from Stage 2, and Stage 2 is necessarily dependent on Stage 1 for each new candidate solution. Ensuring high utilization of compute resources will therefore require interleaving the iterative two-stage evaluation of multiple BnB vertices. This is also what makes this application distinct from the traditional applications of BnB. In traditional applications of BnB such as integer programming, traveling salesman problem (TSP), game tree search algorithms, etc. each tree vertex is an atomic unit of work i.e. when a vertex is processed it is either pruned or tagged as an incumbent solution or branches to generate children. No further processing of that vertex is required. On the other hand, in our application, each tree vertex is a stochastic LP optimization and therefore can require multiple rounds of Stage 1 and Stage 2 computations for optimization. While a vertex is being processed in Stage 2, its Stage 1 state has to be saved, so that it can be retrieved for the next Stage 1 computation (which will happen when the corresponding current Stage 2 finishes).

3.6 Design Considerations

In this section, we discuss the various factors that play an important role in deciding the parallel design for the nested parallelism of stochastic integer programs proposed in the previous section.

3.6.1 Coarse-Grained Decomposition

In our designs, we choose to delegate sequential LP solutions in Stage 1 and Stage 2 to an existing optimization library. This allows us to leverage the expertise encapsulated in these highly tuned libraries and focus on the parallelization and accompanying artifacts. Hence, the fundamental unit of sequential computation in our designs is a single linear program

Figure 3.4: Sample execution profile of evaluating multiple Stage 2 scenarios for candidate Stage 1 solutions. Each processor (horizontal line) is assigned a specific Stage 2 scenario, and evaluates multiple candidate solutions from Stage 1 one after the other. Colored bars represent an LP solve, while white stretches are idle times on that processor. LP solve times vary significantly and show no persistence, both across scenarios and across candidate solutions.

solve. This results in very coarse grain sizes.

3.6.2 Unpredictable Grain Sizes

There is sizeable variation in the time taken for an LP solve in both Stage 1 and Stage 2. Additionally, there is no persistence in the time taken for LP solves. A single Stage 1 LP for a given vertex may take widely varying times as a result of the addition of a few cuts from Stage 2. Likewise, we do not observe any persistence in Stage 2 LP solve times either across different scenarios for a given Stage 1 candidate solution, or for the same scenario across different candidate solutions. An illustrative execution profile is presented in Figure 3.4.

3.6.3 Varying Amounts of Available Parallelism

The BnB tree exposes a varying amount of parallelism as the search for an optimum progresses. The search starts with a single vertex (the tree root) being explored. More parallelism is gradually uncovered in a ramp-up phase, as each vertex branches and creates new vertices. However, once candidate integer solutions are found, the search tree can be pruned to avoid unnecessary work. For large enough search trees, there is usually a middle phase when there are a large, but fluctuating number of vertices on the exploration front depending on branching and pruning rates. Once the optimum is found, the remaining work involves proving its optimality by exploring the tree until all other vertices are pruned. Towards the end, pruning starts to dominate and the front of exploration shrinks rapidly. Any parallel design has to necessarily cope with, and harness these varying levels of available concurrency.

3.6.4 Load Balance

The utter lack of persistence in the sizes of the sequential grains of computation and the constantly varying amount of available parallelism imply that a static *a priori* partition of work across different compute objects (or processors) will not ensure high utilization of the compute resources. It also precludes the use of any persistence-based dynamic load balancing solutions. Hence, our designs adopt pull-based or stealing-based load balancing techniques to ensure utilization. To avoid idle time, a parallel design must maintain pools of available work that can be doled out upon pull requests.

3.6.5 Solver Libraries Maintain Internal State

Unlike other numerical libraries, LP solvers maintain internal state across calls. They maintain the optimal basis of the previous problem that was solved. Most use cases for such solvers involve iterating over a problem with repeated calls to the library. Typically, each call supplies only mildly modified inputs as compared to the previous invocation. In such cases, the search for an optimum can be greatly sped up by starting from the previous solution. Hence, it is highly desirable to retain this internal state across calls as it greatly

shortens the time to solution. This is known as a "warm" start or "advanced" start.

The two-stage optimization problems that interest us follow this pattern too. There are many iterations (rounds) to converge to a solution. In Stage 1, each iteration only adds/deletes a few constraints on the feasible search space. In Stage 2, the coefficient matrix of the LP remains the same, and only the right-hand sides of the constraints are modified across calls. A more detailed discussion on the impact of advanced starts can be found in Chapter 2.

Hence, it is beneficial to (a) allow all the solver library instances in the parallel execution to maintain state across calls and, (b) to maintain an affinity between the solvers and the problems that they work on across iterations. It is desirable to pick a parallel programming paradigm that will permit encapsulating and managing multiple solver instances per processor.

3.6.6 Concurrency Limited by Library Memory Footprint

The lowest levels of the BnB tree that have not been pruned constitute the "front" of exploration. The number of vertices on this front at any given instant represents the maximum available concurrency in exploring the tree. Each vertex on this front represents a unique combination of branching constraints. Since each vertex goes through multiple iterations (rounds), it is desirable to exploit warm starts for each vertex. This can be achieved by assigning one solver instance for each vertex that is currently being explored. However, LP solvers have large memory footprints. The memory usage required for a LP solver instance for 3t, 5t, 10t, 15t are 50MB, 100MB, 230MB, 950 MB, respectively in Stage 1 and 10MB, 15MB, 30MB, 45MB, respectively in Stage 2. This implies that the number of solver instances is limited by available memory, and can be substantially smaller than the number of vertices in a large BnB search tree.

The actual subset of vertices on the front that are currently being explored are known as "active" vertices. The parallel design should account for the memory usage by solver instances, carefully manage the number of active vertices, and expose as much parallelism as permitted by memory constraints.

3.6.7 Stage 2 Feedback Can Be Shared Across the BnB Tree

While the set of branching constraints for each vertex are unique to it, the cut constraints from Stage 2 are not. The branching constraints influences the candidate allocations that are generated in Stage 1. These, in turn, only affect the right hand sides in the Stage 2 LPs, which simply alters the objective function in dual of the Stage 2 LP. The dual polytope of the Stage 2 LPs remains the same across all the vertex in the BnB tree. This implies that the dual optimal solutions obtained in Stage 2 for a candidate solution from the Stage 1 LP of a given vertex, are all valid dual extreme points for any vertex in the BnB tree. Hence, the Benders cuts that are generated from the Stage 2 LPs remain valid irrespective of the branching constraints imposed on a vertex, implying that cuts generated from evaluating scenarios for a given vertex are also valid for all vertices in the BnB tree.

This observation provides a powerful solution to increasing the exposed parallelism while remaining within the memory usage constraints. Since cuts can be shared across vertices, two vertices only differ in the branching constraints unique to them. By applying this delta of branching constraints, a Stage 1 LP solver instance can be reused to solve a Stage 1 LP from another vertex. Solver libraries typically expose API to add / remove constraints. Hence, it becomes possible to reuse a single solver instance to interleave the exploration of multiple BnB vertices. We can simply remove branching constraints specific to the vertex that was just in a Stage 1 LP solve, and reapply constraints specific to another vertex that is waiting for such a Stage 1 solve. This permits exploring more vertices than the available number of solver instances, and also retains the ability to exploit warm starts for Stage 1 LP solves.

The reasoning presented here also implies that the same Stage 2 solver instance can evaluate scenarios across multiple vertices. Hence, we can share both Stage 1 and Stage 2 solvers.

3.6.8 Total Amount of Computation is Variable and Perturbable

The total amount of computation performed to complete the BnB exploration depends on the number of BnB vertices explored and the number of Stage 1–Stage 2 rounds for each

vertex. Unlike traditional iterative HPC algorithms, this total work required is variable and not known a priori. This is compounded by the fact that the shape and size of the BnB tree is easily perturbed. The number of vertices explored depends on the branching and pruning decisions during the exploration. Any factor that affects these decisions can alter the time to solution.

Incumbent Ordering

A parallel exploration of the BnB tree implies that even if the explored trees are identical across two runs, the order in which incumbent solutions are generated can vary slightly because of LP solve times, system noise, network interference in message communication, etc. This order affects the pruning of vertices from the tree. Some cases might even cause a slightly worse incumbent to prune a vertex that would have yielded a slightly better incumbent (but within the pruning threshold) simply because the worse incumbent was generated slightly early on another processor.

Degeneracy

Degeneracy occurs when the same extreme point on the feasible space polytope can be represented by several different bases. When this happens at the optimal extreme point, there can multiple dual optimal solutions. LPs often have degenerate solutions. While solving LPs, depending upon the starting point of the simplex method, one can end up with different solutions. If we share solver resources in an attempt to circumvent memory limitations, we cause an LP solve to start with an internal state that was the result of the previous LP solve for a different vertex. Thus, sharing solvers can yield different solutions to an LP depending on the order in which vertices use the shared LP solver instance. This can happen in both Stage 1 and Stage 2. Different LP solutions can impact the branching decisions under that vertex in the BnB tree. This reasoning implies that sharing LP solver instances can lead to different BnB tree structures.

3.6.9 Better Utilization ≠ Better Performance

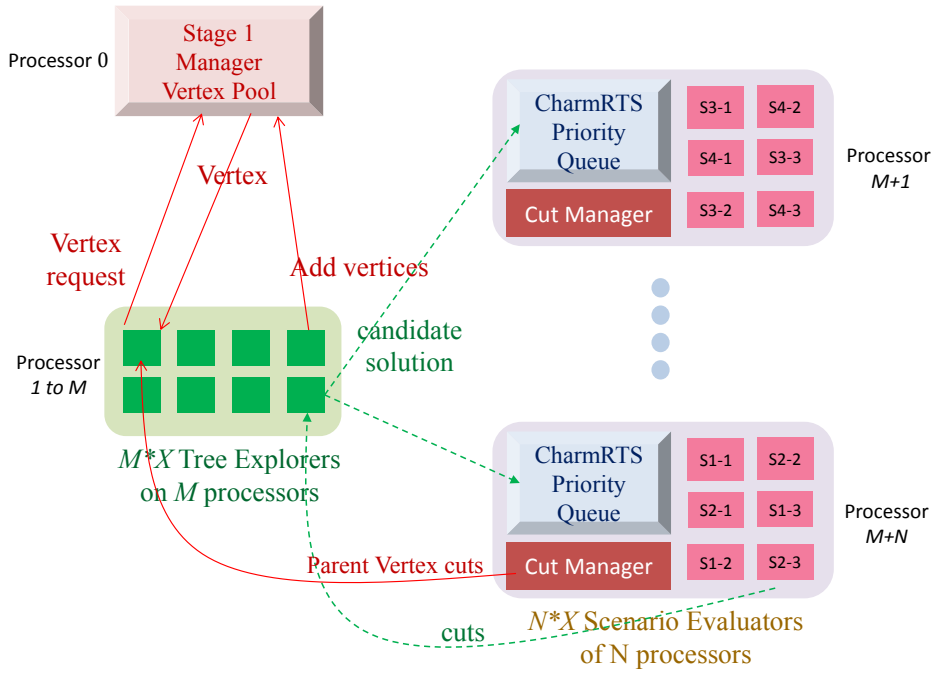
For many parallel, HPC applications, load balance ensures minimal overall compute resource idle time, and hence results in better performance by maximizing the rate of computations. However, parallel, BnB search confounds such thinking. Indeed, reducing idle time by eagerly exploring as much of the tree as possible might be counter-productive by using compute resources for exploring sub-trees that might have been easily pruned later.

3.7 Parallel Programming Model

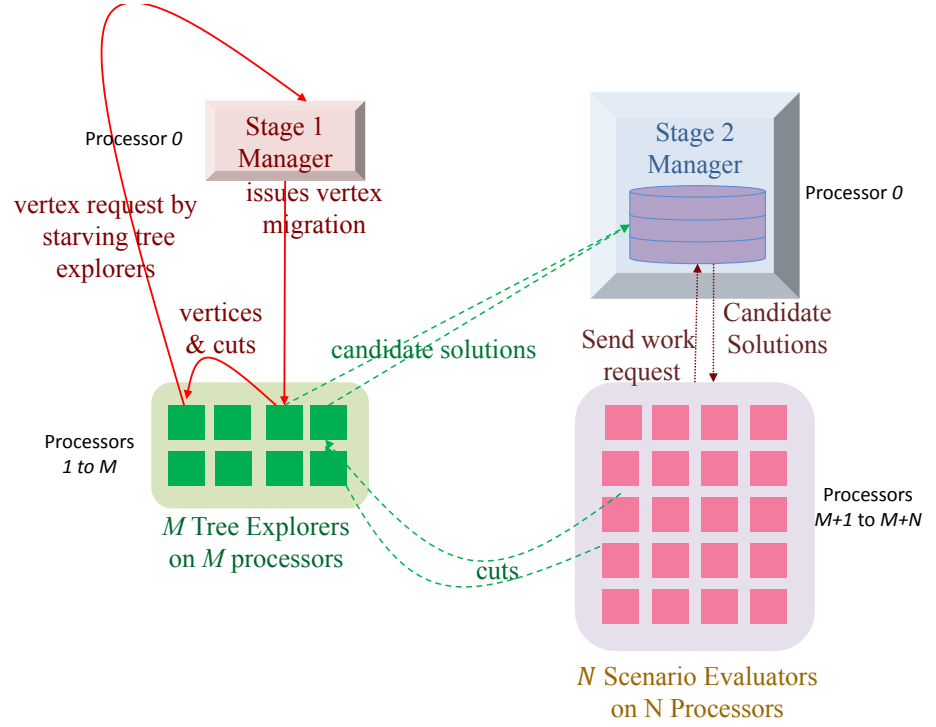
The designs that we discuss here are implemented in an object-based, sender-driven parallel programming model called Charm++ [62,63] . Charm++ is a runtime-assisted parallel programming framework in C++. Programs are designed using C++ constructs by partitioning the algorithm into classes. Charm++ permits elevating a subset of the classes and methods into a global space that spans all the processes during execution. Parallel execution then involves interacting collections of objects, with some objects and methods being invoked across process boundaries. Data transfer and messaging are all cast in the form of such remote method invocations. Such remote methods are always one-sided (only sender initiates the call), asynchronous (sender completes before receiver executes method), non-blocking (sender's side returns before messaging completion) and also do not return any values (remote methods are necessarily of void return type). Charm++ supports individual instances of objects, and also collections (or chare arrays of objects). Some features of Charm++ that enable the designs discussed in this chapter:

One-sided messaging helps express and exploit the synchronization-free parallelism found in parallel BnB. Extracting performance in a bulk synchronous programming model can be quite challenging.

Object-based expression of designs facilitate the easy placement and dynamic migration of specific computations on specific processors. It also permits oversubscribing processors with multiple objects to hide work-starvation of one with available work in another.



(a) Design A: Every vertex in the BnB tree performs its own iterative, two-stage linear optimization in isolation from other vertices. There are X Tree Explorers on every Stage 1 processor. S1-3 corresponds to the *Scenario Evaluator* for scenario 1 of *Allocation Generator* 3, and similarly others.



(b) Design B: BnB vertices share resources (Stage 1 and Stage 2 solver objects) and constraints on the feasible solution space (cuts) while iteratively solving the nested two-stage linear programs.

Figure 3.5: Schematics of the parallel components in the two design variants

Non-blocking reductions for any required data collection, notifications etc avoids any synchronization that could be detrimental to performance. A programming model well suited to such problems, should unlock all the available parallelism without bridling it with synchronization constructs.

Prioritized execution allows us to simply tag messages with appropriate priorities and allow the Charm++ runtime system to pick the highest priority tasks from the available pool.

3.8 Design A: Each BnB Vertex is an Isolated Two-Stage LP

3.8.1 Stage 1 Tree Explorers

A collection of compute objects (chare array in Charm++) explore the BnB tree in parallel. Each *Allocation Generator* hosts an instance of the Gurobi LP library. Tree Explorers are constrained to explore only one vertex at a time. Whenever a new vertex is picked, the library instance is reset and reloaded with a known collection of cuts from an ancestor vertex. When the vertices are waiting on Stage 2 feedback, the *Allocation Generator* idles. The processors dedicated to exploring the tree are oversubscribed by placing multiple Tree Explorers on each. The Charm++ runtime automatically overlaps idle time in one object with computation in another object by invoking any objects which are ready to compute. In the situation when multiple objects on a processor are ready to compute, execution is prioritized according to the search policy. This is indicated to the Charm++ runtime by tagging the messages with a priority field. This field can be an integer (tree depth), a fraction (bounds / cost), or a bitvector (vertex identifier).

3.8.2 Cut Dump Manager

Solving stochastic LP at each vertex from scratch can be very expensive as this potentially repeats a lot of avoidable Stage 1–Stage 2 rounds to regenerate all the cuts that would have

been generated by vertex's ancestors. Each vertex, therefore, starts with the cuts of its parent. This significantly reduces the number of rounds required to optimize the stochastic LPs.

We precompute the available memory on the system and corral a portion of it for storing dumps of cut collections. Whenever a vertex converges, we extract its collection of cuts from the library instance and store it in the available memory. The dump is tagged with the bitvector id of the vertex. Whenever an immediate child of this vertex is picked for exploration, the parent's cut collection is retrieved and applied to the library instance. Once both children of a vertex are explored, the parent's dump is discarded. Hence, at any given time, the number of cut dumps stored is a linear function of the number of vertices on the tree frontier. The cut collection dumps are managed by a third chare collection called the Cut Manager. Objects of this collection are not placed on processors with Tree Explorers in order to keep them reasonably responsive to requests.

3.8.3 *Scenario Evaluators*

Akin to the Tree Explorers, the *Scenario Evaluators* are a collection of compute objects each of which hosts an LP instance. These evaluate the candidate solutions for one or more scenarios and send the generated cuts directly back to the *Allocation Generator* that hosts the specific BnB vertex. We dedicate a collection of *Scenario Evaluators* to each *Allocation Generator*. Each *Allocation Generator* object interacts directly with its collection of *Scenario Evaluators*. We place these multiple collections of *Scenario Evaluators* on the same subset of processors. Idle time in one is overlapped with computation in another. The execution of Stage 2 computations for the most important vertices is again achieved by simply tagging the messages with the priorities of the corresponding vertices.

3.8.4 Load Balancing

When a *Allocation Generator* converges to an LP solution on a vertex, on its currently assigned vertex, further work is generated only if the vertex branches. In this case, the

children are deposited with the Stage 1 Manager vertex queue. After every Stage 1 LP convergence, the *Allocation Generator* requests the Stage 1 Manager for a new vertex to work on. The Stage 1 Manager dequeues the highest priority vertex from its vertex queue and sends it to requesting *Allocation Generator*. Thus all Tree Explorers always pull from a global pool of available work. This effectively balances Stage 1 load and also ensures a globally prioritized tree exploration.

3.9 Design B: BnB Vertices Share Cut Constraints, Tree Explorers and *Scenario Evaluators*

3.9.1 Stage 1 Tree Explorers

Each *Allocation Generator* object stores and explores several vertices. The vertices are divorced from the library instance by separately storing the set of branching constraints specific to each vertex. Every object maintains a set of private vertex queues to manage the vertices in different stages of their lifespan. When the LP library completes a solve, the next vertex is picked from a "ready" queue. This queue is prioritized according to the search policy (depth-first, most-promising-first, etc). The delta of branching constraints between the previously solved vertex and the currently picked vertex is applied to the LP library to reconstruct the Stage 1 LP for the newly selected vertex. The Stage 1 LP is then solved to yield a new candidate solution for the current vertex. This candidate solution is sent for evaluation against the set of Stage 2 scenarios and the vertex is moved to a "waiting" queue. The compute object repeats the process as long as there are vertices waiting to be solved in the ready queue. Vertices move back from the waiting queue into the ready queue when the cuts from evaluating all the scenarios for the generated candidate allocation are sent back to the *Allocation Generator*. When a vertex "converges", that is, when the optimal fractional solution to the stochastic LP described by the vertex is found, it is "retired" by either pruning it or branching further.

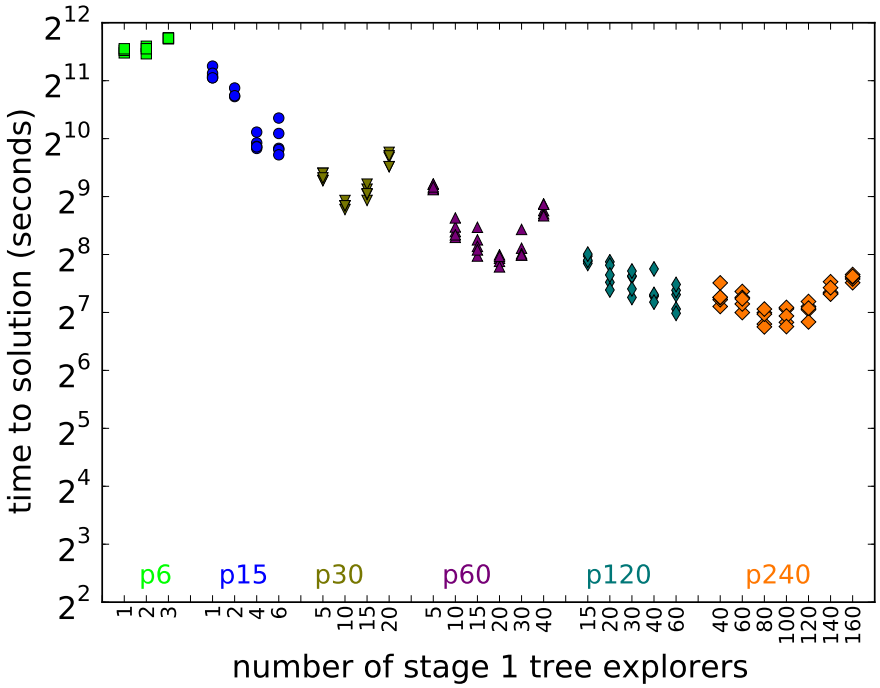
The number of *Allocation Generator* objects is smaller than the number of vertices in the search tree. We also find from experiments that it is sufficient for the number of such Tree

Explorers to be a small fraction of the number of processors in a parallel execution.

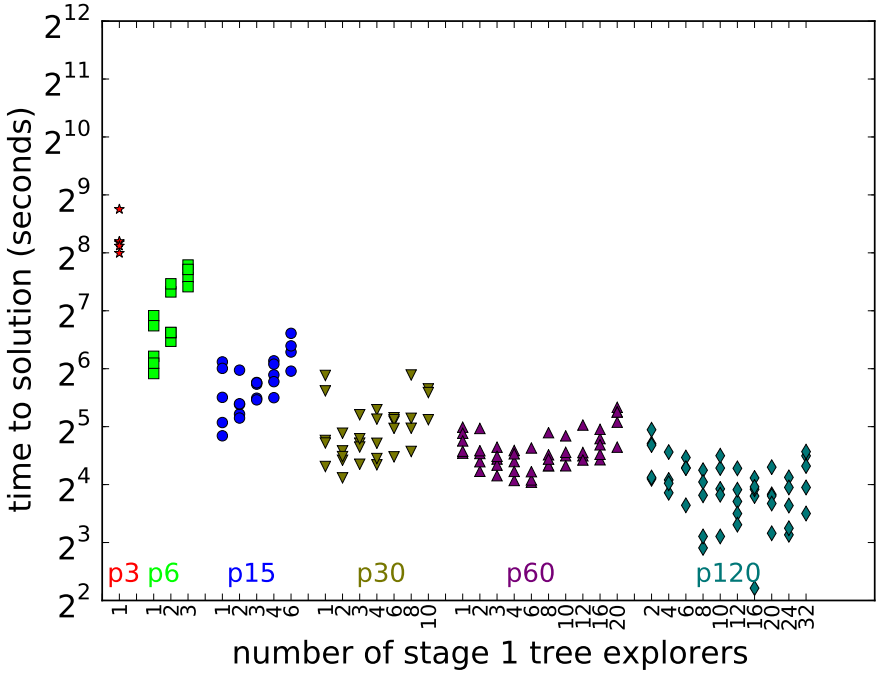
Cuts generated from a scenario evaluation can be used in all the Stage 1 LPs. However, we have found that this results in a deluge of cuts added to the Stage 1 library instances. In earlier work [64], we have observed a strong correlation between the number of cuts added to a library instance and the time taken for the LP solve. Hence, instead of sharing the cuts across the entire BnB tree, we share cuts only across vertices hosted by a single *Allocation Generator*. Cuts generated from the evaluation of a candidate solution are hence messaged directly to the solver hosting the corresponding vertex. However, the collection of cuts accumulated in a library instance continues to grow as more vertices are explored. Since some of these may be loose constraints, we discard them to make space for newer constraints. If these constraints are required again later on, they will be regenerated by the algorithm. We implement bookkeeping mechanisms that track the activity of cuts and retires cuts identified as having low impact (longest-unused, most-unused, combination of the two, etc). This maintains a fixed window of recent cuts that are slowly specialized to the collection of active vertices sharing that library instance. The impact of cut retirement on solve times is illustrated in [64].

3.9.2 Stage 2 Manager

Candidate solutions from the Tree Explorers are sent to a Stage 2 Manager object. This object helps implement a pull-based work assignment scheme across all *Scenario Evaluators*. To do this, it maintains a queue of such candidate solutions and orchestrates the evaluation of all scenarios for each candidate. In order to remain responsive and ensure the quick completion of pull requests, the object is placed on its own dedicated core and other compute objects (which invoke, long, non-preempted LP solves) are excluded from that core. The Stage 2 Manager ensures that each *Allocation Generator* gets an equal share of Stage 2 evaluation resources by picking candidates from Tree Explorers in round-robin fashion.



(a) Design A



(b) Design B

Figure 3.6: Strong Scaling for the 3t-120 model. Colors correspond to the scale (number of processors) e.g. p6 is for 6 processors, p15 for 15, and so on. At each scale, runs for performed for varying number of Tree Explorers. For each configuration 5 trials are performed to measure the variability

3.9.3 Stage 2 *Scenario Evaluators*

In this design variant, all Tree Explorers share the same collection of *Scenario Evaluators*. A *Scenario Evaluator* request the Stage 2 Manager for candidate Stage 1 solutions and evaluate these solutions for one or more scenarios. Upon evaluation, they send the generated cuts directly back to the *Allocation Generator* that hosts the specific BnB vertex. This pull-based scheme ensures good utilization of the processors hosting *Scenario Evaluators*, and also balances the scenario evaluation workload across all the Stage 2 processors. Given that the Stage 2 LP solve times are typically much larger than the messaging overhead to obtain work, the pull-based approach has negligible overhead.

3.9.4 Load Balancing

A *Allocation Generator* maintains a private list of vertices. It regularly updates the Stage 1 Manager of the total number of vertices that it currently has. Whenever a *Allocation Generator* runs out of work i.e. has evaluated all its vertices, it requests the Stage 1 Manager for work. Stage 1 Manager selects the most loaded *Allocation Generator* and sends it a request to offload half of its workload to the starving *Allocation Generator*. The max loaded *Allocation Generator* sends half of its vertices and its LP solver state (cuts) to the starving *Allocation Generator*.

3.10 Performance and Analysis

All experiments were performed on the 300 node (3600 cores) Taub cluster installed at University of Illinois. Each node has Intel HP X5650 2.66 GHz 6C processors and 24GB of memory. The cluster has a QDR Infiniband network communications with a Gigabit Ethernet control network. We used Gurobi [41] as the LP solver.

As noted in Figure 3.5a and 3.5b, Stage 1 Manager and Stage 2 Manager (in Design B) are placed on processor 0. *Allocation Generator* and *Scenario Evaluator* are place on disjunct set of processors, with *Allocation Generator* objects placed on processors 1 through M, and *Scenario Evaluator* objects placed on processors $M+1$ through N, where $M+N+1$

Table 3.2: Average Stage 1 LP solve time comparison between Design A and Design B of PSIPS

Model	Stage 1 LP solve time (s)	
	Design A	Design B
3t-120	0.38	0.04
3t-240	0.98	0.08
5t-120	2	0.25

is the total number of processors. We use depth first search as the BnB vertex prioritization policy, where depth is determined by the total number of branching decisions taken on the path from the root node to that vertex. For vertices with the same depth, one with a smaller lower bound is given higher priority.

3.10.1 Variability in Execution time

As discussed in Section 3.6.8, both designs suffer from variability in execution times across runs with identical configurations. Design A ensures that the branching order remains the same across all runs of the same model. However, as discussed in 3.6, the chronology of incumbent discoveries might vary slightly across runs, thereby causing different pruning decisions and different BnB tree sizes. Design B, in addition, has another source of variation. The order in which the Stage 1 and Stage 2 solves are done can alter the LP solutions to the same problem because of the combined effect of advanced start and degenerate Stage 1, Stage 2 LPs. This changes the branching decisions and hence different trees are generated. This can cause significant variation in the time to solution.

Figure 3.6 plots the performance of the two designs for 3t-120. On x-axis is the number of Tree Explorers. Each color corresponds to a scale e.g. p3 is for 3 processors, p6 for 6, and so on. At each scale, we measured the performance for varying number of Tree Explorers. For every configuration, we did 5 trials to measure the variability. The time to solution in these trials is plotted with markers in the same vertical line. Design A has much less variability as the markers are very close to each other as compared to the Design B, where performance varies even by an order of magnitude in some cases. In Figure 3.7, we plot the BnB trees

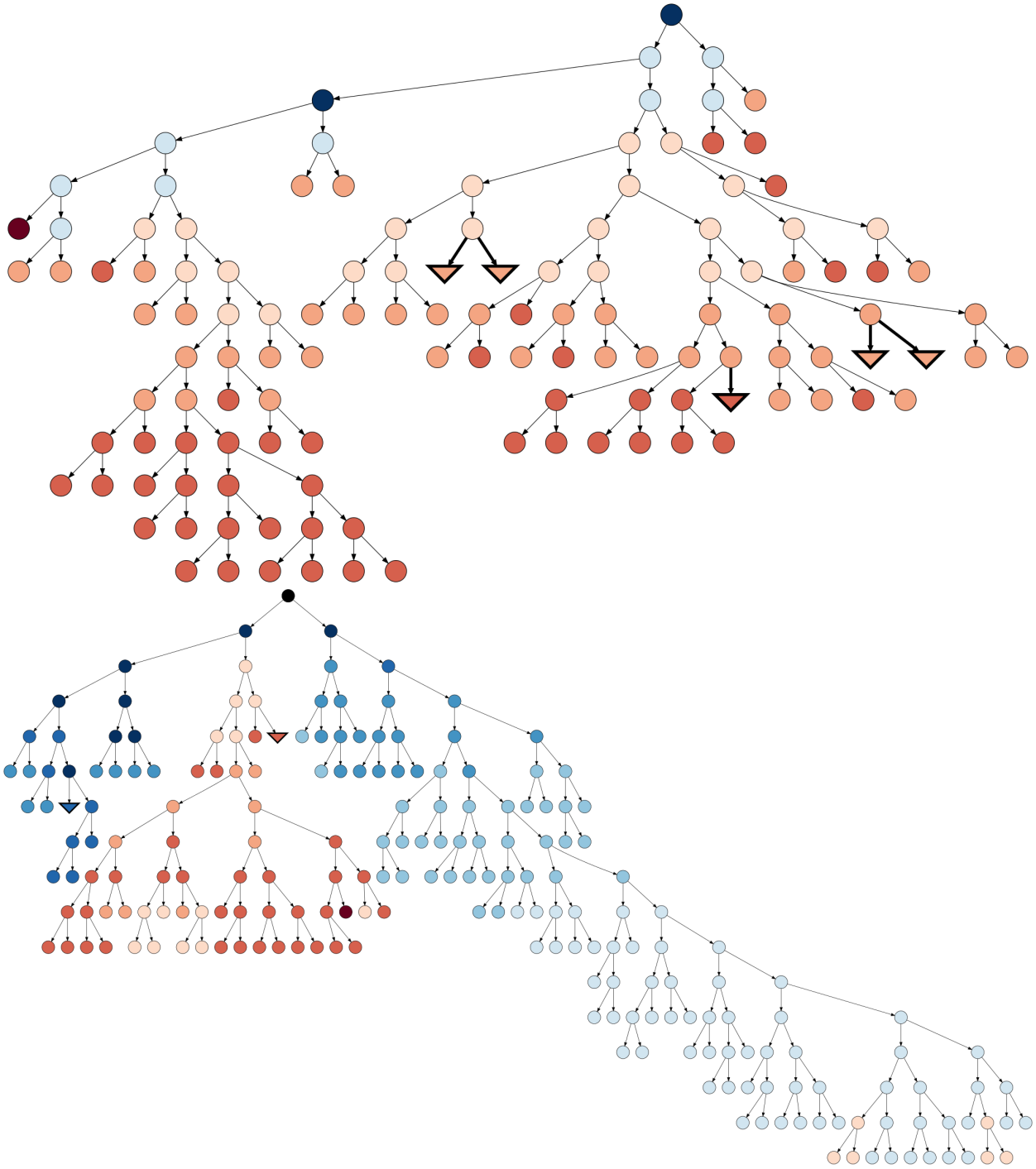
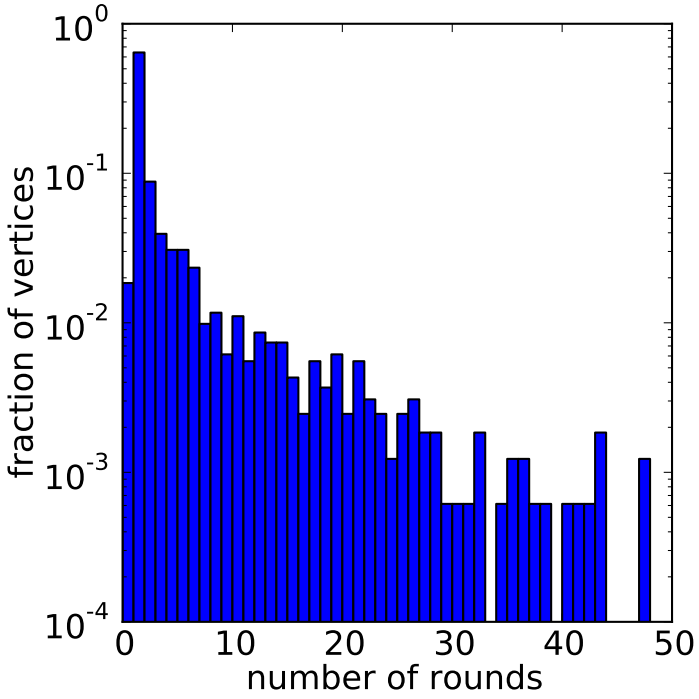
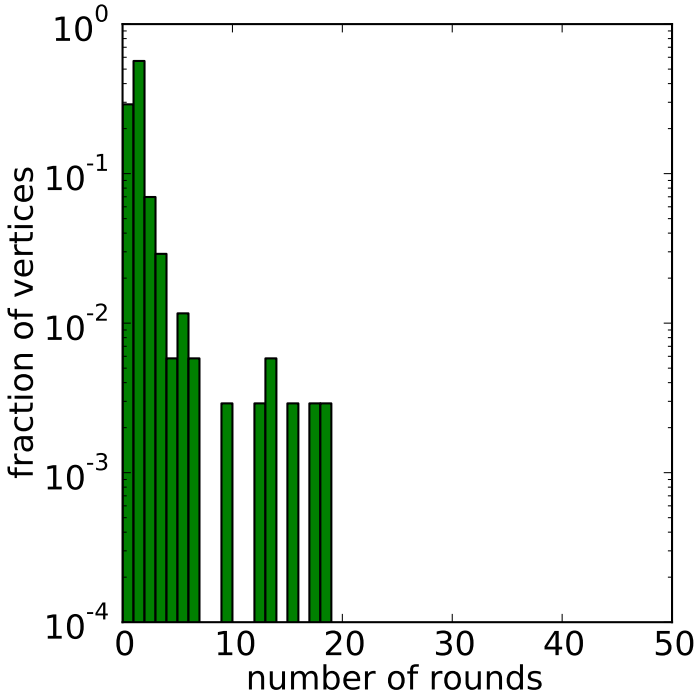


Figure 3.7: The Branch-and-Bound trees from two identically configured executions of Design B for the 3t-120 dataset. The trees from the two trials are significantly different because of branching on different variables in different orders. This explains the large variation in performance across trials. Triangles represent integer solutions (incumbents), while the vertices are colored by the value of bound



(a) Design A



(b) Design B

Figure 3.8: Analyzing the cause of slower performance of Design A as compared to Design B. (a) and (b) plot the histogram of the number of rounds taken to solve the stochastic LP at the BnB tree vertices in the 5t-120 model.

explored in two identically configured executions of Design B on the 5t-120 model. This explains the large variation in performance of Design B.

3.10.2 Performance Comparison

The number of Tree Explorers at any given execution scale has a significant effect on the performance. Expectedly, increasing the number of Tree Explorers too much inundates Stage 2 with work and deteriorates performance. We have also ascertained that the concurrent execution of several Stage 1 LPs on the same compute node of the machine increases the individual solve times because of memory bandwidth limitations. From Figure 3.6 it is clear that Design B, despite having high variability has significant advantage in terms of solution speed over Design A. This advantage is two-fold. First, the number of rounds to achieve convergence at the tree vertices is much smaller in Design B. This effect is shown in Figure 3.8a, Figure 3.8b in which we plot a histogram of the number of rounds vertices take to converge in the two designs. This difference can be attributed to the difference in the set of Benders cuts that are maintained by the two designs. In an effort to maintain repeatability, Design A always starts with the cuts from the parent vertex. On the other hand, Design B uses the most current set of cuts resident on the processor being used. This means that Design B has access to cuts generated in different parts of the tree and is therefore likely to have more cuts that are binding and thus speed up convergence. Secondly, the stage 1 linear programs also take less time to solve in Design B (Table 3.2). Since in Design A, every new vertex starts with a fresh start of the Gurobi library instance, a significant number of simplex iterations are required to optimize the LP in the first round for each vertex. Conversely, Design B always uses advanced start and the most recent cut set. The LPs differs from the previous vertex LP only in the few branching constraints and thereby, the LP solves very quickly using advanced start.

Even though Design A has better repeatability, the worst performance using Design B is better than the best performance using Design A. Therefore, Design B is the design of choice because of quicker time to solutions. Additionally, Design A suffers from large memory requirements for cut dump collection, which can become a bottleneck for larger data sets in

Table 3.3: Cumulative distribution of trials of Design B for a target minimum parallel efficiencies on 3t-120 (baseline: 3 cores)

Efficiency(%) >	**Number of processors**				
	6	**15**	**30**	**60**	**120**
100	1.0	0.9	0.95	0.066	0.0
90	1.0	1.0	0.95	0.066	0.2
80	1.0	1.0	0.95	0.466	0.4
70	1.0	1.0	1.0	0.733	0.4
60	1.0	1.0	1.0	0.866	0.6
40	1.0	1.0	1.0	1.0	1.0

Table 3.4: Cumulative distribution of trials of Design B for a target minimum parallel efficiency on 5t-120 (baseline: 3 cores)

Efficiency(%) >	**Number of processors**				
	6	**15**	**30**	**60**	**120**
100	0.95	0.7	0.8	0.2	0.0
90	0.95	0.75	0.85	0.4	0.0
80	0.95	0.8	0.85	0.4	0.0
70	1.0	0.8	0.9	1.0	0.2
60	1.0	0.85	0.9	1.0	0.6
40	1.0	0.85	1.0	1.0	0.8

which the tree frontier becomes very large before the solution is found.

3.10.3 Performance of Design B

Using large-scale parallel computing for an application is advantageous when it is guaranteed that running the application on more processors will give faster times to solution. Unlike typical scientific iterative applications, Design B for this application suffers from large variability in execution times for runs with identical configurations, which makes it difficult to measure its parallel efficiency. We therefore need a different method to quantify its parallel efficiency in the wake of variation. Our method is to measure the probability of getting a certain parallel efficiency. To measure the performance of Design B with this metric, we did 20 trials of Design B with each of 3t-120 and 5t-120 datasets. In Table 3.3 and Table 3.4, first column has the parallel efficiencies. Rest of the columns report, at different scales, the fraction of trials that achieved greater efficiency than the corresponding entry in the first

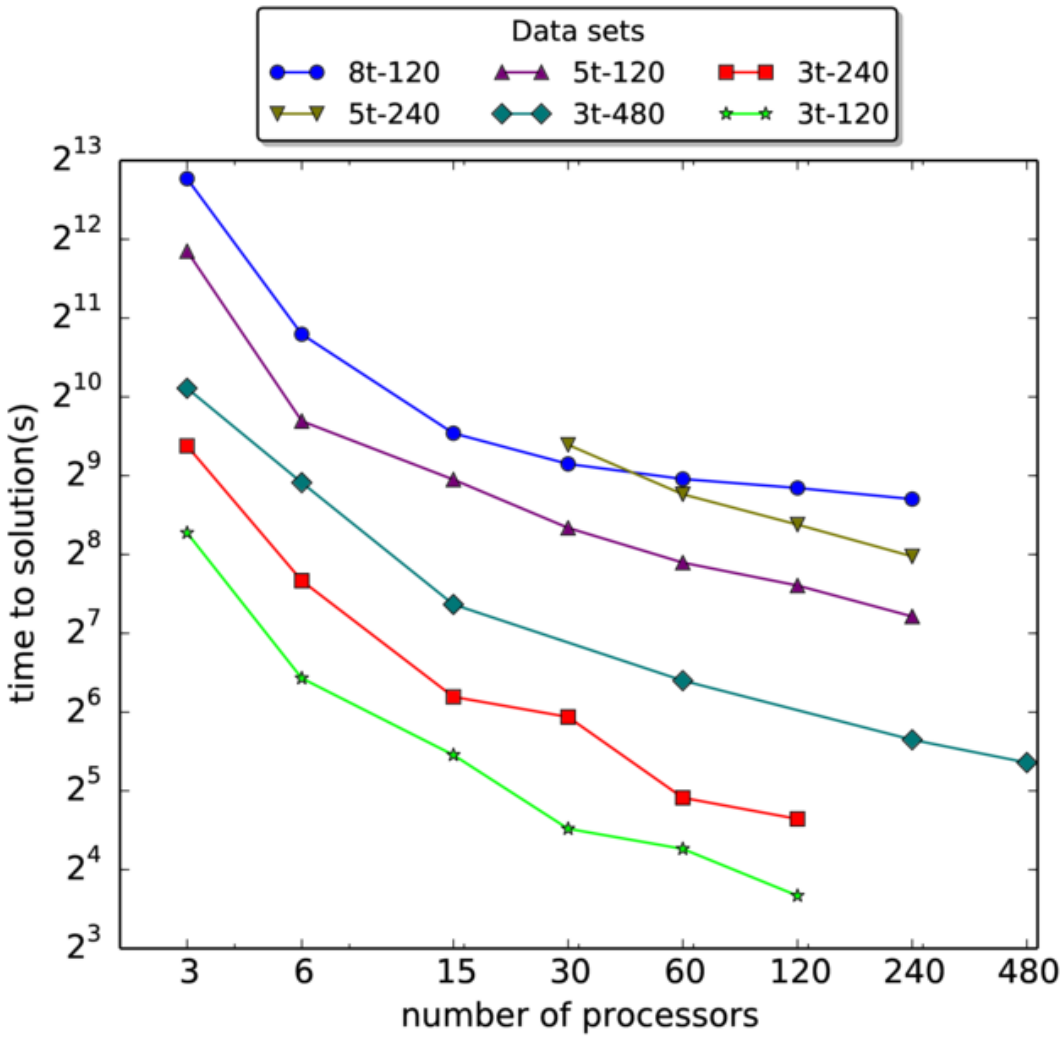


Figure 3.9: Scaling of various models for Design A and Design B

column. For example, for 3t-120, the parallel efficiency was greater than 90% in 95% of the trials at 6 processors and in 75% of the trials at 15 processors. These results show that in majority of the cases efficiency was greater than 40% at all scales for both the datasets. Also note the super linear speedup in some cases. As compared to Gurobi's typical efficiency of 10% for IPs [53], our algorithms yield significantly higher parallel efficiencies even at larger scales.

We further report the scaling of Design A and Design B in Figure 3.9. We identify the best performing *Allocation Generator* count at each scale by comparing the average time to solution across 5 trials. Average times to solutions for these *Allocation Generator* counts are presented in Figure 3.9 for several datasets. We get very good incremental speedups on up to 480 processors for several datasets. The scaling at large scales is limited by the root vertex optimization, which takes many rounds to converge as compared to the other vertices. During root node optimization there is only 1 vertex and hence no Stage 1 parallelism. Scaling at large scales is additionally limited by the critical path to reach the optimal solution.

3.11 Summary

We have discussed and presented several factors that influence the design and performance of parallel, two-stage stochastic integer programs solved using Branch-and-Bound. We have also presented two designs that prioritize different factors: 1. a nested parallel decomposition that solves each BnB vertex in isolation and 2. a design variant that shares LP library solvers as well as Stage 2 feedback across BnB vertices. The interplay between some of the factors like memory usage, solver sharing, degeneracy and tree structure are borne out by the performance results for both these designs on multiple datasets. Sharing solvers and cuts results in more variable, yet better performance. We also show strong scaling from 6 cores up to 480 cores of a dual hex-core, 2.67 GHz, Intel Xeon cluster. Because of the inherent variability in the amount of computation required, we also report the spread in performance by tabulating the fraction of trials that achieved various parallel efficiencies. We believe these are noteworthy results for strong scaling such an unconventional problem.

However, there is still a need for further characterizing the behavior of parallel stochastic

integer programs; and for further research into techniques for improved scalability. We feel our experiences and findings are a useful addition to the literature and can seed further work in this direction.

CHAPTER 4

Split-and-Merge Method for Accelerating Convergence of Stochastic Linear Programs

Stochastic program optimizations are computationally very expensive, especially when the number of scenarios are large. Complexity of the focal application, and the slow convergence rate add to its computational complexity. In this chapter, we propose a split-and-merge (SAM) method for accelerating the convergence of stochastic linear programs. SAM splits the original problem into subproblems, and utilizes the dual constraints from the subproblems to accelerate the convergence of the original problem. Our results are very encouraging, giving up to 74% reduction in the optimization time.

4.1 Introduction

In this chapter, we focus on the stochastic linear programs, that is, problems that have linear variables and constraints both in Stage 1 and Stage 2. However, the proposed approach can be extended as it is to stochastic integer programs that have mixed-integer variables in Stage 1. We leave the evaluation of the proposed method to stochastic integer programs for future work.

As we have studied before, the usual method of solving stochastic linear program uses the multicut Bender's method [3]. However, the number of cuts can become very large quickly,

particularly for problems with large number of scenarios. In most real world applications, the number of uncertain parameters are large, and therefore the number of scenarios are also very large. In addition to the complexity of the focal application, factors such as the number of Stage 2 evaluations, number of rounds it takes to converge to optimality (within the user-specified convergence criteria), and the size of the Stage 1 linear program which increases with the increase in number of scenarios, add to the computational complexity of the stochastic programs.

The convergence of the multicut method can be very slow in cases when the number of Stage 1 variables are large and/or there are large number of Stage 2 scens. This chapter focuses on accelerating the convergence of multicut Bender's method. Equation 4.1 shows the Stage 1 program after r rounds.

$$\begin{aligned} \min \quad & cx + \sum_s p_s \theta_s \\ s.t. \quad & Ax \leq B \\ \forall s \text{ and } l \in [1, r], \quad & E_{sl} x + \theta_s \leq e_{sl} \end{aligned} \tag{4.1}$$

where, $E_{sl} + \theta_s \leq e_{sl}$ are the cut constraints obtained from Stage 2 optimization and θ_s is the cost of scenario s.

This chapter considers a scenario split-and-merge approach to accelerate the convergence of multicut Bender's method. In Section 4.2, we do a literature review on stochastic optimization methods and their convergence properties. In Section 4.3, we propose the split-and-merge method for accelerating the convergence of multicut L-shaped method. We corroborate our ideas with results in Section 4.6. Finally, we conclude the chapter with the summary in Section 4.7.

4.2 Related Work

Magnanti and Wong in their seminal paper [65], proposed a method for accelerating Bender's decomposition by selecting good cuts to add to the master problem. A cut $\theta \leq \pi_1^* h + \pi_1^* T x$

dominates or is stronger than the cut, $\theta \leq \pi_2^* h + \pi_2^* Tx$, if $\pi_1^* h + \pi_1^* Tx \leq \pi_2^* h + \pi_2^* Tx$ for all $x \in X$ with a strict inequality for at least one $x \in X$, where π_1^* and π_2^* are any two dual optimal solutions of the degenerate Stage 2 problem. They define a cut as pareto optimal if it has no dominating cut. The corresponding Stage 2 dual optimal solution is called the pareto optimal solution. Given the set of Stage 2 dual optimal solution set $S(x^*)$, the pareto optimal solution (π^p) solves the problem:

$$\min_{\pi \in S(x^*)} \pi h + \pi T x^c$$

where, x^c is a core point of X i.e. $x^c \in$ relative interior of X and $S(x^*) = \{\pi | \pi$ maximizes $Q(x^*)\}$. The downside of this approach is that it requires solving additional optimization problem to identify pareto optimal cuts in every iteration which can trade-off the benefit of reduction in total number of iterations.

Linderoth et al [37] developed asynchronous algorithms for stochastic optimization on computational grids. They use a multicut method and add a cut of a particular scenario to the master program only if it changes the objective value of the proposed model function corresponding to that scenario. This requires solving several additional optimization problems at each iteration to determine the usability of each cut, which can be prohibitive.

Initial iterations in the multicut method are often inefficient because the solution tends to oscillate between different feasible regions of the solution space. Ruszczyński [66] proposed a regularized decomposition method that adds a quadratic penalty term to the objective function to minimize the movement of the candidate solution. Linderoth and Wright [37] use a linearized approach to this idea by binding the solution in a box called the trust region. Trust region method is used to decide the major iterates that significantly change the value of the objective function in each iteration. This requires doing several minor iterations at each major iteration to come-up with a good candidate solution x^k. Trust-region method at minor iterations limits the step-size by adding constraints of the form $||x - x^k||_\infty \leq \Delta$. Heuristics are used to decide and update Δ. The cuts generated during the minor iterations can be discarded without affecting the convergence of the problem.

The Progressive Hedging algorithm proposed by Rockafellar and Wets [29] solves each

scenario independently by introducing lagrangean multipliers for the Stage 1 variables in the objective function of the individual problems. This approach requires search for the optimal lagrangean multipliers which can be computationally prohibitive. In Chapter 2, we proposed clustering schemes for solving similar scenarios in succession that significantly reduces the Stage 2 scenario optimization times by use of advanced/warm start. However, this does not address the slow convergence rate of the problem. Other stochastic program decomposition studies can be found in [67–69].

4.3 Split and Merge Algorithm

In each iteration of the multicut method, as many cut constraints are added to the Stage 1 program as there are scenarios. In the initial iterations of the multicut Bender's method, all the scenario cut constraints are not active in the Stage 1 linear program optimization. This is because few cuts are needed to perturb the previous Stage 1 solution and provide a new candidate solution. Therefore, the cuts from the Stage 2 evaluation of most of the scenarios remain inactive in Stage 1 during the initial iterations of the Bender's method. For such scenarios, similar cuts will be generated in successive iterations, and hence a lot of computation is wasted.

We propose a split-and-merge (SAM) algorithm (Algorithm 2) that divides the scenarios into N clusters $(S_1, S_2, ..., S_n)$.

Algorithm 2: Split-and-Merge (SAM)

```
1  Input: S (set of scenarios), Original Stochastic Program (P)
2  Divide S into n clusters, S_1, S_2, ...., S_n
3  Generate n stochastic programs, P_1, P_2, ...., P_n, with
4   scenarios from S_1, S_2, ...., S_n, respectively
5  Scale scenario probabilities in each of these subproblems
6   such that they sum up to 1
7
8
9  for i in range(1,n):
10   scosts_i = [] #scenario costs
11   cuts_i = [] #scenarios cut constraints
12   while r_i < r or hasConverged(i):
13     x_i = solveStage1(P_i, scosts_i, cuts_i)
14     scosts_i, cuts_i = solveStage2(x_i)
15     r_i=r_i + 1
16   end while
17
18 #wait until all the subproblems have returned
19 cuts = []
20 scosts = []
21 for i in range(1,n):
22   cuts.add(getCutConstraints(P_i))
23
24 #now solve the original problem
25 while not hasConverged(P):
26   x = solveStage1(P, scosts, cuts)
27   scosts, cuts = solveStage2(x)
28 end while
```

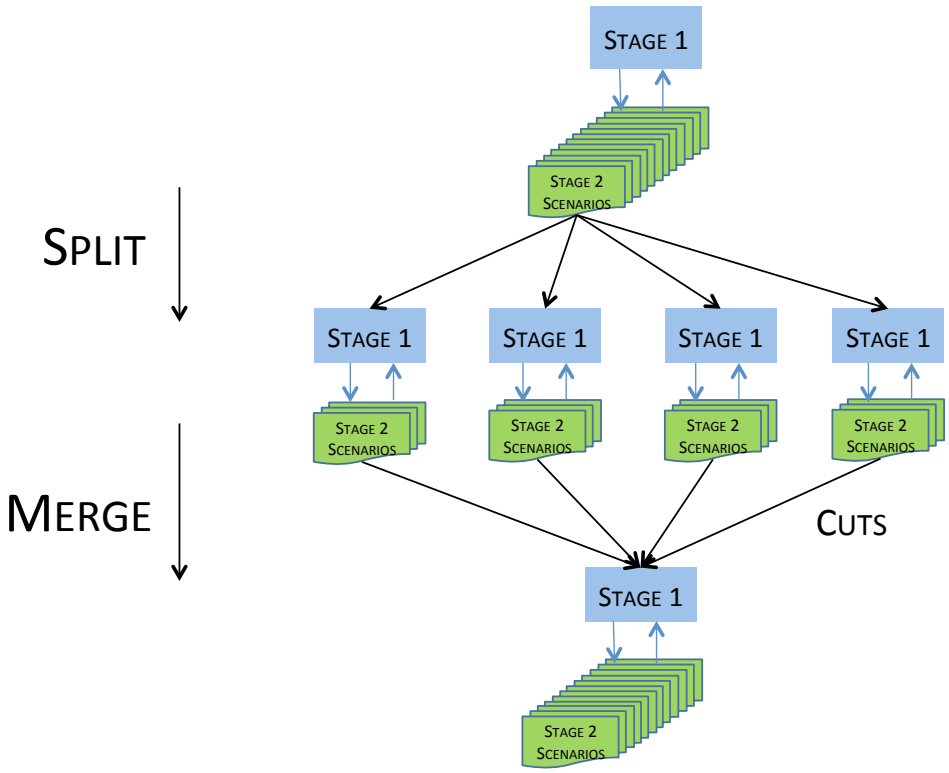


Figure 4.1: Schematic of the Splt-and-Merge (SAM) method

In SAM, n stochastic programs $(P_1, P_2, ..., P_n)$ are created and each of these is assigned one cluster of scenarios (lines 3-4). Probabilities of the scenarios in each of these subproblems are scaled up so that they add up to 1 (lines 5-6). We then apply the Bender's multicut method to these n stochastic programs independently of each other (lines 8-16). Bender's decomposition is applied to these subproblems for a fixed number of rounds (r) or till the subproblem has converged to optimality, whichever is the earliest (line 12). Once this criteria has been met for all the subproblems, the cut constraints from these problems are collected (lines 21-22). The cuts from subproblems are also valid for the original problem with all the scenarios. These cuts are used as the initial set of cut constraints for applying the multicut Bender's method to the original stochastic linear program.

There are several benefits of this approach. The chances of a scenario having active cuts is higher in the subproblems because of the smaller number of scenarios present in the subproblems. Scenario cut activity helps in generating newer and different cuts for those scenarios, and thus doing more useful work, as compared to the original problem in which

most of the scenarios remain inactive in the initial iterations.

Stage 1 optimization is often a serial bottleneck in Bender's decomposition, especially when the number of scenarios is large. In the decomposition approach, the number of scenarios per subproblem are much smaller than the original problem, which speeds up the Stage 1 optimization and thus the candidate solutions for Stage 2 evaluation become available much earlier. Additionally, this also gives an opportunity to have parallelism in Stage 1, in addition to the obvious Stage 2 parallelization available in stochastic linear programs. These subproblems being independent of each other, can be optimized in parallel in Stage 1.

4.4 Parallel Design of SAM

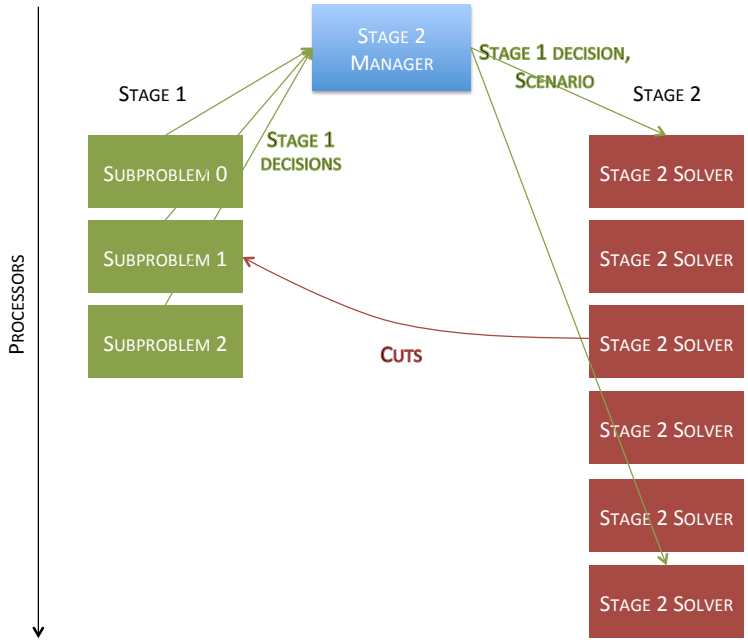


Figure 4.2: Parallel design for Split-and-Merge (SAM) method

Parallel design for implementation of the SAM method is depicted in Figure 4.2. Processor 0 is dedicated for Stage 2 manager. Stage 2 manager is responsible for receiving Stage 1 decisions from Stage 1 solvers and assigning them to Stage 2 solvers when they request for work. Stage 1 of each of the subproblems is assigned to different processors so that they can

run in parallel. Stage 2 solvers are shared by all the subproblems. When Stage 2 manager assigns work to a Stage 2 solver, it sends the Stage 1 decision for one of the subproblem and also the scenario that it should solve. Since Stage 2 solver execution is interleaved with Stage 1 solver on the same processor, Stage 1 solver execution is given higher priority so that a subproblem's progress is not stalled.

4.5 Experimental Setup

Our experiments of the proposed SAM approach are based on the military aircraft allocation problem (Chapter 6). As in Chapter 3, the parallel implementation of the SAM method is done in Charm++ parallel programming language and runtime system.

For our experiments, we consider two stochastic programs, the details of which are given in Table 5.2. We consider two variants of each of the two stochastic programs in Table 5.2, one with 120 scenarios and another with 1000 scenarios. These models are named as 8t120s, 8t1000s, 10t120s and 10t1000s, respectively. The total number of variables and constraints in these models is given in Table 5.3.

Table 4.1: Stage 1 LP and Stage 2 LP sizes of the stochastic linear program datasets

Test	1st Stage		2nd-Stage Scenario		Nonzero Elements		
Problem	Vars.	Constrs.	Vars.	Constrs.	A	W_i	T_i
8t	216	144	20944	13378	456	66881	252
10t	270	180	25573	16572	570	82797	308

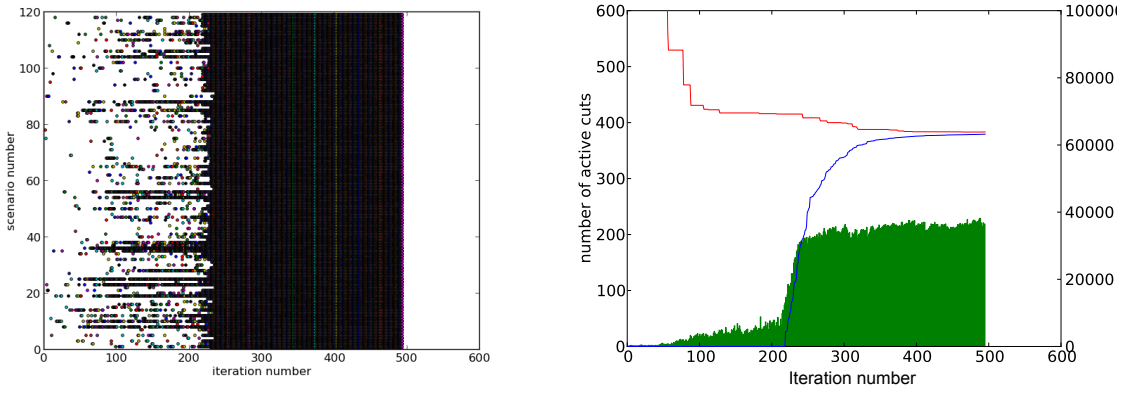
Table 4.2: Size of stochastic linear program datasets

Model Name	Num Stage 1 Vars.	Num Stage 2 Vars.	Num Stage 2 Constrs.
8t120s	336	2513280	1605360
8t1000s	1216	20944999	13378000
10t120s	390	3068760	1988640
10t1000s	1270	25573000	16572000

All experiments were performed on the 300 node (3600 cores) Taub cluster installed at University of Illinois. Each node has Intel HP X5650 2.66 GHz 6C processors and 24GB of memory. The cluster has a QDR Infiniband network communications with a Gigabit Ethernet control network. We used Gurobi [41] as the LP solver.

4.6 Results

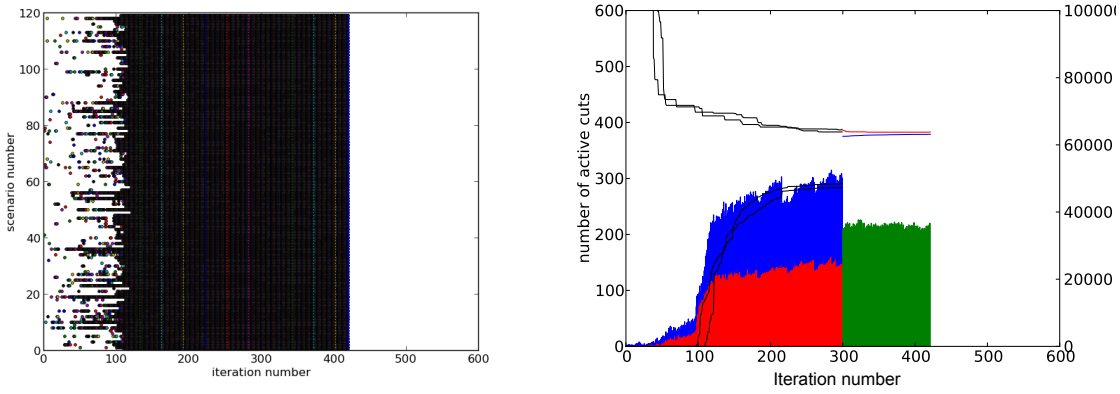
In Figure 4.3a, we show the scenarios that have active cuts in Stage 1 in each iteration of the Bender's multicut method applied to 8t120s dataset. The x-axis is the iteration number, and y-axis is the scenario number. In the vertical line corresponding to any iteration number, a dot in the horizontal line corresponding to a scenario number means that a cut obtained from the Stage 2 optimization of that scenario was active in that iteration. As can be seen in the figure (Figure 4.3a), very few scenarios have active cuts in the initial few rounds. As the optimization progresses, the number of scenarios with active cuts increases with the increase in the iteration number. And eventually, after approximately 220 iterations, all the scenarios have active cuts in Stage 1. The total number of active cuts in each iteration are shown in Figure 4.3b. The upper line shows the upper bound, and the lower line shows the lower bound as the number of iterations increase.



(a) Scenario Activity

(b) Cut Activity

Figure 4.3: Multicut Benders Method. Total Iterations = 495, Time to Solution = 1190s



(a) Scenario Activity

(b) Cut Activity

Figure 4.4: SAM with decomposition into 2 subproblems for 300 iterations. Total Iterations = 415, Time to Solution = 784s

For testing the proposed SAM algorithm, we divided the original problem with 120 scenarios into two subproblems each with 60 scenarios. The subproblems are solved for a maximum of 300 rounds, after which the cut constraints are collected from both of them and these cut constraints are used as the initial set of constraints for solving the original problem with 120 scenarios. Figure 4.4a shows the scenario activity for this method. As can be seen in the figure, the overall scenario activity is much higher in the initial iterations of the SAM approach than in the original Bender's method. Figure 4.4b shows the number of cuts that were active in each of the subproblems. The two bar shades correspond to the two subproblems. The bars are stacked on top of each other to show the total number of active cuts in both the subproblems. Bars after iteration 300 show the number of active cuts for the original problem (P), which begins optimization at iteration 301. As in Figure 4.3b, the lower and upper lines correspond to the lower and upper bounds, respectively - initially of the subproblems, and then of the original problem. Total time to optimization is 784 seconds with the SAM approach as compared to 1190 seconds with the original Bender's method.

We have extended our algorithm to split-and-hierarchical-merge (SAHM) algorithm, in which the merging of the subproblems into the original problem is done in stages instead of at once as in the SAM algorithm. Figure 4.5 shows a schematic diagram of the SAHM approach. In the SAHM method, as the hierarchical phases progress, number of subproblems

and hence the number of Stage 1 solvers are reduced by half. For instance, SAHM method that begins with 8 subproblems in Phase 0 has 4 subproblems in Phase 1, 2 subproblems in Phase 2 and finally 1 problem, that is the original problem. The user specifies the number of rounds for which the split phase should be run. These rounds/iterations are equally divided across the phases, that is, each phase of the SAHM method is run for the same number of rounds such that the total number of rounds sum up to the user specified rounds for the split phase. At the end of each phase, subproblem i of the current phase sends its cuts to the $\frac{i}{2}$ subproblem of the new phase. Figure 4.6 shows the cut activity for SAHM approach. The original problem is first divided into 6 subproblems each with 20 scenarios. In the next stage, sets of two subproblems combine to form one subproblem, giving a total of three subproblems. Finally, these three subproblems are combined into the original problem. Each of these stages is executed for 150 rounds, after which optimization of the original problem begins. Various colors in Figure **??** correspond to different subproblems. Total time to solution using SAHM was 507 seconds, giving us an improvement of 58% over the Bender's method.

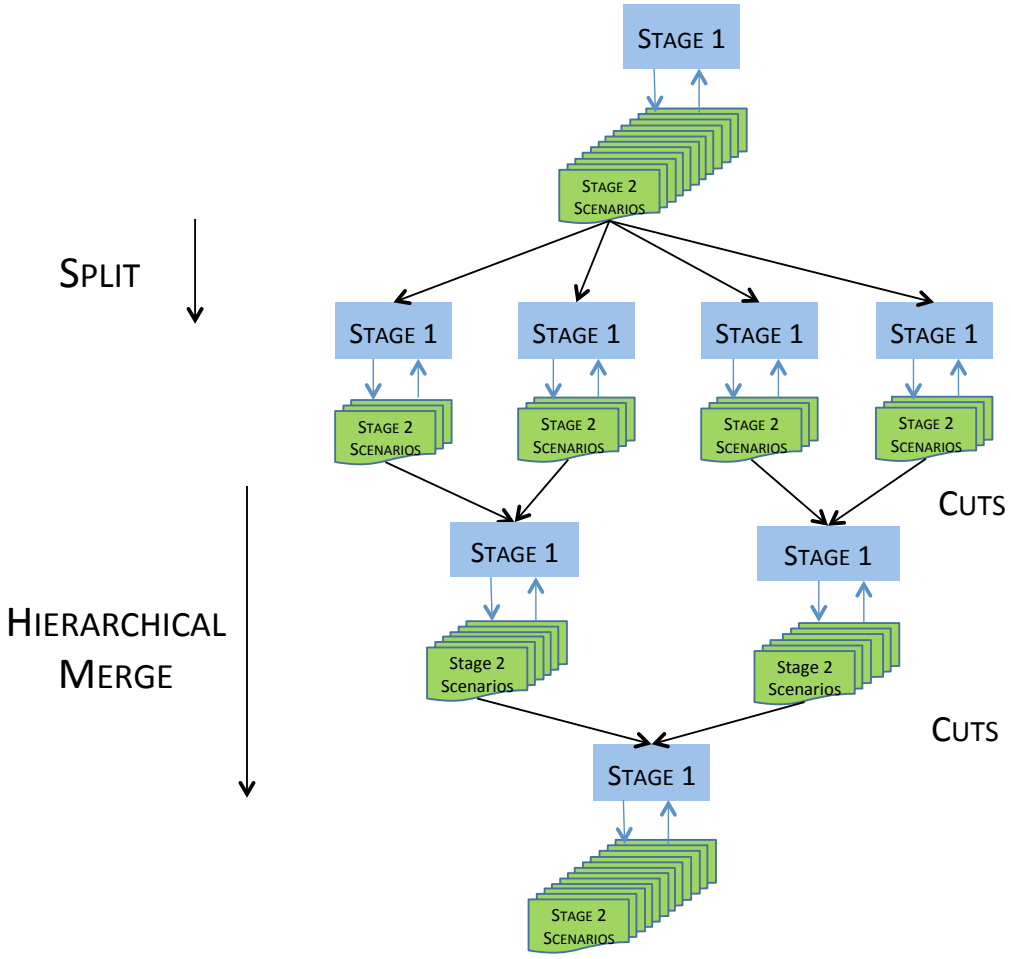


Figure 4.5: Schematic of the Split and Hierarchical Merge (SAHM) method

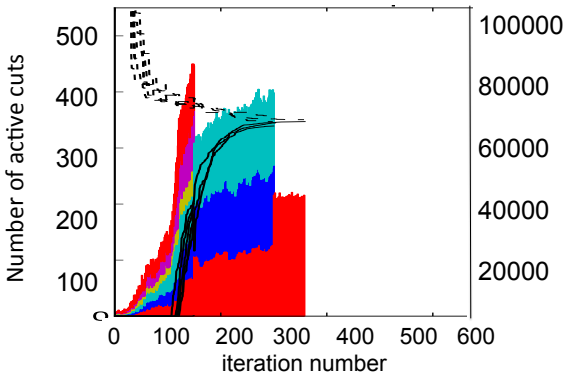
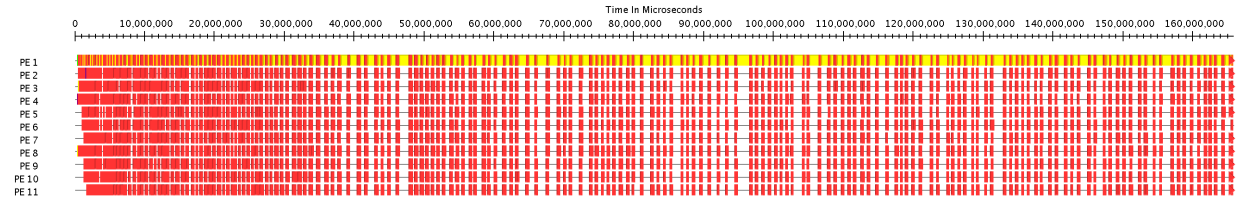


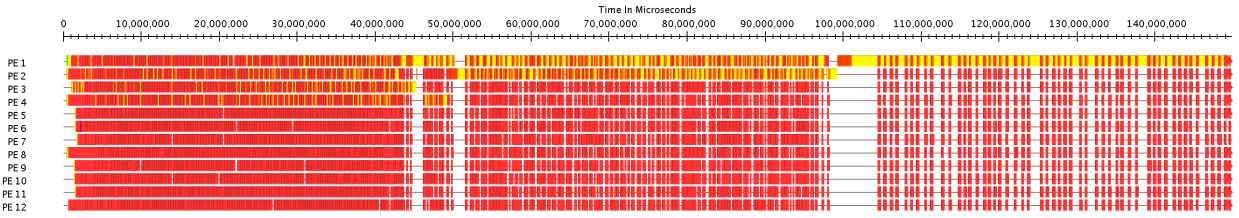
Figure 4.6: SAHM with decomposition into 6 subproblems for 150 iterations followed by 3 subproblems for 150 iterations. Total Iterations = 360, Time to Solution = 507s

Figure 4.7 presents the timeline view of processors during naïve Bender's method and

SAHM method execution. Processor activity is shown for 12 processors, where each processor is represented by a horizontal line and the bars on top of the line show the activity on that processors at that time. Red bars correspond to Stage 2 optimizations and the yellow bars correspond to Stage 1 optimization. White space means that the processor is sitting idle at that time. Therefore, larger white space means poor parallel efficiency. In Naïve Bender's method, there is only one Stage 1 solver, which is insufficient to keep keep the Stage 2 solvers busy (Figure 4.7a). On the other hand, in the SAHM method with 4 as the initial number of subproblems, there are 4 and 2 Stage 1 problems in Phase 0 and 1, respectively. These keep the processors busy as they can generate sufficient Stage 2 work because of the smaller Stage 1 bottleneck (Figure 4.7b). The three phases in the SAHM method can be seen in Figure 4.7b. They are distinguishable by the different processor utilizations during the three phases and also by the the number of Stage 1 solvers. SAHM method, therefore, also has higher parallel efficiency and therefore higher speedup as the number of processors are increased to solve the same problem.



(a) Naïve Bender's method



(b) SAHM method

Figure 4.7: Timeline view of parallel naïve Bender's method and SAHM method execution

Tables 4.3-4.5 shows the solution time for the various datasets using the SAHM method (with different number of rounds of the split-phase) and the naïve Bender's method. As we know from Chapter 2 and Chapter 3 that multiple runs of parallel benders method with

identical configuration can have different execution time, we report the average of at least 3 runs done for each configuration in these tables. SAHM outperforms the benders method in all the cases. For example, SAHM method leads to a reduction in solution time by as much as 74% as compared to the naïve Bender's method for 10t1000s dataset at 192 procesors (Table 4.5b). Additionally, it has significantly higher parallel speedups. For example, we get up to 7.8x speedup with SAHM for 8t1000s as the number of processors are scaled from 12 to 192, while the naïve Bender's method gives a speedup of only 4.2x (Table 4.4b). Our results also show that while the solution time decreases as we increase the number of initial subproblems from 4 to 8, we do not see any significant improvements as the number of initial subproblems are increased further to 16 (Table 4.4). We also observe that the benefits of SAHM are much higher when the number of scenarios are larger e.g. benefits of SAHM for 10t1000s (Table 4.4) are much higher than they are for 8t120s (Table 4.3).

Table 4.3: Solution time of 8t120s model with Naïve Bender's and SAHM method with different number of split-phase rounds

(a) SAHM method with 4 initial subproblems

#Processors	#Split-phase Rounds					SAHM	SAHM	Benders	Benders
	100	**200**	**300**	**400**	**500**	Best Time	Speedup	Time	Speedup
12	837	800	777	729	758	729	1.0	876	1.0
24	506	479	508	413	437	413	1.8	571	1.5
48	409	317	255	265	278	255	2.9	449	2.0
96	637	258	213	208	202	202	3.6	345	2.5

(b) SAHM method with 8 initial subproblems

#Processors	#Split-phase Rounds					SAHM	SAHM	Benders	Benders
	100	**200**	**300**	**400**	**500**	Best Time	Speedup	Time	Speedup
12	939	868	787	748	844	748	1.0	889	1.0
24	604	481	446	443	477	443	1.7	580	1.5
48	413	313	266	243	254	243	3.1	425	2.1
96	301	320	207	202	205	202	3.7	354	2.5

Table 4.4: Solution time of 8t1000s model with Naïve Bender's and SAHM method with different number of split-phase rounds

(a) SAHM method with 4 initial subproblems

#Processors	#Split-phase Rounds					SAHM	SAHM	Benders	Benders
	100	**200**	**300**	**400**	**500**	Best Time	Speedup	Time	Speedup
12	5639	4966	4138	4550	4469	4138	1.0	5970	1.0
24	3020	2557	2145	2241	2315	2145	1.9	3398	1.8
48	1975	1575	1267	1180	1173	1173	3.5	2507	2.4
96	1417	1033	858	916	771	771	5.4	1901	3.1
192	1142	852	699	652	643	643	6.4	1491	4.0

(b) SAHM method with 8 initial subproblems

#Processors	#Split-phase Rounds					SAHM	SAHM	Benders	Benders
	100	**200**	**300**	**400**	**500**	Best Time	Speedup	Time	Speedup
12	4954	4475	4098	4279	4216	4098	1.0	6062	1.0
24	2942	2005	2025	2054	2179	2005	2.0	3293	1.8
48	1820	1644	1151	1181	1296	1151	3.6	2510	2.4
96	1328	997	790	713	783	713	5.7	1863	3.3
192	1006	665	600	536	526	526	7.8	1445	4.2

(c) SAHM method with 16 initial subproblems

#Processors	#Split-phase Rounds				SAHM	SAHM	Benders	Benders
	200	**300**	**400**	**500**	Best Time	Speedup	Time	Speedup
48	1312	1465	1361	1288	1288	1.0	2507	1.0
96	989	820	812	837	812	1.6	1901	1.3

Table 4.5: Solution time of 10t1000s model with Naïve Bender's and SAHM method with different number of split-phase rounds

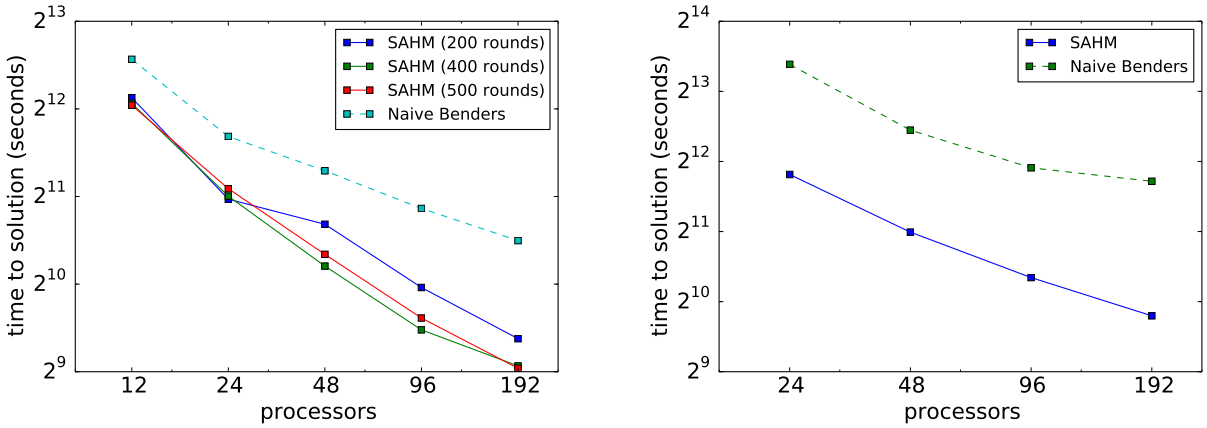
(a) SAHM method with 4 initial subproblems

#Processors	#Split-phase Rounds				SAHM	SAHM	Benders	Benders
	200	**300**	**400**	**500**	Best Time	Speedup	Time	Speedup
24	6477	4944	4291	3555	3555	1.0	10699	1.0
48	3958	2318	2402	2041	2041	1.7	5583	1.9
96	2440	2033	1632	1611	1611	2.2	3847	2.8
192	2188	1513	1321	1118	1118	3.2	3367	3.2

(b) SAHM method with 8 initial subproblems

#Processors	#Split-phase Rounds				SAHM	SAHM	Benders	Benders
	200	**300**	**400**	**500**	Best Time	Speedup	Time	Speedup
24	6753	3484	3502	-	3502	1.0	10699	1.0
48	3292	2032	1903	2036	1903	1.84	5583	1.9
96	2449	1490	1282	1299	1282	2.73	3847	2.8
192	1706	1266	982	890	890	3.93	3367	3.2

Figure 4.8 shows the scaling plot of two datasets with SAHM and naïve Bender's method.



(a) 8t1000s

(b) 10t1000s

Figure 4.8: Solution time with SAHM (with 8 initial subproblems) and Naïve Bender's method

4.7 Summary

We plan to evaluate and extend the proposed scenario decomposition schemes in the following ways:

- Currently, the number of rounds for which the subproblems are executed before they are merged is specified by the user/programmer. An important milestone is to dynamically determine during the execution of the program, the optimal time to merge the subproblems into the original problem. This could be based on the cut activity of the subproblems.

- Explore clustering schemes such that either similar or different scenarios are in the same subproblem during the split phase of the SAM algorithm. Study the affect of clustering on the solution time of the stochastic programs.

CHAPTER 5

Accelerating Two-stage Stochastic Linear Programs Using Lagrangean Decomposition

In this chapter, we propose a lagrangean decomposition based approach for accelerating the convergence of stochastic linear programs. With the proposed approach we are able to significantly reduce the solution time of many datasets and solve some of the previously intractable problems. We show scalability of the proposed approach to up to 192 processors.

5.1 Introduction

As in the previous chapter, our focus is on two-stage stochastic linear programs but the approach can be applied as it is to two-stage stochastic integer programs that have mixed-integer variables in Stage 1 but only linear variables in Stage 2. In two-stage stochastic linear programs, first stage optimizes over the Stage 1 decision variables (also called the strategic decisions) which are evaluated against several scenarios in Stage 2 (which makes the operational decisions). Stage 2 provides a feeback in the form of cuts to Stage 1, which then reoptimizes itself with the new set of constraints. The iterative process continues, until an optimal solution within the convergence threshold is found.

The design of a parallel implementation of stochastic linear program consists of a master that solves the Stage 1 problem and then the workers evaluate the Stage 2 scenariors for

the current Stage 1 solution. Because of the uncertain and variable solve times of linear programs, the work assignment to workers can be implemented as a pull-based scheme in which a worker requests work from the master whenever it runs out of work. This parallel design has been described in detail in Chapter 2.

In each iteration of the multicut L-shaped method [3], every scenario adds 1 cut to the Stage 1. Therefore, size of the Stage 1 linear program increases as the computation progresses, making it a serial bottleneck. In Chapter 2 we defined the concept of cut-window to alleviate the severity of this bottleneck. Cut-window is defined as the maximum number of cuts that are kept in the Stage 1 linear program at any time. Cut-window is maintained by discarding cuts that have low usage rate (details in [64]). While this approach makes certain problems tractable, larger problem still emain intractable. This is because of the large cut-window and the large number of iterations required by such problems to converge. A large cut-window increases the Stage 1 bottleneck (Fig 5.1) which decreases the parallel efficiency significantly. In this work, we propose an appraoch in which we decompose the Stage 1 and Stage 2 linear programs using lagrangean decomposition.

In Section 5.2, we give a preliminary to the lagrangean decomposition method. We then briefly describe the two-stage stochastic formulation of the military aircraft allocation in Section 5.3. In Section 5.4, the proposed decoposition scheme is described. Section 5.5 describes the application of the proposed approach to the military aircraft allocation problem. In Section 5.6 we present our parallel design of Lagrangean Decompose And Merge (LDAM) method and analyze its performance in Section 5.7. Finally, the conclusion and future work are given in Section 5.8.

5.2 Introduction to Lagrangean Decomposition

Simplex algorithm and Interior point methods, used for solving linear programs, are inherently difficult to parallelize. Hence alternative ways are sought for solving large scale linear programs. Lagrangean decomposition is an approach in which a large linear program

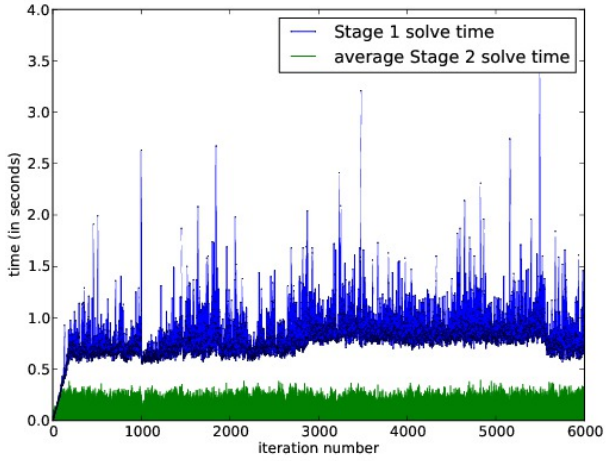


Figure 5.1: Stage 1 and Stage 2 solve times of a sample run of 15t problem

is decomposed into multiple independent smaller linear programs which can be solved efficiently. The results of these subproblems can then be combined to obtain the optimal value of the original linear program. For a simple working example of lagrangean decomposition, consider the linear program in Table 5.1 with 3 variables and two constraints. This linear program is to be optimized by decomposition into two subprograms with one constraint each. Lagrangean decomposition involves the following steps:

Step 1: Identify linking variables (only x_2 in this case) and introduce a cloned variable ($x_2^{'}$) for each such linking variable. Transform the linear program into an equivalent problem by using that copy of the variable that corresponds to the subproblem to which the constraint belongs. Add copy constraints of the form $x_2 - x_2^{'} = 0$ to equalize the values of the cloned variables. The resultant LP is called as the LDP.

Step 2: Dualize the copy constraints ($x_2 - x_2^{'} = 0$) with multiplier λ.

Step 3: Now the linear program can be decomposed into two independent smaller linear programs.

Methods such as *subgradient* search [70, 71] are used for searching optimal value of λ that minimizes the value of the dual term in the objective function. *Subgradient* search for opti-

	$\max\ -2x_1 - 3x_2 - 4x_3$	
	$x_1 + 2x_2 \leq 2$	
	$3x_2 + 5x_3 \leq 5$	
	$x_1, x_2, x_3 \in \{0,1\}$	
Step 1	$\max\ -2x_1 - 3x_2 - 4x_3$	
	$x_1 + 2x_2 \leq 2$	
	$3x_2^{'} + 5x_3 \leq 5$	
	$x_2 - x_2^{'} = 0$	
	$x_1, x_2, x_2^{'}, x_3 \in \{0,1\}$	
Step 2	$\max\ -2x_1 - 3x_2 - 4x_3 + \lambda(x_2^{'} - x_2)$	
	$x_1 + 2x_2 \leq 2$	
	$3x_2^{'} + 5x_3 \leq 5$	
	$x_1, x_2, x_2^{'}x_3 \in \{0,1\}$	
Step 3	$\max\ -2x_1 - (3+\lambda)x_2$	$\max\ \lambda x_2^{'} - 4x_3$
	$x_1 + 2x_2 \leq 2$	$3x_2^{'} + 5x_3 \leq$
	$x_1, x_2 \in \{0,1\}$	$x_2^{'}, x_3 \in \{0,1\}$

Table 5.1: Steps in lagrangean decomposition applied to a sample linear program

mal λ starts with an initial guess of λ and the objective value. Subproblems are evaluated in each iteration, and a new λ and objective value are proposed. The process continues until the obtained objective value converges with the proposed objective value. In this way, the optimal solution to the original linear program is obtained using lagrangean decomposition. Lagrangean decomposition has been used to solve a variety of problems in literature. Shah et al [72] proposed a new lagrangean decomposition algorithm to solve realistic large scale refinery scheduling problem in reasonable computation time. Hosni et al [73] propose a lagrangean decomposition approach to solve the problem of assigning passengers to taxis and computing the optimal routes of taxis, also known as the shared-taxi problem. They show lagrangean decomposition approach leads to tighter bounds in shorter computational time as compared to solving the full mixed integer program using CPLEX. Ghaddar et al [74] use lagrangean decomposition to solve the pump scheduling problem in water netowrks that accomodates the changing price of energy (dynamic pricing) for pumping water while ensuring continuous supply of water. Raidl [75] has done a literature review of how metaheuristics alongwith lagrangean decomposition has been used to solve some complex optimization

problems [76–79]. Mouret et al [80] propose a lagrangean decomposition approach solve a large-scale mixed-integer nonlinear program (MINLP) that integrates two main optimization problems in the oil refining industry: crude-oil operations scheduling and refinery planning. Rosa et al [81] study the application of lagrangean decomposition method on convergence of multistage stochastic programs. Ruszczyński et al [82, 83] review various decomposition methods proposed in literature for stochastic program optimization. Saeed et al [84] propose a modified version of the L-shaped method that uses augmented lagrangean to reduce the number of iterations and time of solving a two-stage stochastic linear program.

5.3 jetAlloc: The Military Aircraft Allocation Problem

The Air Mobility Command (AMC) of the U.S. Department of Defense has to deal with allocation of $1,300$ military aircraft over a period of 30 days. The allocation to different missions at different locations and days is to be optimized in the presence of uncertainty in the demands. We model this problem as a two-stage stochastic linear program. Stage 1 does the aircraft allocation to different missions during the month. These allocation decisions are evaluated in Stage 2 by scheduling the aircraft to meet the demand requirements. Details of the problem and its stochastic formulation are given in Chapter 6. The structure of the formulation is given in Figure 5.2. Stage 1 linear program allocates a fixed number of aircraft to different missions and operating wings every day for a fixed number of upcoming days. Stage 2 schedules these aircrafts to meet the mission demands. Unmet demands on a given day are carried over to the following day. New mission demands on day t added to the unmet demands from the previous days constitutes the total mission demand on a given day t. Unmet demands are penalized at a given rate (\$/day).

5.4 Lagrangean Decomposition for Two-Stage Stochastic Problems

Lets say that Stage 1 allocates aircrafts over a period of t_p days. We decomposed Stage 1 linear program into $n_{sp} = \frac{t_p}{t_{sp}}$ subproblems, where subproblem i $(i = 0, 1, .., n_{sp} - 1)$ corresponds to aircraft allocation from day $i * t_{sp}$ to $(i + 1) * t_{sp} - 1$. These subproblems

are independent of each other. Similar decomposition in Stage 2, however, does not lead to independent subproblems as there are linking variables e.g. the unmet demand variables that are present in more than one subproblem. See Chapter 6 for other linking variables in the Stage 2 LP of the stochastic formulation of military aircraft allocation problem. Figure 5.3a shows the lagrangean decomposition of Stage 1 and Stage 2 of the stochastic program. Since, Stage 1 subproblems are independent of each other, there are no lagrangean mulitpliers in Stage 1, while the Stage 2 optimization involves search for the optimal dual multipliers. Since search for optimal dual multipliers is required in every iteration of the L-shaped method, this can be computationally prohibitive as the linking variables can be large in number and the subgradient methods for optimal multiplier search are known to be slow in convergence.

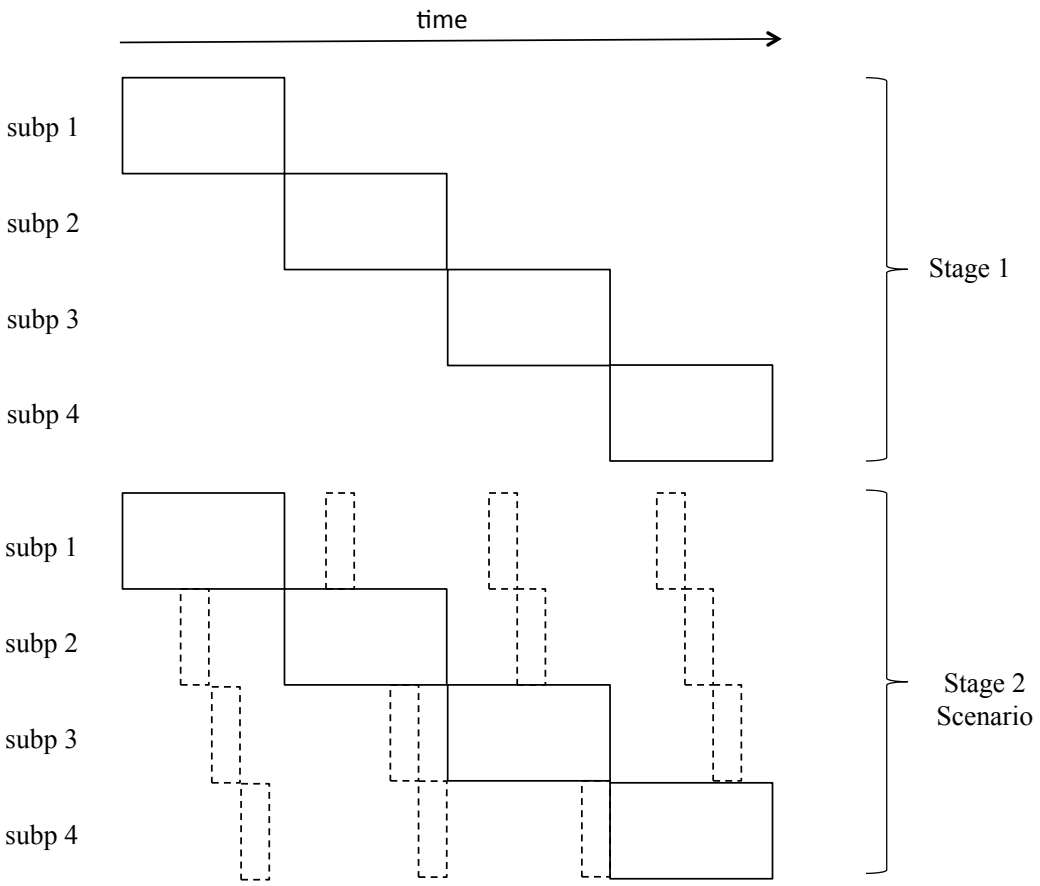


Figure 5.2: Structure of the two-stage stochastic program for military aircraft allocation problem. Vertical bars correspond to columns. Stage 1 LP can be divided by time index of variables into independent subproblems. Stage 2 LP has variables that are present across the subproblems and hence subproblems are not completely independent of each other.

Theorem 1 below shows that the optimal dual multipliers for the subproblems are equal to the dual values corresponding to the respective copy constraints in the LDP (See Section 5.2 for definition of copy constraints and LDP). We further show that the optimal solution of the subproblems is same as the solution of the lagrangean decomposed problem. Therefore the cuts from the subproblems can be computed without actually having to optimize the subproblems and do the expensive search for the optimal lagrangean multipliers. Since, Stage 2 solve times are significantly lesser than the Stage 1 solve times, solving the lagrangean decomposed problem does not hamper the parallel efficiency. Additionally, this avoids the computationally prohibitive lagrangean multiplier search. The schematic for this approach

is given in Figure 5.3b.

Decomposing the cut from the lagrangean decomposed problem into cuts of Stage 2 subproblems eliminates part of the Stage 1 search space when these cuts are added to the Stage 1 LP. This is easily understood in this way that the search space for constraints $\theta_1 \geq d_1y_1 + d_2y_2, \theta_2 \geq d_3y_3 + d_4y_4$ is a subset of the search space of the constraint $\theta_1 + \theta_2 \geq d_1y_1 + d_2y_2 + d_3y_3 + d_4y_4$. Therefore, to get to the optimal solution, the original stochastic program needs to be solved. The cuts obtained from the lagrangean decomposed stochastic program can be merged together and used as the initial set of cuts for optimization of the original stochastic program. We solve the lagrangean decomposed stochastic program for a fixed number of iterations and then solve the original stochastic program.

Theorem 1

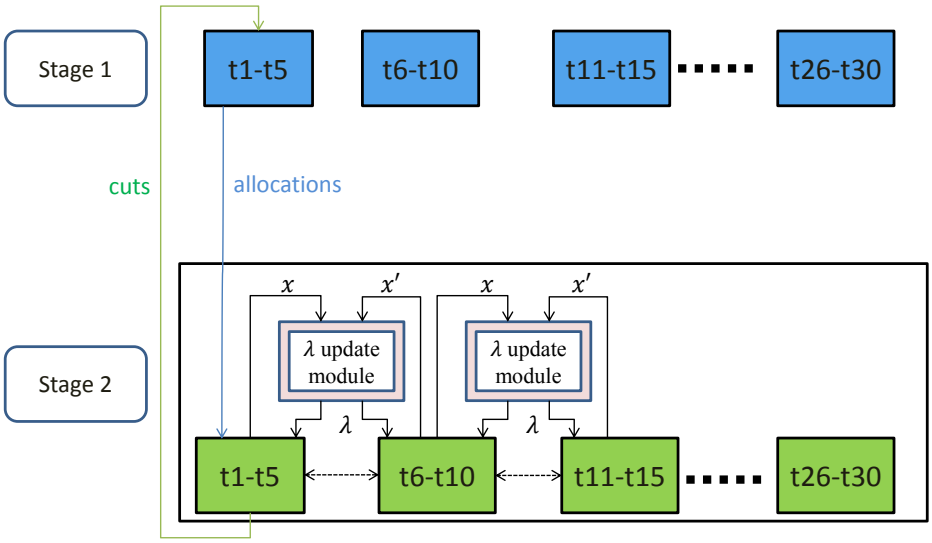
Dual values corresponding to copy constraints in LDP are equal to the optimal lagrangean multipliers of the decomposed subproblems.

Proof

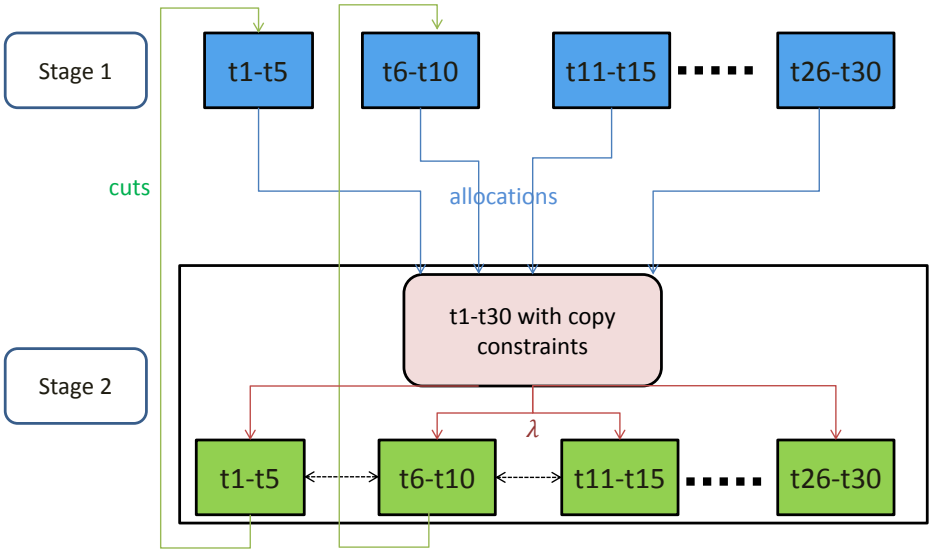
We call a constraint that has variables from different time-periods as the linking constraints. Let P be a linear program with n variables and m constraints of :the form:

$$\begin{aligned} (P): \quad & \min \quad cx \\ & s.t. \quad Ax = b \end{aligned}$$

where $A \in \mathbb{R}^{m,n}$ is the coefficient matrix, $x \in \mathbb{R}^{n,1}$ is the decision variable vector, $b \in \mathbb{R}^{m,1}$ is the vector of the constant terms in the constraints and $c \in \mathbb{R}^{1,n}$ is the cost coefficient vector. The problem can be decomposed into two subproblems $P1$ and $P2$ such that $m1$ constraints are in $P1$ and the remaining $m2(= m - m1)$ constraints are in $P2$. Let $x_1 \in \mathbb{R}^{n_1,1}$ be the set of variables that appear only in the constraints of $P1$ and $x_2 \in \mathbb{R}^{n_2,1}$ be the the set of variables that appear only in constraints of $P2$. Each of the remaining variables which appear in both the subproblems can be assigned to one of the two subproblems. Let $x_{12} \in \mathbb{R}^{n_{12},1}(x_{21} \in \mathbb{R}^{n_{21},1})$ be the set of variables that are assigned to $P1(P2)$ but also appear



(a) Schematic for lagrangean decomposition of the two-stage stochastic program that requires search for optimal lagrangean multipliers for Stage 2 optimization. Search for optimal Stage 2 lagrangean multipliers in every iteration of the Bender's method can be computationally very expensive



(b) Optimal lagrangean multipliers in Stage 2 can be obtained by solving the Stage 2 lagrangean decomposed problem. This eliminates the search for lagrangean multipliers and thus also reduces the computation required for calculating cuts of Stage 2 subproblems

Figure 5.3: Lagrangean decomposition of the 30 day two-stage stochastic program and the proposed method to obtain cuts of Stage 2 subproblems

in the constraints of $P2(P1)$. The coefficient matrix A can now be written as:

$$A = \begin{bmatrix} A_1 & A_{12}^{(1)} & 0 & A_{21}^{(1)} \\ \\ 0 & A_{12}^{(2)} & A_2 & A_{21}^{(2)} \end{bmatrix}$$

Here, $A_1(A_2)$ is the coefficient matrix of variables $x_1(x_2)$ in subproblem $P1(P2)$. $A_{12}^{(1)}(A_{12}^{(2)})$ is the coefficient matrix of variables x_{12} in $P1(P2)$.

In order to decompose P into two independent subproblems, $P1$ and $P2$, one copy(or cloned) variable corresponding to each of the variable in x_{12} and x_{21} is added to P. Let those variables be x_{12}^c and x_{21}^c, respectively. Subsequently, terms corresponding to x_{21} variables in $P1$ are replaced by their corresponding cloned variables in x_{21}^c. Copy constraints of the form $x_{12} - x_{12}^c = 0$ are added to ensure that the value of the cloned variables equals that of their corresponding original variables. Let this lagrange-decomposed problem be named LD:

$$\begin{aligned} (LD): \quad & \max \qquad cx \\ & s.t. \quad A_D x_D = b_D \end{aligned}$$

where,

$$A_D = \begin{bmatrix} A_1 & A_{12}^{(1)} & 0 & 0 & 0 & A_{21}^{(1)} \\ \\ 0 & 0 & A_2 & A_{21}^{(2)} & A_{12}^{(2)} & 0 \\ \\ 0 & I & 0 & 0 & -I & 0 \\ \\ 0 & 0 & 0 & I & 0 & -I \end{bmatrix}, b_D = \begin{bmatrix} b^T & 0 \end{bmatrix}^T, \text{and} \tag{5.1}$$

$$x_D = \begin{bmatrix} x_1^T & x_{12}^T & x_2^T & x_{21}^T & {x_{12}^c}^T & {x_{21}^c}^T \end{bmatrix}^T$$

The only linking constraints between $P1$ and $P2$ in LD are the copy constraints. These constraints can be relaxed and corresponding penalty terms are added to the objective function, giving the Lagrange-relaxation problem (LR):

$$LR: \min \qquad cx + \lambda_{12}(x_{12}^c - x_{12}) + \lambda_{21}(x_{21}^c - x_{21})$$

$$s.t. \quad \begin{bmatrix} A_1 & A_{12}^{(1)} & 0 & 0 & 0 & {A_{21}}^{(1)} \\ 0 & 0 & A_2 & A_{21}^{(2)} & A_{12}^{(2)} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_{12} \\ x_2 \\ x_{21} \\ x_{12}^c \\ x_{21}^c \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

where, λ_{12} and λ_{21} are the dual variables corresponding to the copy constraints in LD. Independent subproblems $P1$ and $P2$ can now be obtained from LR.

$$P1: \min \quad [c_1 \quad c_{12} - \lambda_{12} \quad \lambda_{21}] \begin{bmatrix} x_1 \\ x_{12} \\ x_{21}^c \end{bmatrix} s.t. \quad [A_1 \quad A_{12}^{(1)} \quad A_{21}^{(1)}] \begin{bmatrix} x_1 \\ x_{12} \\ x_{21}^c \end{bmatrix} = b_1$$

$$P2: \min \quad [c_2 \quad c_{21} - \lambda_{21} \quad \lambda_{12}] \begin{bmatrix} x_2 \\ x_{21} \\ x_{12}^c \end{bmatrix} s.t. \quad [A_2 \quad A_{21}^{(2)} \quad A_{12}^{(2)}] \begin{bmatrix} x_2 \\ x_{21} \\ x_{12}^c \end{bmatrix} = b_2$$

Theorem: Let x^*, λ^* be the optimal primal and dual solution to the linear program P. Then the optimal primal and dual solutions of the subproblems $P1$ and $P2$ in which $\lambda_{12} = \lambda_{12}^*$ and $\lambda_{21} = \lambda_{21}^*$ are ${x_1}^*$, λ_1^* and ${x_2}^*$, λ_2^*, respectively.

Proof: The dual problem LD_λ corresponding to LD (which is equivalent to P) is:

$$\begin{aligned} LD_\lambda : \max \quad & \lambda b_D \\ s.t. \quad & \lambda A_D \leq c \end{aligned} \tag{5.2}$$

where $\lambda = [\lambda_1 \quad \lambda_2 \quad \lambda_{12} \quad \lambda_{21}]$ is the dual variable vector. The dual problems corresponding to $P1$ and $P2$ with dual multipliers obtained from the optimal solution to LD_λ, can now be written as:

$$\begin{aligned} P1_\lambda : max \quad & \omega_1 b_1 \\ s.t. \quad & \omega_1[A_1 \quad A_{12}^{(1)} \quad A_{21}^{(1)}] = [c_1 \quad c_{12} - \lambda_{12}^* \quad \lambda_{21}^*] \end{aligned}$$

$$\begin{aligned} P2_\lambda : max \quad & \omega_2 b_2 \\ s.t. \quad & \omega_2[A_2 \quad A_{21}^{(2)} \quad A_{12}^{(2)}] = [c_2 \quad c_{21} - \lambda_{21}^* \quad \lambda_{12}^*] \end{aligned}$$

Since λ^* is the optimal solution to the linear program (5.2), using the definition of A_D from (5.2) we have:

$$\begin{aligned} \lambda_1^* A_1 &\leq c_1 \\ \lambda_1^* A_{12}^{(1)} + \lambda_{12}^* &\leq c_{12} \\ \lambda_2^* A_2 &\leq c_2 \\ \lambda_2^* A_{21}^{(2)} + \lambda_{21}^* &\leq c_{21} \\ \lambda_2^* A_{12}^{(2)} - \lambda_{12}^* &\leq 0 \\ \lambda_1^* A_{21}^{(1)} - \lambda_{21}^* &\leq 0 \end{aligned} \tag{5.3}$$

Feasible solution of $P1_\lambda$ and $P2_\lambda$ will satsify the constraints in Equations (5.4) and (5.5), respectively.

$$\omega_1 A_1 \leq c_1 \qquad\qquad \omega_2 A_2 \leq c_2$$
$$\omega_1 A_{12}^{(1)} + \lambda_{12}^* \leq c_{12} \qquad\qquad \omega_2 A_{21}^{(2)} + \lambda_{21}^* \leq c_{21}$$
$$\omega_1 A_{21}^{(1)} - \lambda_{21}^* \leq 0 \qquad (5.4) \qquad\qquad \omega_2 A_{12}^{(2)} - \lambda_{12}^* \leq 0 \qquad (5.5)$$

Equation 5.3 implies that λ_1^*, λ_2^* are also feasible solution to $P1_\lambda$ and $P2_\lambda$, respectively as they satisfy constraints in Equations (5.4) and (5.5), respectively. Lets say there exists better solutions ω_1^* and ω_2^* to $P1_\lambda$ and $P2_\lambda$, respectively such that $\omega_1^* b_1 > \lambda_1^* b_1$ and $\omega_2^* b_2 > \lambda_2^* b_2$. Then optimal value of LD_λ at $\lambda = [\omega_1^*\ \omega_2^*\ \lambda_{12}^*\ \lambda_{21}^*]$ is $\omega_1^* b_1 + \omega_2^* b_2 > \lambda_1^* b_1 + \lambda_2^* b_2$, which is contradictory to the fact that $\lambda = [\lambda_1^*\ \lambda_2^*\ \lambda_{12}^*\ \lambda_{21}^*]$ is the optimal solution to LD_λ. Hence, λ_1^* and λ_2^* are also the optimal solutions to $P1_\lambda$ and $P2_\lambda$, respectively. Q.E.D.

5.5 Lagrangean Decomposition for Military Aircraft Allocation Problem

In the military aircraft allocation problem, aircraft allocations are made to different missions on day-to-day basis. These allocation decisions are made in the Stage 1 of the two-stage stochastic formulation. Allocation decisions on a given day do not affect the decisions on any other day. Equations in Appendix A.4 are the constraints in the Stage 1 linear program. Since variables from two different time-periods are never present in the same constraint, the Stage 1 linear program can be trivially decomposed into subproblems without introducing any cloned variables. Each subproblem corresponds to the allocation problem for a given time range. The objective value of the original linear program is then simply the sum of the objective values of its subproblems.

Unlike Stage 1, in Stage 2 variables from different time periods are linked with each other via constraints. We tag each constraint with the time-period it corresponds to e.g. Consider a simplified constraint from the Stage 2 of our formulation (Equation 5.6). a constraint

that assigns load to aricrafts on day t is tagged with the t index.

$$-u_{t-1} + z_t + u_t = D_t \tag{5.6}$$

This constraint corresponds to load assignment at time t. u_{t-1} is the unment demand at time $t-1$, D_t is the new demand at time t, u_t is the unment demand at time t, and z_t is the met demand at time t. We assign this constraint to subproblem that corresponds to time t. The variables of the form u_t can belong to more than one subproblems and hence their clones are introduced.

Following steps are involved in extracting subproblem cuts from the solution of the original problem in Stage 2. Lets assume that the original problem schedules aircraft over a period of t_p days. This original problem is divided into n_{sp} subproblems and hence each subproblem schedules aircrafts for $t_{sp} = \frac{t_p}{n_{sp}}$ days. Subproblem i ($i \in 0, 1, ..., n_{sp} - 1$) schedules aircraft from day $t_{sp} * i$ to day $t_{sp} * (i + 1) - 1$. Based on the t index, each constraint and variable is assigned to one of the subproblems, also called the *native* subproblem of that constraint or variable, respectively. Lets say $subp(v), subp(c)$ gives us the *native* subproblem of the variable v, constraint c, respectively. Let $\mathcal{V}, \mathcal{C}$ be the set of variables and constraints in the original problem, respectively. Algorithm 3 gives the psuedocode for generating LDP.

Algorithm 3: Pseudocode for generating LDP

```
for all v ∈ V do
  if ∃c ∈ C s.t. subp(v) ≠ subp(c) then
     add new variable v^subp(c)
     add equality constraint v = v^subp(c)
  end if
end for
for all c ∈ C do
  for all v ∈ varlist(c) do
    if subp(c) ≠ subp(v) then
      v ← v^subp(c)
    end if
  end for
end for
```

In Lagrangean relaxation, the equality constraints for the cloned variables are removed

and added to the objective function. Algorithm 4 gives the pseudocode for generating the objective functions of the subproblems which are in turn used for generating the cuts for the subproblems.

Algorithm 4: Pseduocode for generating objective function of the subproblems

for all $i \in [0, n_{sp} - 1]$ **do**
  $objexpr(subp_i) = 0$
end for
for all $v \in objexpr(P)$ **do**
  $objexpr(subp(v)) += \text{coeff}(v, objexpr(P)) * v$
end for
for all constraints of the form $v - v^{subp_I} = 0$ and dual optimal value $= \lambda_v^{subp_I}$ **do**
  $objexpr(subp(v))$ $+= \lambda_v^{subp_I} * v$
  $objexpr(subp_I)$ -= $\lambda_v^{subp_I} * v$
end for

The objective value of each of the subproblem can be obtained by evaluating the corresponding objective expressions with the solution of the original problem i.e. if y^* is the solution of the original problem then $val(objexpr(subp_I), y^*)$ gives the objective value of the subproblem I. Dual values of the allocation variables are used to obtain the cuts for each subproblem.

5.6 Parallel Design of LDAM and Experimental Setup

We do a parallel implementation of the LDAM scheme using Charm++ [5,62] as the parallel programming language and the parallel runtime system. Stage 1 subproblems can be solved independent of each other and hence also in parallel. Stage 1 decomposition, therefore, gives us parallelism in Stage 1 which is not present in the naive Bender's design. Additionally, since the size of the subproblems is much smaller than the original Stage 1 problem size, size of the individual Stage 1 bottleneck is also reduced because of the simultaneous optimization of the Stage 1 subproblems on parallel processors. The processor distribution is as follows. Processor 0 is dedicated to Stage 1 and Stage 2 manager. Each of processor 1 to n_{sp} are assigned one Stage 1 subproblem each. Processor 1 to $p-1$ also act as Stage 2 solvers, where p is the total number of processors. The parallel design is depicted in Figure 5.4.

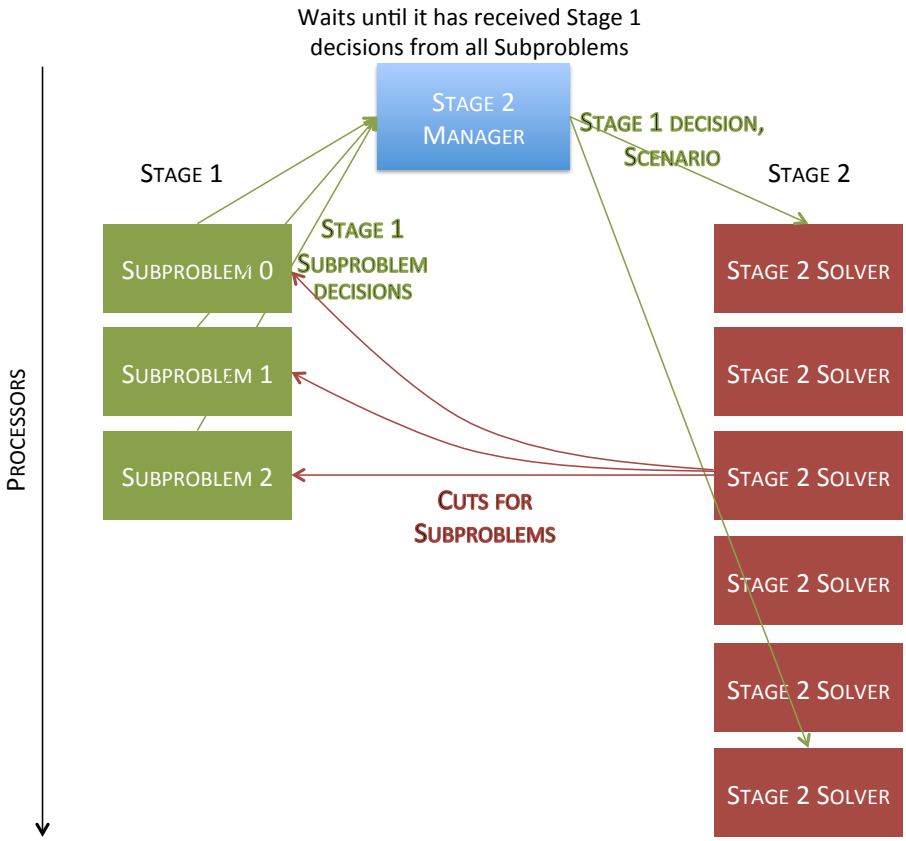


Figure 5.4: Parallel design of the bootstrap phase with lagrangean decomposition. In each iteration, Stage 2 Manager waits until it has received Stage 1 decision from each Stage 1 subproblem solver. It then combines the Stage 1 decision from the subproblems and sends it to Stage 2 solvers. Stage 2 solvers solve the full lagrangean decomposed problem and send the respective cuts to the Stage 1 subproblem solvers.

The stochastic program datasets that we use are from the military aircraft allocation problem (Chapter 6). Table 5.2 gives the number of variables, contraints, and non-zeroes in Stage 1 LP and Stage 2 scenario LP. Table 5.3 gives the total number of variables and constraints in the specific instance of these problems, that we use for demonstrating the benefits of LDAM method.

Fix the size of models in this table

Table 5.2: Size of Stage 1 and Stage 2 LP in the stochastic linear program datasets

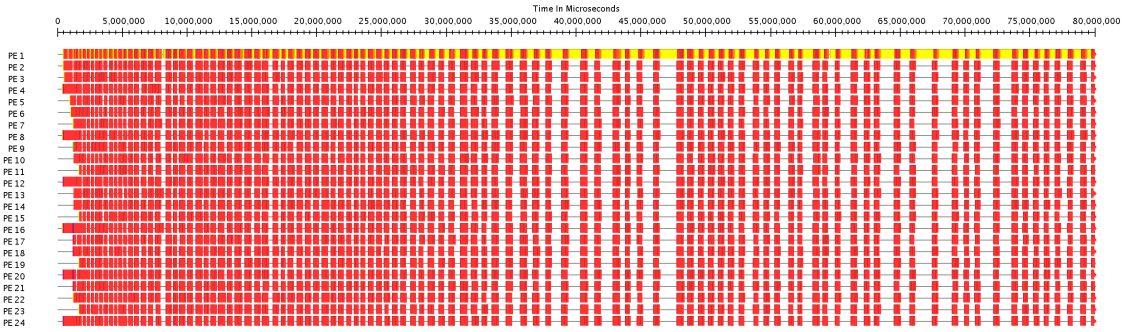
Test	1st Stage		2nd-Stage Scenario		Nonzero Elements		
Problem	Vars.	Constrs.	Vars.	Constrs.	A	W_i	T_i
8t	216	144	20944	13378	456	66881	252
10t	270	180	25573	16572	570	82797	308
12t	270	180	25573	16572	570	82797	308
15t	270	180	25573	16572	570	82797	308

Table 5.3: Size of the stochastic linear program datasets

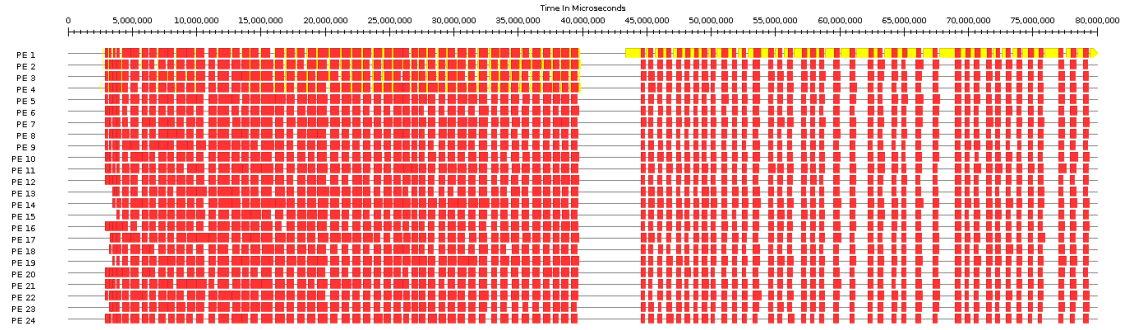
Model Name	Num Stage 1 Vars.	Num Stage 2 Vars.	Num Stage 2 Constrs.
8t120s	336	2513280	1605360
10t120s	390	3068760	1988640
12t120s	1270	25573000	16572000
15t120s	1270	25573000	16572000

5.7 Results

Figure 5.5 shows the timeline view of processors during the execution of naïve Bender's method (Figure 5.5a) and LDAM method (Figure 5.5b). During the decomposition phase, LDAM method has a higher processor utilization (average of 60%) as compared to the naïve Bender's method (averagoe of 54%). This is because of the smaller size of the Stage 1 bottleneck during the decomposition phase of the LDAM method.



(a) Naïve Bender's



(b) LDAM

Figure 5.5: Timeline view of processors for the naïve Bender's and LDAM method

Table 5.4-5.7 gives the performance of the LDAM method and NB method for several datasets. Each of the reported numbers are average of at least 3 runs for every configuration. While the benefits are not significant for smaller datasets, LDAM method significantly reduces the solution times for larger datasets (for example, 12t120s in Table 5.6) and makes otherwise intractable problems tractable (for example, 15t120s in Table 5.7).

LDP has higher number of variables and constraints than the original Stage 2 LP used in the NB method (Table 5.8). This leads to higher Stage 2 optimization time in the LDAM method as compared to the NB method. Therefore, parallel scalability of the LDAM method is smaller as compared to NB method.

Table 5.4: Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 8t120s dataset

(a) With 2 subproblems

#Processors	#Bootstrap Rounds					LDAM Best Time	Speedup	Naive Benders Time	Speedup
	10	20	30	40	50				
6	1338	1444	1702	1810	1434	1338	1.0	1696	1.0
12	851	1024	969	906	862	851	1.57	943	1.8
24	571	646	606	616	577	571	2.34	587	2.89
48	408	464	469	449	432	408	3.27	454	3.73
96	395	400	377	336	307	307	4.36	386	4.39

(b) With 4 subproblems

#Processors	#Bootstrap Rounds					LDAM Best Time	Speedup	Naive Benders Time	Speedup
	10	20	30	40	50				
6	1664	1952	1656	1543	1590	1543	1.0	1696	1.0
12	940	840	1015	906	831	831	1.86	943	1.8
24	573	600	589	629	597	573	2.69	587	2.89
48	410	493	447	425	400	400	3.85	454	3.73
96	381	372	381	359	366	359	4.29	386	4.39

Table 5.5: Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 10t120s dataset

(a) With 2 subproblems

#Processors	#Bootstrap Rounds				LDAM Best Time	Speedup	Naive Benders Time	Speedup
	75	125	175	225				
6	-	3609	3133	-	3133	1.0	5722	1.0
12	2414	5163	1563	2070	1563	2.0	2276	2.51
24	1269	1282	986	0	986	3.17	1233	4.64
48	708	584	661	680	584	5.36	704	8.12
96	452	410	413	524	410	7.64	420	13.6

(b) With 5 subproblems

#Processors	#Bootstrap Rounds				LDAM Best Time	Speedup	Naive Benders Time	Speedup
	20	40	60	80				
12	2167	2457	1601	1371	1371	1.0	2276	1
24	1516	814	698	934	698	1.96	1233	1.85
48	695	488	525	506	488	2.8	704	3.23
96	449	348	386	392	348	3.93	420	5.4

Table 5.6: Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 12t120s dataset

(a) With 2 subproblems

#Processors	#Bootstrap Rounds					LDAM Best Time	Speedup	Naive Benders Time	Speedup
	50	100	200	300	400				
24	-	-	-	0	2983	2983	1.0	13435	1.0
48	-	-	-	2000	2476	2000	1.49	10962	1.23
96	-	3074	1550	1407	1502	1407	2.12	-	-

(b) With 4 subproblems

#Processors	#Bootstrap Rounds					LDAM Best Time	Speedup	Naive Benders Time	Speedup
	50	100	200	300	400				
24	-	2237	2656	2977	3149	2237	1.0	13435	1.0
48	-	1694	1708	1865	2071	1694	1.32	10962	1.23
96	-	1161	1315	1420	1541	1161	1.92	-	-

Table 5.7: Solution time of LAHM (with different decompositions) and comparison with naïve Bender's time for 15t120s dataset

(a) With 3 subproblems

#Processors	#Bootstrap Rounds				LDAM Best Time	Speedup	Naive Benders Time	Speedup
	100	200	300	400				
48	-	-	-	6515	6615	1.0	-	-
96	-	-	5157	4920	4920	1.34	-	-

(b) With 5 subproblems

#Processors	#Bootstrap Rounds				LDAM Best Time	Speedup	Naive Benders Time	Speedup
	100	200	300	400				
48	9421	8522	9448	10072	8522	1.0	-	-
96	5852	6599	7264	7146	5852	1.46	-	-

Table 5.8: Stage 2 variables, constraints count and optimization times for various LDPs and the original Stage 2 LP

Parameter	Dataset					
	10t120s			12t120s		
	1 Subp	2 Subp	5 Subp	1 Subp	2 Subp	5Subp
Num Vars (per scenario)	25573	27497	30460	29262	31537	34161
Num Constrs (per scenario)	16572	18496	21459	19369	21644	24268
Optimization Time	0.12s	0.16s	0.18s	0.13s	0.18s	0.24s

5.8 Summary

Decomposing the original problem into smaller problem using lagrangean decomposition significantly improves the convergence rate of stochastic linear programs. We obtain up to 75% reduction in solution time for some problems while being able to solve some otherwise intractable problems. There is a significant amount of future work that ensures from here. The Lagrangean decomposition approach presented in this approach is used for solving problems with larger complexity, while the SAM approach proposed in Chapter 4 is used to solve stochastic programs with large number of scenarios. We intend to combine these two approaches to solve problems with higher complexity that also have large number of scenarios.

CHAPTER 6

Stochastic Optimization for Military Aircraft Allocation with Consideration for Disaster Management

The allocation of military aircraft to different operating wings and military functions must often be done without complete knowledge of demand. Such planning must provide for aircraft capacity in the event of imminent disaster threats. The problem is particularly difficult for militaries with large airlift capabilities such as the US military. In this work, we propose a two-stage mixed-integer stochastic optimization approach with complete recourse that can model the special requirements imposed by disaster relief. We demonstrate, using an imminent disaster scenario, a 35% benefit in cost as compared to deterministic optimization that does allocation assuming a nominal scenario in the future. The number of scenarios in the problem can also be very large because of the presence of large number of uncertainties. This makes the problem intractable for state-of-the-art commercial integer program solvers, such as, Gurobi. We show up to 57x speedup in time to solution while scaling from 4 to 242 processors, using our distributed solver for two-stage stochastic program, and compare its performance with the Gurobi mixed integer program solver.

6.1 Introduction

Most nations with large militaries have the ability to rapidly deploy forces using military and civilian aircraft. Such capabilities allow these nations to carry out military, humanitarian, and diplomatic activity across large geographic regions. These missions ship cargo, move military personnel and distinguished visitors, and serve during contingencies such as wars and natural disasters. Most of these activities are inherently unpredictable. In particular, natural disasters seldom provide any advance warning. Even in cases of hurricanes, the scale and extent of the disaster is difficult to predict. The challenge to military planners is that aircraft used to provide assistance during such disasters much be cannibalized from other planned operations. For large military nations such as the US, it is common to allocate aircraft to various geographic regions or operation wings months in advance. This poses a difficult challenge as the allocation of aircraft must be done under enormous uncertainty.

The US has an air mobility command which plans the allocation of aircraft for different types of missions and to different operating wings [85]. The air mobility command must provide periodic working operations plans (WOPs) in the face of uncertain demands, uncertain aircraft availability, etc. These plans must be robust; they must be sufficiently flexible to address myriad random changes while being cost- and mission-effective. They must also be sufficiently detailed to account for the unique details of military logistics. The WOP cycle is nominally one month. WOPs specify what mission types will be flown by which aircraft types using which operating wings. These plans primarily allocate aircraft to missions; details such as aircraft assignment, crew pairing, and mission routing are addressed subsequently. The WOP must incorporate sufficient buffer between aircraft allocated and available to address changes not only in cargo requirements, weather delays and aircraft breakdowns but also special requests for aircraft to assist in disaster recovery. When allocations are insufficient to meet these anomalies, the air mobility command must choose between delaying deliveries or potentially chartering civilian aircraft at a significantly higher cost to fill the void.

This chapter describes a stochastic mixed integer program that allocates aircraft to three of the primary mission types 1. Channel missions, which are regularly scheduled missions, 2. Contingency missions, which are irregularly scheduled missions that deliver cargo to an

international "hot spot," and 3. Special assignment airlift missions, in which military aircraft are chartered by organizations for a specific purpose such as disaster recovery. We model the allocation problem as a two-stage stochastic mixed integer program with complete recourse. Aircraft are allocated in the first stage while in the second stage subproblems conduct more detailed planning with probability-weighted mission and cargo demands over tens to hundreds of scenarios. The recourse is assured by allowing missions to be not flown, and penalizing the unflown missions. We use Bender's decomposition to iteratively solve the problem by passing allocations from the first stage to the second stage and dual information back. The iterative process is continued until a threshold convergence tolerance is met. Because of the scale of the problem caused by both the number of aircraft involved and the number of potential scenarios we use parallel processing to speed up the solution of the problem.

In this paper, we present the design of a distributed solver and demonstrate the feasibility of using large scale parallel computing for such problems and analyze the benefits of using stochastic optimization for such stochastic aircraft allocation problems. In particular, we show that in cases of imminent disaster, the stochastic approach leads to much lower expectation and variability of the cost of operation.

The chapter is organized into 5 sections. A survey of related work is given in Section 6.2. We present our stochastic formulation of the Air Mobility Command (AMC)'s aircraft allocation problem in Section **??**. Results are presented and analyzed in Section 6.4. Finally, the conclusion and future work are discussed in Section 6.5.

6.2 Literature Review

There has been significant work in optimization methods for providing relief during disaster. Natarajarathinam et al [86], Altay et al [87] and Paulsson [88] review how disasters and disruptions in supply chains have been handled in academic literature. Natarajarathinam et al classify the literature based on the scientific research method employed to address the crisis. 38% of the literature uses analytical approach, that is, simulation and mathematical modeling, while other approaches - conceptual, applied and empirical are used in

only 31%, 22%, 9% of the literature. In particular, Oh et al [89] have developed multi-commodity, multi-modal network models for predicting detailed routing and scheduling of commodities during a disaster using given modes of transportation. Stochastic optimization has been used previously for planning in disaster. Barbarosoğlu et al [90] propose a two-stage stochastic programming model to plan the transportation of first-aid commodities to disaster-affected areas. They develop a multi-commodity, multi-modal network formulation with resource supply, resource requirements, and transportation system capacities treated as random variables. Goh et al [91] present a stochastic model of the multi-stage global supply chain network problem, with supply, demand, exchange, and disruption as the stochastic parameters. Rappoport et al [92] propose a airlift-planning heuristic (APH) for planning the allocation of airlift resources for moving cargo and passengers during peacetime and crisis situations. Being a heuristic, APH can only give approximate solutions and provides a user-interface for the user to evaluate multiple options. In our work, we propose a stochastic formulation that aims at finding globally optimal solution, as even minor improvement in solution can lead to significant savings in cost. Stochastic models were also developed by Beamon et al [93] for inventory control during long-term emergency relief response. Barbarosoğlu et al [94] develop a hierarchical multi-criteria mathematical model for helicopter logistics planning in disaster relief operations.

Optimization methods have been applied successfully to several different problems in the commercial airline industry, for example, Teodorovic [95], Yu et al [96], Barnhart et al [97]. However, nature of the problems faced by the military airlift planners differ significantly from the commercial airline decision makers. For example, airlines face demands that have considerably less variability as compared to military airlift requirements that are driven largely by infrequent visits that can be of huge magnitude. Additionally, commercial airlines can chose the market that they want to serve and also with what frequency while the military airlift does not have this flexibility, although in many cases, they have much greater control over the transportation network infrastructure. Currently, CAMPS [98], Consolidated Air Mobility Planning System, is used to manage US military airlift. CAMPS consists of a Windows program, a set of web and application servers, and a database management system. It is essentially a rule based system that does automated scheduling of missions based on

a host of rules/criteria that the planner can model with CAMPS. However, more analytical methods are required for military operations and logistics. Baker et al [99] describe a large-scale linear programming model for routing cargo and passengers through a specified transportation network of U.S. Air force, subject to many physical and policy constraints. Salmeron et al [100] employ stochastic programming for planning the wartime, sealift deployment of military cargo that is subject to attack. Their approach shows that the stochastic solution incurs only a minor penalty when no attacks occur, but outcomes are much better in worst-case scenarios. To handle the uncertainty of aircraft reliability, Goggins et al [101] add a stochastic extension to the optimization model for determining the maximum on-time throughput of cargo and passengers that can be transported with a given fleet over a given network. On the other hand, our work addresses the problem aircraft assignment in the presence of stochasticity in military mission requirements. A wide range of applications of stochastic programming can be found in [27].

6.3 Stochastic Model for Aircraft Allocation Problem

The intended output of this model is the vector of aircraft allocations produced in Stage 1 of the stochastic mixed integer program. Realizations of future outcomes in Stage 2 influence this output. The second stage realizations consist of aircraft capacity constraints, aircraft flow constraints, cargo demand satisfaction constraints, and limits on usage of allocated aircraft. Currently, only cargo and aircraft demand fluctuations are modeled as random variables; incorporating random mission durations is ongoing. The objective function seeks to minimize the probability-weighted costs of aircraft operation and leasing, plus the cost of late and undelivered cargo.

While the categorization of missions may vary for different countries, in the context of the US Air Mobility Command, the missions can be categorized as follows. Any of these may require aerial refueling and transshipment of cargo, which are also incorporated.

- **Channel** These missions originate and terminate at an aircraft's home base, making several enroute stops to pick up and drop off cargo and passengers at major aerial

ports. The routes are regularly scheduled based on forecast cargo demand patterns. A realization consists of random cargo and passenger draws for each day along each route (outbound and inbound). The routes are fixed, but the frequency and aircraft used may be varied for the purposes of this model. Non-delivery penalties occur if channel cargo is undelivered for more than seven days.

- **Contingency** These are similar to channel missions in that they require cargo and passengers to be carried between specified locations, generally from the US to an overseas region. However, they differ from channel missions due to high demand variance and localized destinations. A realization consists of a Bernoulli draw for each type of contingency, each of which requires between a few and (potentially) hundreds of sorties. Late and non-delivery penalties are similar to those used for channel missions.

- **Special Assignment Airlift Missions (SAAM)** These aircraft are chartered for a specific time frame by a military organization for its exclusive use. A realization consists of daily aircraft required, aircraft type, mission routing, and mission duration. Demand is aircraft, not cargo centric, and is of moderate variance. There are opportunities for special airlift assignment missions to carry channel cargo while positioning to or from the customer's specified location. The unmet missions are penalized above the short-term rental rate of the associated aircraft.

Assignment of airlift capacity for disaster relief can be modeled as a mixture of contingency and special assignment airlift missions. While some relief supplies can be viewed as contingency cargo, other requirements may require charting entire planes to move cargo to the affected regions. We now discuss the Stage 1 and Stage 2 formulations (See Appendix A for the linear programs).

6.3.1 Stage 1 Formulation

The first stage determines the values of $y_{j,l,m}$, the allocation of aircraft by type (j), location (l), and mission (m). The optimal Stage 1 values (y^*) are sent to Stage 2 for scenario evaluations. Stage 2 sends back the optimal cost $Opt(\mathbf{y}*, \mathbf{s})$, and the dual value information

for every scenario (s), which together forms Stage 2 cuts. Stage 2 cuts are added successively until a convergence tolerance is met.

The objective function seeks to minimize the cost of civilian aircraft allocation, plus the probability-weighted sum of Stage 2 cuts. Military aircraft are excluded from the objective function because they do not incur allocation costs. The feasible allocation constraints limit the total allocated aircraft to the number available at each base. The Stage 2 cuts represent the dual costs from the mission and flight time constraints as affected by the Stage 1 y variables.

6.3.2 Stage 2 Formulation

The second stage models the execution of channel, contingency, and special airlift assignment missions for a large number of stochastic realizations. The constraints are given in Appendix A.5, and are referred along with the descriptions in this section. The values of $y_{j,l,m,t}$ generated from Stage 1 are used as inputs to this program. The Stage 2 objective function minimizes the sum of short-term rental costs incurred to meet SAAM and other disaster related requirements, the cost of late channel and contingency cargo, the cost of very late or undelivered cargo, and the cost of aircraft operations.

The demand constraints are represented as cargo inventories for each requirement across time periods; the difference between previous and current inventories, adjusted for deliveries, equals demand. There are separate constraints for cargo that must be directly delivered (Constraint 1), and cargo that may be either directly delivered or transshipped (Constraint 2). The transshipment constraints (Constraint 3) ensure cargo delivered in the first leg of a transshipment equals cargo delivered in the second leg. The aggregate capacity constraints (Constraints 4a, 4b) limit cargo deliveries by the cumulative capacity of the aircraft assigned for delivery. The specific capacity constraints (Constraint 5) are similar, but constrain the individual cargo types separately to account for their unique loading characteristics. The price break constraint (Constraint 6) enforces limits on late cargo.

Mission time constraints (Constraints 7, 8) ensure that no more aircraft are away from home base than have been allocated. Similarly, the flying time constraint (Constraint 9)

limits aircraft flight hours throughout the model time horizon to their historical maximum. Finally the air refueling constraint (Constraint 10) ensures that tankers are flown in support of aircraft sent along routes that transit air refueling locations.

Since, Stage 2 program does the aircraft scheduling for an entire month, it is very large in size and has many integral variables. The total number of Stage 2 scenario evaluations required for stochastic optimization can be very large in number and hence make the approach computationally intractable. The intended output of the problem is the Stage 1 decisions. Stage 2 optimizations are done only to obtain the cost of the Stage 1 decision and the optimal values of the Stage 2 optimizations are not used. Therefore, in order to make the problem computationally tractable, we relax the Stage 2 program to a linear program. The cost of the relaxation is then used as a proxy for the cost of the original Stage 2 integer program.

6.4 Results

The data used to implement this model have a variety of sources. Aircraft characteristics, costing, basing, and routing are based on historical patterns and publically available information. Channel and Special airlift assignment mission demands are historically based; contingency demands and locations are derived from a commonly used analytical data set. Tradeoff between leasing additional aircraft and delaying cargo is subject to a variety of conditions, but we generalize as follows: the maximum penalty for a planeload of non-delivered cargo is 10 percent higher than the cost of the most appropriate short-notice leased aircraft, multiplied by the duration of a typical mission length for that aircraft.

In this section, we discuss the cost benefits of stochastic optimization, followed by parallel speedup of our distributed solver and its comparison with a commercial state-of-the-art mixed-integer program solver.

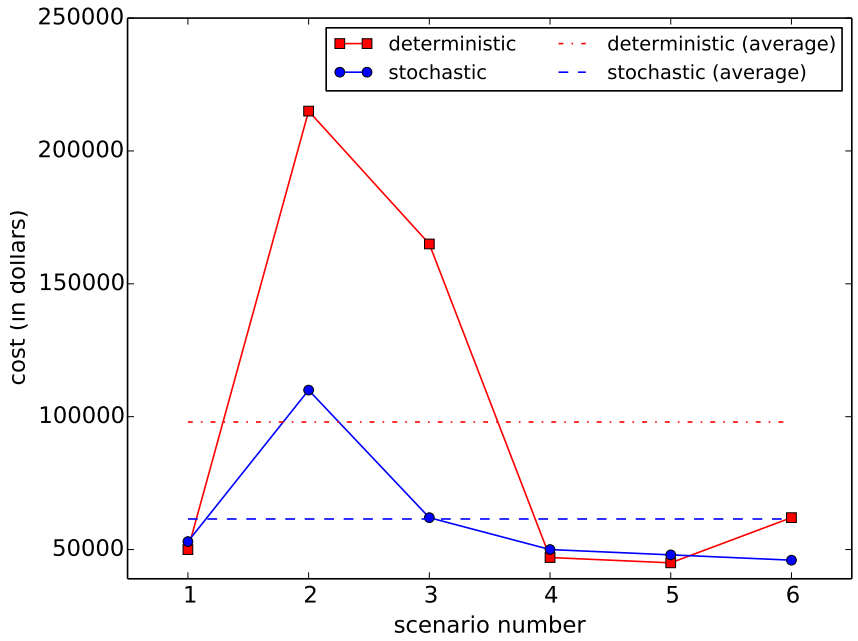


Figure 6.1: Hedging against future uncertainty can play a significant cost-saving role. In this 6-scenario example with an imminent disaster, squares correspond to each scenario's cost using deterministic optimization of expected demands; the horizontal average is depicted as a dashed line. In contrast, the circles correspond to stochastic optimization costs for each scenario (with the dashed-line average). Stochastic optimization yields only slightly more costly solutions when actual demands are low (scenarios 1, 4, and 5), but are much less costly when demands are elevated due to the disaster (scenarios 2, 3, and 6). The average cost savings in this example is approximately 35 percent.

6.4.1 Benefits of Stochastic Optimization

We do comparison of stochastic optimization as compared to deterministic optimization by dividing our analysis in two cases. In Case 1, we look at a scenario provided to us and add the needs from an imminent disaster relief operation in some scenarios. Case 2 is a larger scale problem in which there is no imminent disaster threat but some capacity is to be allocated as a routine requirement for disaster relief.

6.4.2 Case 1

We consider the case of an imminent hurricane which may or may not cause disaster depending upon whether it hits the shores. If it hits the shore, it will cause disaster leading

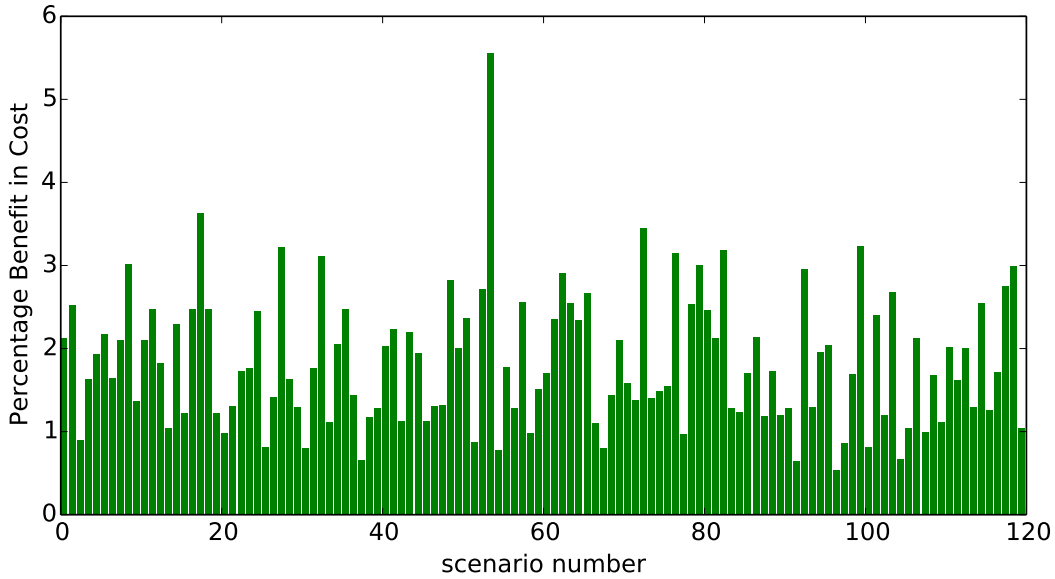


Figure 6.2: For randomly generated scenarios with no imminent disaster, hedging against future uncertainty usually leads to cost saving. In this 120-scenario example, the stochastic programming solution has a lower cost in all scenarios. The average cost savings in this example are 3 percent

to dedicated aircraft requirement for disaster relief depending upon the extent of disaster caused by it. This situation is captured by scenario 2, 3 and 6. On the other hand, if the hurricane does not hit the shores, there is small regular perturbations in channel and contingency missions. This is captured by scenario 1, 4 and 5. Figure 6.1 illustrates this 6-scenario example. It shows that deterministic optimization of average demands yields low cost solutions when actual cargo and aircraft demands are small or average, but perform very badly when actual demands due to the disaster are elevated. In contrast, stochastic modeling incorporates hedging against uncertainty and yields much improved solutions when actual demands due to disaster are elevated. The cost of hedging is approximately 5 percent, but reduces overall costs by as much as 57 percent and an average of 35 percent. Additionally, the cost variance is reduced by 66 percent. This finding supports the planner's goal of finding not a point solution at an unstable minimum, but a stable "trough" on the solution surface that balances cost savings with re-planning needs, while minimizing disruption to the existing plan. When implemented, this methodology could realize a significant reduction in cost, and/ or a significant increase in timely mission accomplishment.

6.4.3 Case 2

While Case 1 demonstrated the potentially large benefits that are obtainable from using stochastic programming when there is a specific demand scenario, Case 2 is more generic where there is no imminent knowledge of a disaster and hence no specific aircraft requirement. Here we randomly generate 120 scenarios using poisson distribution. These scenarios are used to obtain the stochastic programming solution. The figure shows the percent improvement of the stochastic solution over the corresponding deterministic solution obtained using average values for demand. Notice that in all cases the stochastic solution does better than the deterministic solution, albeit the differences are not as significant as in Figure 6.1. Because the scenarios are randomly generated, the average scenario is not one of the randomly generated scenarios. Moreover, in random scenarios, the chance that all demands will be higher or lower than their expected demand is very small. Accordingly, the benefits obtained from stochastic programming are somewhat smaller - averaging approximately 3 percent and with a maximum of 6 percent. The stochastic program performs better in all cases because it provides higher allocations to hedge against the chance that there will be high demands, especially where the recourse involves mission cancellation. Thus, it incurs higher costs of short term leases for demands that can be satisfied by commercial aircraft, but ensures that missions that require military only aircraft are suitably covered.

6.4.4 Parallel Performance

Our experiments were performed on a cluster of 300 compute nodes, each with two 2.67 GHz Intel® Xeon hex-core processors and 24 GB of RAM. The interconnection network is a high speed Voltaire QDR Infiniband switch.

The Stage 1 mixed-integer program has 57 integer variables and 18 constraints. Cut constraints are added to the Stage 1 problem after every iteration of the multicut method. Each Stage 2 scenario has 60281 variables, 42822 constraints, and 220018 non-zeroes. An equivalent extensive formulation of the stochastic program with 120 scenarios has 7233777 variables, and 5138658 constraints. Figure 6.3 shows the time it takes to solve the extensive formulation of the stochastic program using the parallel Gurobi IP solver. Gurobi has a

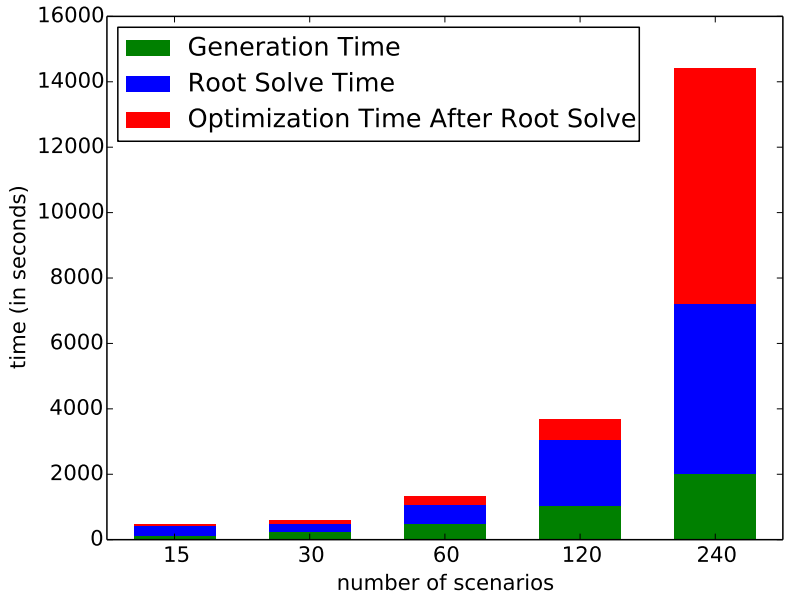


Figure 6.3: Time to solution of the extensive formulation of the stochastic program for different number of scenarios. The extensive formulation ILP is solved with Gurobi parallel integer solver running on 12 processors. Total time is broken down in to 3 components: *Generation time* - time it takes to generate the extensive formulation ILP from individual scenarios, *Root Solve Time*- time taken by Gurobi to solve the root relaxation of the extensive formulation ILP, *Optimization time After Root Solve* - time after the root solve to obtain the integer solution that is within 1% of the optimal. Optimization of the problem with 240 scenarios was terminated after walltime limit of 14000 seconds.

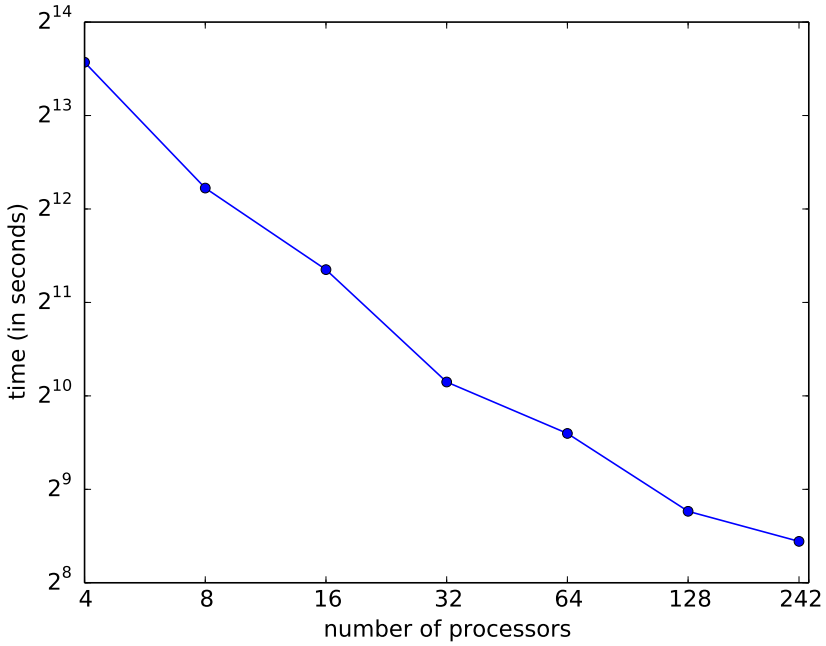


Figure 6.4: Time to solution using our distributed solver for the aircraft allocation problem with 240 scenarios.

parallel IP solver that can explore the vertices of the BnB tree simultaneously on multiple cores of the same node. In our experiments Gurobi was configured to launch 12 threads on 12 cores of the compute node. Total time to solution increases superlinearly with the increase in number of scenarios, rendering the problem intractable in the required walltime when the number of scenarios become large. Time to solve the relaxed linear program of the extensive formulation also increases superlinerarly. Therefore, benders decomposition is used to speedup the optimization time of stochastic programs.

Figure 6.4 shows the parallel speedup of the problem with 240 scenarios. Note that even with 4 processors, the solution time is lower than Gurobi solver with 12 processors. We obtain up to 57x speedup (94.2% parallel efficiency) while scaling from 4 to 242 cores. The maximum number of cores that can be used for a 240 scenario problem is 242 - one core for Stage 1 solver, one for Stage 1 manager, and 240 Stage 2 solvers for simultaneous evaluation of the 240 scenarios. During Stage 1 optimization, Stage 2 cores are sitting idle. We could use other cores on the same node as the Stage 1 solver for solving the Stage 1 mixed-integer program in parallel using Gurobi's parallel integer program solver for shared memory

machines. However, we observed that this did not give us any observable improvements in performance. This has been verified by Koch et al [53], that Gurobi integer program solver suffers from poor parallel efficiency.

We get superlinear speedups, for example, while scaling from 16 to 32 processors, the time to solution reduces from 2620 seconds to 1135 seconds. This is a result of degeneracy present in the Stage 1 and Stage 2 linear programs. Across scenarios and iterations, Stage 2 linear programs vary only in the right hand sides of the constraints. Successive optimizations can therefore be significantly faster, if the optimization starts from the basis and solution of the previous optimization. This is also called as warm start or advanced start. Using warm starts significantly reduces the optimization time. However, the optimal solution depends on the starting basis, because a degenerate program can have multiple optimal solutions with the same objective cost. Different optimal solutions can result in different dual values and hence also the cut constraints. The sequence in which scenarios are optimized in Stage 2 can also vary across multiple runs of the same problem. This is because the order in which Stage 1 manager receives work requests from Stage 2 solvers can vary because of network interference during the sending of messages and different optimization times because of operating system noise. Different optimization sequences cause different cuts to be generated, and therefore the course of convergence can vary across multiple runs of the same problem on the same cores. This causes some variation in the number of iterations to convergence across multiple runs on the same/varying number of processors and hence leading to superlinear speedups.

6.5 Summary

Our research clearly demonstrates the applicability of stochastic optimization in planning for disaster related airlift capability. Significant costs savings are obtainable when disasters are imminent as shown in Case 1. More generally smaller but still significant benefits are obtained when allocating aircraft without a specific disaster threat as shown in Case 2. Our work also shows the effectiveness of using high performance parallel computing when solving large stochastic optimization problems. We obtain speedup of 57x while scaling from 4 to 242 processors.

CHAPTER 7

Responding to disruptive events at execution time: a stochastic integer programming approach to Dynamic Mission Replanning by the Air Mobility Command of the Department of Defense

7.1 Introduction

The most dynamic component of the department of defense's (DoD) logistics domain involves airlift operations. As discussed in Chapter 6, the airlift missions are subject to a great deal of uncertainty in the timings and amounts of demands that they have to serve. Further complicating matters are disruptions that arise due to airfield closures on account of weather or other events and aircraft breakdowns.

The management of DoD's airlift operations is the responsibility of the 618th Tanker Airlift Control Center (TACC), an element of AMC. The AMC publishes a planned schedule which consists of a list of itineraries and a pairing of aircraft and crew with these itineraries. This plan is typically generated with a nominal set of anticipated demands and the published

schedule also includes details of how much of a demand is served by any given leg of any given itinerary. At the time of execution of this schedule the barrel master must take into consideration the nature of actually realized demands and the associated cargo and passenger quantities. Anticipated and already occurring service- and weather-related disruptions must also be taken into consideration. Therefore the barrel master must impose some changes on the published schedule. However, she must aim to minimize the impact of these changes on crew schedules, delivery delays, and the number and magnitude of further *future* changes. The current approach used by the AMC relies on experienced barrel masters who use various visualization tools to identify and eliminate infeasibilities in the schedule. In particular, such an approach is myopic and ignores the impact of the changes on future missions.

This chapter describes an analytical approach to the barrel master's problem. Given that different instances of this problem need to repeatedly be solved at *execution time*, one desires a solution approach that finds the optimal solution *quickly*. We present the details of our modeling and computational approach. Within the framework of optimization, there are several possible approaches. For example, one may wish only to eliminate the current set of infeasibilities in the system or one may focus only on a 'most likely' future scenario and optimize the schedule for that. Finally, one may consider several (probability weighted) scenarios and try to find the schedule changes that not only restore feasibility but also optimize the 'expected value' of the objective. We propose the third approach for this problem and compare it with the other two approaches.

However, there are some significant challenges that arise with this approach. For example, with more scenarios, the problem size increases beyond the capability of sequential machines. Secondly, one has to formulate scenarios in such a way that we maintain fidelity with reality while also keeping the model structure simple enough to analyze and code. Furthermore, standard packaged MIP solvers scale quite poorly as problem size increases. We use our parallel PSIPS solver (described in Chapter 3) to solve these large sized stochastic mixed-integer programs. PSIPS solver exploits scenario parallelism, and branch-and-bound parallelism on a distributed, high-performance cluster to solve these problems in a timely manner. The major contributions of this work are the mathematical modeling of the very complex Dynamic Mission Replanning (DMR) problem of the US AMC. We propose a com-

putationally feasible stochastic optimization formulation for this problem and demonstrate how this outperforms the myopic and deterministic optimization approaches.

This chapter is divided into 11 sections. Section 7.2 does a brief literature review on optimization of airline operations. Section 7.4 gives a detailed description of the military dynamic mission replanning problem. Section 7.3 gives a description of the various terms used in the chapter. We then discuss the prominent features of our model and the modeling approach in Section 7.5 and Section 7.6, respectively. The proposed stochastic formulation is explained in Section 7.6. Some implementation details are mentioned in Section 7.8. This is followed by the experimental setup (Section 7.9) and the computational results (Section 7.10). Finally, the findings of the chapter are summarized in Section **??**.

7.2 Related Work

Most of the research in optimizing airline operations has focused on cargo and passenger movement in the commercial airline sector; please see papers by Barnhart [102–104], Klabjan [105–113], and Nemhauser [114,114–136] and the references therein for an overview. Mulvey et al. [137] give an overview of robust optimization approaches to several real-world problems including air-force airline scheduling and describe the suitability of parallel and distributed computer architectures for the solution of such models. In the last chapter (Chapter 6) we demonstrated the applicability of parallel computing to Air Mobility Command's aircraft allocation problem, which is modeled as a stochastic mixed-integer program.

Baker et al. [138, 139]) discuss military applications of airlift optimization; McGarvey et al. [140] at RAND corportation developed an optimization model examining the cost-effectiveness of commercial intratheater airlift (CITA) movements in the U.S. Central Command (USCENTCOM) area of responsibility. Approaches such as those in the master's thesis of Williams [141] share some characteristics with ours, for example, route and itinerary generation is not a part of the optimization model – these are generated separately in accordance with various business rules. In the master's thesis [142], Pflieger indicates an optimization approach to minimizing the cost of a single airlift mission which requires air-refueling. In Stojkovic et al [143], an extension of PERT/CPM models is proposed for solving "Day of

Operations Scheduling" problem. However, this model only allows small ground delays as recourse decisions and as such it can only handle 'small' disruptions. In the master's thesis [144], Kopp adapts the approaches used in the motor carrier industry to AMC's context and seeks to optimize the operating ratio. In [145], Kramer et al describe AMC's Barrel Allocator tool which assigns missions to air wings. An assignment specifies an air crew and an aircraft of a specified type at a particular base. The Air Tasking and Efficiency Model (ATEM), proposed by Brown et al [146] is a deterministic optimization model that has been used for a few years by the US CENTCOM. Rushmeier et al. [147] model and handle schedule planning, fleet assignment, and crew scheduling as three separate steps in the commercial sector. Smith et al. [148] present an incremental optimization approach for the Barrel Master's problem at Air Mobility Command (AMC). Wilkins et al [149] propose a decision support system for AMC flight managers that identifies disruptions that require corrective actions and offers suggestions for dynamic rescheduling of missions. Finally, Wu et al [150] give a unified view of various simulation and optimization approaches that one may take for airlift problems.

7.3 Terminology

In this section we give precise definitions of some of the terms that we use throughout this chapter.

MOG This stands for Maximum On Ground space for aircraft parking at a base. This could either be specific to the aicraft type (for example, two wide body aicraft, three narrow body aircraft) or an aggregated total amount of space. Rules are available for converting parking space for one type of aircraft to parking space for another aircraft.

Itinerary An *itinerary* is an aircraft type or tail number together with an alternating sequence of legs and stops; the legs specify their flying time, load capacity, fuel requirement, crew requirement, and flying cost; the stops specify the minimum stop length, fuel requirement, etc.

Demand A *demand* consists of a point of embarkation, a point of disembarkation, an earliest time (epoch) at or after which the demand may be loaded onto an aircraft, an earliest arrival time at which the demand may be delivered at its destination, and the latest arrival time at which the demand may be delivered without incurring a non-delivery penalty.

Weather event This is a disruption event that leads to unavailability of an airbase for either landing or takeoff or both. The event description consists of the affected airbase, the time at which the event becomes known, the start time of the event, the duration for which the event affects the airfield, and an alternative location (generally close to the affeced base) where the itineraries destined to the affected base can be diverted if required.

Breakdown event This is a disruption event due to an aircraft breakdown. It consists of the itinerary number of the affected aircraft, the stop number of the itinerary at which the aircraft breaks down, and the time required to repair the aircraft.

Command and Control delay event These are either TACC or other command-directed delays of an aircraft at a base. paper work.

Demand event This event corresponds to any change in the existing demand, for example, change of load, change in latest arrival time, etc. A *demand event* can also include addition of a new demand.

Resource event A *resource event* corresponds to a change in the available resources for carrying out the planned set of itineraries. The change could be in the available amount of MOG space at a base, the number of aicraft, the number of crews or the amount of fuel at a base.

7.4 Dynamic Mission Replanning

90% of the airlift missions by US Air Mobility Command (AMC) do not execute as planned and approximately 5% of them end up with delays[1]. The missions are disrupted because of 1) command and control delays, 2) airfield and weather delays, 3) cargo requirement changes and cargo delays, and 4) aircraft and crew delays. In the occurrence of such disruption events, all the missions need to be adjusted. For example, consider a situation in which a Tanker Airlift Control Center (TACC) duty officer (Figure 7.1) receives a phone call informing him that the C5 aircraft, tail number 451, that was about to takeoff in an hour from Dover airbase for Tripoli International airbase has been called off. The plane is still loading and will be ready to takeoff in three hours from now. The job of the duty officer in such a situation is to consider the effect of this event on this mission and all other missions around the world.

Figure 7.1: TACC duty officer

While doing this, the duty officer must consider the following at all the airbases: 1) Maximum on Ground (MOG) parking space limitations, 2) weather events, 3) cargo and passenger available times for loading, 4) target delivery time and priorities of cargo/passenger, 5) fuel availability, 6) air refueling requirements, 7) country over-flight permissions, etc. The current systems available to TACC officers for doing mission replanning consists of a set of applications that provide an effective way of visualizing various factors that could result in delay or deviation of a planned mission. These help the duty officer to decide the recourses for all the affected missions. The decisions are taken ad-hoc and need not be globally optimal.

[1]Source: DMR Business Case Analysis

Additionally, the process is very cumbersome, error-prone and leads to cumulative delays in mission. This also leads to crew dissatisfaction (which impacts crew retention rate), fuel costs inefficiencies, inefficient cargo velocity, and so on. Hence, there is a general agreement that AMC missions should be optimized for variables that are within AMC control like the mission itineraries that are selected, the aircraft type chosen for each mission, recourse decisions that are taken at the time of disruptions, etc. Business rules and realities that are outside of AMC control constitutes the constraints and input parameters of the optimization problem.

In this section, we give a more detailed description of the U.S. Dynamic Mission Replanning (DMR) problem. As the name suggests this is concerned with recovery from disruptions. The basic outline of the problem is as follows. We are given the following pieces of information:

7.4.1 Static Information

There are resources available with the AMC that it can use for executing the planned set of missions. These include the MOG space at each of its airbase at any given time, fuel influx at each airbase, aircraft and crew availability at a base at any given time. In addition, the static information includes the currently executing schedule, consisting of:

1. A list of itinerary that are flying (or are scheduled to be flying).
2. The list of demands that were known when the currently executing schedule was generated.
3. The crew and aircraft assignments to various itineraries.
4. The amount of each load (demand) that each itinerary carries on its various legs.

7.4.2 Dynamic Information

The dynamic information consists of all the dynamic elements that can disrupt the planned execution of the schedule. This consists of:

- A list of disruptive events that affect the current schedule. These effects include making the cost too high or even making the currently planned schedule infeasible.
- A list of newly realized demands, or changes in the amounts of various demands, since the currently executing schedule was generated.
- The changes in amount of available resources from the time when the currently executing schedule was generated.
- A list of future scenarios, which consist of demands (which may be either new or modified versions of known demands) and disruptions, along with their associated probabilities of occurrence.

As noted above, the presence of disruptive events can make the currently excuting schedule impractical, expensive, or even infeasible. The primary goal of dynamic mission replanning is to make replanning decisions that make the modified schedule *feasible*. Secondly, we aim to take the possible *future* scenarios into consideration and come up with a modified schedule that has a high chance of either remaining feasible or requiring fewer changes when future disruptions occur. Finally, we also wish to make sure that we do it as cheaply as possible.

Therefore, the output of any DMR algorithm must include the following:

- A modified schedule. This consists of:
    1. A mapping that indicates what changes, if any, are to made to the previous itineraries, loads, and aircraft/crew assignments.
    2. A list of newly added itineraries, loads, and aircraft/crew assignments.

7.5 Model Features

In this section, we discuss some of the prominent features or methods that we took to formulate the DMR problem. At first, we discuss the way we define a scenario and various types of disruption events in our modeling approach. This is followed by the description of the disruption handling.

7.5.1 Scenarios

An important question is: what constitutes a scenario in the context of this problem. We define a scenario to be a random variable $\omega = (\omega_r, \omega_d, \omega_w, \omega_b)$ as follows:

1. The component ω_r of ω characterizes the state of resources (aircraft, crew, fuel, etc.) at the time of execution i.e. the available resources could be different from what was assumed at the time of planning in Stage 1. Any change in the resources is characterized as a Resource Change Event (RscChngEve).

2. The component ω_d of ω characterizes a specific set of realized demands. Any change in existing demand or addition of a new demand is characterized as a Demand Event (DmdEve).

3. The component ω_w of ω characterizes a specific set of realized events that lead to a base becoming unavailable for landing or take-off or both, for example, weather events. We call these types of events as Base Unavailable Event (BaseUnavailEve). For a weather event, we assume that it has the following features: t_a a time at which we become aware of the (imminent) weather event, $0 \leq t_a \leq t_b$ a time at which the weather event begins, and $d > 0$ the duration of the weather event, and l a location (base) at which the weather event occurs. Furthermore, we assume that the effect of a weather event is simply to make the specified location unavailable for takeoffs and landings for the duration of the weather event (t_b to $t_b + d$). We also have the potential to handle reduced landing and take-off capacities at a given base.

4. The component ω_w of ω characterizes a specific set of realized on ground delays, for example, aircraft breakdowns. We call these types of events as Ground Delay Event (GndDelayEve). We assume that a GndDelayEve has the following features: $\mathcal{I}$ an itinerary thatgets delayed, $\mathcal{K}$ a leg of the said itinerary during which the delay occurs, $\mathcal{D}$ a numerical measure of the event intensity, expressed as the time delay that is introduced into the itinerary because of the breakdown.

Scenario formation is depicted in Figure 7.2. We have a set of Resource Change Events (RscChngEves), Demand Events (DmdEves), Base Unavailable Events (BaseUnavailEves),

and Ground Delay Events (GndDelayEves). We take a cartesian product of all of these event sets to get as many as #RscChngEves × #DmdEves × #BaseUnavailEves × #GndDelayEves scnearios.

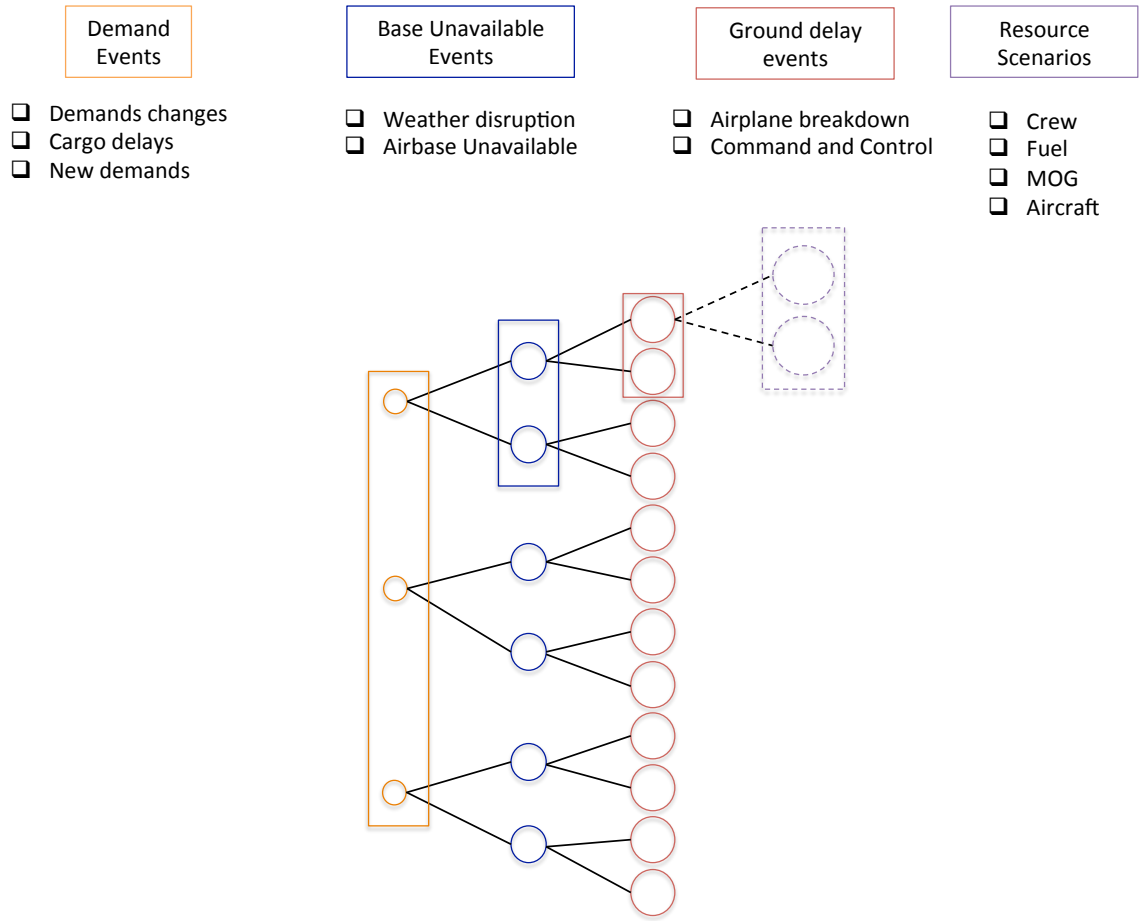


Figure 7.2: Different types of disruption events and scenario formation

7.5.2 Handling Disruptions

It is easy to check which itineraries are *directly affected* and thus must be modified in light of the disruption events in a given scenario. Therefore, for each itinerary i we precompute and keep a continuation set of 'derived' (or recourse or continuation) itineraries, denoted $\mathcal{D}_{i,t}$ that will be chosen as the continuation if i is (known to be) directly affected at time t. These derived itineraries include a set of delayed continuations as well as rerouted continuations. In particular this set also includes itineraries that get terminated before reaching their final

destination.

1. We note that if a breakdown event affects an itinerary i then we assume that it affects all itineraries in all its continuation sets.

2. At this stage we have some flexibility in deciding weather we reroute or delay only those itineraries that are "directly affected" (keeping the rest of the schedule unchanged) or if we should allow a greater set of itineraries to be modified with the view that this might yield a better or more robust schedule given the altered state of the system. To this end, we may wish to compute $\mathcal{D}_{i,t}$ for all the selected itineraries that fall in the so-called "cone of causality" of the given disruptive event. (See 'light cone' for a fuller motivation, replacing the speed of light by a suitably tight upper bound for the speed of the jets in question.)

Some other features of the $\mathcal{D}_{i,t}$ set are:

1. Itineraries in $\mathcal{D}_{i,t}$ can only start at or after t and only at the location of itinerary i at that time.

2. Itineraries which simply introduce a delay of a specified length into i at all times after t but are otherwise identical are included in the set $\mathcal{D}_{i,t}$.

3. Also included are itineraries that get rerouted to different (nearby) locations at and after time t.

7.6 Modeling Approach

In this section, we discuss the various ways of formulating the DMR problem.

7.6.1 A naïve approach

A naive approach to the DMR problem is to simply solve a *deterministic* optimization problem at each time t when we become aware of a disruptive event, *i.e., assuming that*

no further disruptions will occur in the future. Therefore, at the first time of awareness of a disruption event, we simply try to optimally find a feasible set of recourse itineraries while assuming that there will be no disruptive events. The solution (i.e., new execution schedule) that is thus obtained is followed until the awareness time of the next disruption event. In other words, this solution gives us the state of the system at all times prior to the next disruptive event (or the next time at which we have the awareness of an imminent disruption). At the next time of awareness we then try to find an optimal schedule that takes us from this state to the desired final state (or as close to it as possible) by choosing itineraries from a feasible set obtained by replacing the affected itineraries by their continuation sets. To disincentivize wholesale changes to the existing schedule we might wish to include the ℓ_1 distance from the original schedule in our objective function.

Therefore, once we are given a scenario ω, we know the number of disruptive events that show up in this scenario. Let's denote this number by N_ω. Now consider the "time 0" problem: We solve the following problems at each of the N_ω 'times of awareness':

$$\begin{aligned} &\min c_k y_k \\ &A_k y_k = b_k - L_k y^*_{k-1} \end{aligned}$$

where $1 \leq k \leq N_\omega$, and y^*_{k-1} is the optimal solution of the 'previous' problem (that is solution from the previous disruption event).

7.6.2 A 'prescient' approach

In this approach, given a scenario, we assume that $t_a = 0$ for all weather events and also that we have advance knowledge of all future breakdowns at time 0. Therefore we might, in principle, be able to avoid all disruptive events by just planning around them. As such, this approach is rather more unrealistic than the ones indicated above but it is nonetheless useful for providing a theoretical benchmark or bound on well one might be able to do.

7.6.3 A stochastic optimization approach

At the time of execution, we can easily imagine a scheduler that **does not** naïvely solve a deterministic optimization problem but instead solves a stochastic optimization problem. However, such a scheduler has no knowledge of the *number* of disruptive events that can still occur in her planning horizon[2].

In other words, at the k^{th} awareness time of a disruptive event, the scheduler solves a stochastic program of the form

$$\min_{y_k} E_{\bar{\eta}_{k+1}} \left[Q_k(y_k, \bar{\eta}_{k+1}) \right]$$
$$L_k y^*_{k-1} + A_k y_k = b_k$$

for $k \in \mathbb{Z}_{\geq 0}$ where $\bar{\eta}_{k+1} = (\eta_{k+1}, \eta_{k+2}, \dots)$ denotes the future uncertainty, and at $(k+1)^{\text{th}}$ event only η_{k+1} becomes known. If we assume that there are no more than N disruptive events in any planning window, then we have no more than N stages and the N^{th} stage problem becomes deterministic:

$$\min_{y_N} Q_N(y_N)$$
$$L_N y^*_{N-1} + A_N y_N = b_N$$

Clearly, this approach is in some sense the most rational amongst the ones that we have listed, however, it is also quite unrealistic and computationally demanding. We don't expect that a scheduler, at execution time, is likely to have the computational resources necessary to follow such a procedure. Therefore, we formulate the problem as a two-stage stochastic program. Consider a monolith problem in which all the LP's that are solved at each event awareness time are put together into a single LP. The itinerary variables from the previous event LP become the right hand sides of the next event LP (which in the naive approach are constant values derived from the optimal solution of the previous LP).

[2]More on that in the next sub-section.

Below is shown the structure of the aggregated monolith LP (the objective function is largely immaterial):

$$\begin{aligned}
&\min \sum_{k=0}^{N} c_k x_k \\
&L_1 x_0 + A_1 x_1 = b_1 \\
&\vdots \\
&L_N x_{N-1} + A_N x_N = b_N
\end{aligned}$$

where, N is the number of events in the scenario and x_0 is the stage 1 solution.

7.6.4 Moving horizons, replanning at fixed intervals, and combinations thereof with previous approaches

Finally, doing replanning at each disruption event can be too expensive and may require a lot of changes to the schedule because of the large number of events involved in a real-case scenario. We indicate an approach which we claim is realistically implementable. We summarize this approach as follows:

- Pick a sufficiently fine, discrete set of times, say $\{0, \delta t, 2\delta t, \ldots, M\delta t\}$, at which we review the available information and plan recourses, as necessary.
- Pick the length of a fixed window of time, say Δt, in the future, and a suitable performance measure Q on this time-window on which the scheduler tries to optimize the performance given the information available.
- At each time $k\delta t$, check if new information is available that affects the currently planned scheduled. If so, re-plan the schedule for the period $(k\delta t, k\delta t + \Delta t]$ that optimizes the performance measure Q.

7.7 Model Formulation

We have a whole host of modularized functions that compute model parameters using the input data. This allows us to satisfy many requirements, for example those concerning the requirement of an augmented crew or for satisfying rules related to hazardous cargo, without needing to explicitly include complicated constraints.

- Each index j denotes a set of aircrafts. This set is either a singleton (if we wish to indicate a specific jet) with a (unique) tail number or it contains multiple jets of the same 'type' (if the identity of individual jets is not important). In general, we wish to track the identity of individual jets when they have been assigned to flying itineraries but not of those that are ready to handle disruptions but are not assigned specific itineraries or missions.

- Each index k denotes a set of crews and again, this set is either a singleton (if we wish to indicate a specific crew) or it contains multiple crews of the same 'type' and qualifications (if the identity of individual jets is not important). In general, we wish to track the identity of an individual crew when they have been assigned to scheduled itineraries but not of those that are ready to handle disruptions but have not been assigned specific itineraries or missions.

- To handle itinerary legs that require an 'augmented crew', for example an additional pilot or loadmaster but not an additional 'full' crew, we include a set of 'dead-heading' variables $c^d_{kll't}$ which denotes the number of crews of type k who are dead-headed to location l from location l' at time t. It is assumed that the dead-heading time $t^d_{ll'}$ between any two locations is known in advance.

- Our itinerary descriptions include aircraft configurations by hop and thus the parameter b_{fih} can be inferred. For example, if a particular leg h of itinerary i is configured for carrying hazardous material, the coefficient b_{fih} is 1 if and only if a demand f consists of hazardous material. This allows us to satisfy the requirement that a cargo consisting of hazardous material may not share a plane with non-hazardous cargo.

- Similarly, if an itinerary requires air-refueling (which is encoded by $\nu_{ih} \neq 0$ for some hop h (which goes from l to l') on itinerary i), then the crew for such an itinerary, at least on the relevant hop, must be suitably qualified. In other words, η_{iklt} must be set to zero for the location l unless crew-type k is qualified to handle air-refueling.

Variables The decision variables in the model can be divided into the following three types:

- Itinerary variables,
- Load variables, and
- Resource slacks

The **itinerary variables** are associated with either newly added itineraries (x_i^{new} in stage-1 and $y_i^{e_s,\text{new}}$ in stage-2) or recourses for previously scheduled itineraries ($x_{ii'}$ in stage-1 and $y_{ii'}^{e_s}$ in stage-2). The **load variables** are similarly associated with either previously loaded demands ($z_{fii'}$, μ_{fi} and $u_{fii'}^{e_s}$, $\mu_{fi}^{e_s}$ respectively) or with demands that have newly materialised (z_{fi}, μ_f and $u_{fi}^{e_s}$, $\mu_f^{e_s}$ respectively). Finally, we have various **resource slacks** which represent the amount of unused resources of various kinds (aircraft, crew, refueling tankers, ground space, etc). [3]

Constraints The constraints in in the model can be divided into the following three types:

- Aircraft capacity constraints: these ensure that the load on any aircraft does not exceed its capacity (in area, weight, or the number of passengers).
- Ground resource constraints: these constraints include ground space, fuel, and crew constraints. Fuel constraints include in-air refueling. Also included are 'aircraft constraints': the number of aircrafts assigned to itineraries cannot exceed the available number of the aircrafts at various locations.

[3]We also allow for the movement of some of these resources from one location to another via crew dead-heading.

- Demand constraints: these ensure that the demands are loaded onto the correct itineraries. We also allow for and include transshipment constraints.

- Recourse itinerary flow constraints: these constraints ensure that whenever an itinerary is disrupted, we have a recourse for it.

Our model is a (large-scale) stochastic mixed-integer program. We take a scenario-based approach to handle uncertainty. This variability manifests in four ways - RscChngEves, DmdEves, BaseUnavailEves, and GndDelayEves. These events have been described in Section 7.5. The nature of these sources of uncertainty and the types of real-life situations that can be handled by our framework have also been discussed there. Our general approach to modeling the DMR problem consists of externalizing the (potentially changeable) business rules to an itinerary generator and focusing attention on selecting a suitable set of itineraries which perform well on average and satisfy the capacity and resource constraints. Disruption events such as BaseUnavailEves and GndDelayEves only changes the candidate itinerary set supplied to the Stage 1 or Stage 2 LP, that is, only feasible itinerary sets are supplied to the LP. This approach handles the disruptions outside of the LP and thus avoids making wholesale changes to the available resources in the LP model.

Demand variability (DmdEve) We consider three sources of variability for demands:

- Whether a demand materializes at all.

- The amount of demand that materializes.

- An altogether new demand has to be carried.

Whenever a new demand has to be carried, it leads to generation of new itineraries that can carry that demand. These itineraries are added as new itineraries, $x_i^{new} ory_i^{e_s,new}$, to the linear program, along with the newly added demand. The LP program then chooses to fly one or more of these new itineraries and/or adjusts the new demand into one of the exisiting itinerary in the execution schedule.

Location-specific disruptions (BaseUnavailEve) The types of disruptions that we call 'location-specific disruptions' in this article are any disruptions that fit the following framework:

- The disruptive event has a location associated with it.
- The disruptive event has the following times associated with it:
    1. A time of awareness, denoted t_{aware}.
    2. A start time, denoted t_{start}.
    3. An end time, denoted t_{end}.

The consequence of a location-specific disruption is that the associated base (location) becomes unavailable to either landings or take-offs or both for all times between t_{start} and t_{end}.

It is worth noting that this definition is quite general and it can encompass any other disruptions (such as weather-related outages) that fit this description.

Aircraft-specific disruptions (GndDelayEve) The types of disruptions that we call 'aircraft-specific disruptions' in this article are any disruptions that fit the following framework:

- The disruptive event has an itinerary i (and hence an aircraft) associated with it.
- The disruptive event has the following information associated with it:
    1. A 'leg number' k of the itinerary, at whose conclusion (i.e., at landing) the disruption occurs.
    2. A minimum length of time, denoted t_{ground} for which the aircraft must remain on ground before next take off.

The consequence of an aircraft-specific disruption is that the associated aircraft (itinerary) must remain on ground for all times between t_{landing} and $t_{\text{landing}} + t_{\text{ground}}$, where t_{landing} is the time at which itinerary i lands for the $k-$th time.

It is worth noting that this definition is quite general and it can encompass any other disruptions (such as aircraft-breakdown or a command and control event) that fit this description.

Remark 1. *If an itinerary i is associated with an aircraft-specific event e, we assume that all its recourse (or replanned) itineraries (generated for other any other event e' that occurs before e) are also associated with e. This ensures that model cannot avoid aircraft-specific events simply by altering the associated itinerary at an earlier time.*

Resource variability (RscChngEve) This consists of any change in the amount of available MOG, number of aircraft, number of crews, amount of fuel, etc. that can lead to any change in the current execution schedule. These events would involve change in the right hand side constants of the LP, and new/ modified itineraries that can be feasibly flown with the updated available resources. Handling these events wourld require a richer itinerary generator. Currently, we do not handle these types of events in our simulations.

7.7.1 Stage 1

7.7.2 Input parameters

$\mathcal{TS}$	: set of transshipment locations
$\mathcal{SB}$	: set of locations that are 'small' bases (have sub-limit on widebody aircraft)
$\mathcal{LJ}$	: set of 'jet-types' (aircraft) that are classified as 'large' (widebody)
J_j^L	: 1 if 'jet' j is a widebody (i.e., if $j \in \mathcal{LJ}$)
S_{it}	: 1 if t is the start time for itinerary i
E_{it}	: 1 if aircraft from itinerary i becomes available for use by another itinerary at time t
J_{ij}	: 1 if j is the 'jet' for itinerary i
H_{il}	: 1 if l is the home-base for itinerary i
T_{ilt}	: 1 if itinerary i takes-off from location l at time t

L_{ilt} : 1 if itinerary i lands at location l at time t

Y_{il} : 1 if location l is a home-base or staging location for itinerary i

$civa(l,t)$: set of civilian itineraries that arrive at location l at time t

$mild(l,t)$: set of military itineraries that depart from location l at time t

D_f : total units (mass) of cargo in demand f

χ_{fi} : area (sqft) required per unit mass of cargo in demand f when flown on itinerary i

Typically independent of itinerary, except in the case of passengers, where only some aircraft (and hence, itineraries) use the cargo area for passengers

ρ_{fi} : units of passenger area required per unit mass of cargo in demand f when flown on itinerary i

Typically independent of itinerary, only needed for some aircraft such as C5

$legs(i)$: set of legs (departure-arrival location pairs) in itinerary i

K^m_{ih} : capacity (in units of mass: tonnes) of itinerary i's aircraft along leg or 'hop' h

K^a_{ih} : capacity (in units of area: sqft) of itinerary i's aircraft along leg h

P_f : penalty for every unshipped unit of demand f (units of mass)

R_k : recuperation time (crew rest) for a crew of type k (i.e., a crew that can serve on a 'jet' of type j)

$\mathcal{J}_k$: the set of 'jets' that a crew of crew-type k can serve

S_{jk} : 1 if a crew of type k can serve a 'jet' of type j (i.e., if $j \in \mathcal{J}_k$)

R'_j : recuperation (turn) time for a 'jet' of type j

F_{ilt} : fuel requirement of itinerary i at location l at time t (it is 0 when l is home-base)

F'_{ih} : fuel requirement of itinerary i on leg h where $h \in rhop(i)$

ν_{ih} : fuel requirement of itinerary i on leg h expressed in number of tanker-loads

G_{lt} : fuel added to location l at time t

M_{lt} : MOG added to location l at time t

M^L_{lt} : MOG for widebody aircraft $j(\in \mathcal{LJ})$ added to location l at time t

N_{jlt} : number of new 'jets' of type j added to location l at time t for assignment to new itineraries

C_{klt} : number of new crew of type k added to location l at time t

$gs(i)$: returns the ground-space requirement of the aircraft flying itinerary i

$rhop(i)$: set of legs in itinerary i when the aircraft requires refueling

$rloc(i,h)$: set of locations from which a tanker can fly to refuel itinerary i on leg h

$rdep(l,i,h,t)$: 1 if the tanker departs from location l at time t to refuel itinerary i on leg h

$rarr(l,i,h,t)$: 1 if the tanker arrives back at location l at time t after refueling itinerary i on leg h

τ_{ilt} : 1 if a tanker needs to fly from location l at time t to refuel itinerary i

ρ_{ilt} : 1 if a tanker lands at location l at time t after refueling itinerary i

ζ^e_i : cost of flying itinerary i at disruption event e

$reci(i,e)$: set of recourse itineraries of itinerary i at the time of of awareness of event e

J^e : set of itineraries selected during scheduling at the time of awareness of event e

I^e : set of currently scheduled itineraries that are affected by all events whose $t_{aware} < t^e_{aware}$ and $t_{impact} \geq t^e_{aware}$

I'^e : set of itineraries that are not affected by the awareness of event e $J = I^e + I'^e$

$itin(i,f,e)$: set of recourse itineraries at the time of awareness of event e that can carry demand f currently being flown by itinerary i

X_i : equal to the optimal x_i that is decided prior to event e

Z_i : equal to the optimal z_i that is decided prior to event e

7.7.3 Inferred Parameters

ϕ_{ijklt}	1 if itinerary i has jet-type j, crew-type k, home-base at location l and starts at time t	$S_{it}\mathcal{J}_k J_{ij} H_{il}$
ψ_{ijklt}	1 if itinerary i has jet-type j, crew-type k, home-base at location l and terminates at time t	$E_{it}\mathcal{J}_k J_{ij} H_{il}$
η_{ijklt}	1 if itinerary i having crew-type k takes-off from a home-base or staging location l at time t	$\mathcal{J}_k J_{ij} T_{ilt} Y_{il}$
γ_{ijklt}	1 if itinerary i having crew-type k and jet-type j lands at home-base or staging location l at time t	$\mathcal{J}_k J_{ij} L_{ilt} Y_{il}$
θ_{ilt}	1 if itineray i lands at a non home-base location l at time t	$L_{ilt}(1 - H_{il})$
ω_{ilt}	1 if itineray i takes-off from a non home-base location l at time t	$T_{ilt}(1 - H_{il})$

7.7.4 Variables

x_i^e : ith itinerary variable for replanning at disruption event e

z_{fi}^e : units of cargo (mass) from demand f carried by itinerary i after awareness of event e

$z_{fii'}^e$: units of cargo (mass) from demand f carried by itinerary i after awareness of event e
that was previously being flown by itinerary i'

μ_{if}^e : undelivered cargo of demand f that was being flown by itinerary i before awareness of event e

μ_f : units of cargo (mass) from demand f that remain unshipped after scheduling

u_{flt} : units of cargo of demand f at location l ($l \in \mathcal{TS}$) that remain unshipped at time t

a_{jlt}^u : unused 'jets' of type j at location l after time t

c_{klt} : unassigned crews of type k at location l after time t

f_{lt} : leftover fuel at location l at time t

m_{lt} : unoccupied combined MOG for at location l after time t

m'_{lt} : unoccupied MOG for widebody aircraft at location $l(\in \mathcal{SB})$ after time t

v_{ihl} : integer variable, the number of tankers to refuel itinerary i on leg h that fly from location l

t_{lt} : number of unused tanker aircraft at location l and time t

7.7.5 Stage 1 Integer Program:

Objective Function

$$\text{minimize} \quad \sum_i \zeta_i^e x_i^e + \sum_i \sum_{lt} \Gamma_l^f F_{ilt} x_i^e + \sum_s p_s Q_s$$

where Q_s is the optimal value of the stage-2 objective in the scenario s which occurs with probability p_s.

Aircraft Constraints

$$\sum_{i,k} \phi_{ijklt} x_i^e - \sum_{i,k} \psi_{ijkl(t-R'_j)} x_i^e + a_{jlt} - a_{jl(t-1)} = N_{jlt} \quad \forall j,l,t$$

Crew Constraints

$$\sum_{i,j} \eta_{ijklt} x_i^e - \sum_{i,j} \gamma_{ijkl(t-R_k)} x_i^e + c_{klt} - c_{kl(t-1)} = C_{klt} \quad \forall k,l,t$$

Fuel Constraints

$$\sum_i F_{ilt} x_i^e + f_{lt} - f_{l(t-1)} + \sum_{i,h \in rhop(i)} F'_{ih} rdep(l,i,h,t) v_{ihl} = G_{lt} \quad \forall l,t$$

Refueling constraints

$$\sum_{l \in rloc(i,h)} v_{ihl} - \nu_{ih} x_i^e = 0 \quad \forall i, h \in rhop(i)$$

$$\sum_{i,h \in rhop(i)} rdep(l,i,h,t) v_{ihl} - \sum_{i,h \in rhop(i)} rarr(l,i,h,t) v_{ihl} + t_{lt} - t_{l(t-1)} = N_{jlt} \quad \forall l,t,j = \text{tanker-type}$$

Maximum on Ground

$$\sum_i gs(i) \theta_{ilt} x_i^e - \sum_i gs(i) \omega_{ilt} x_i^e + m_{lt} - m_{l(t-1)} = M_{lt} \quad \forall l,t$$

$$\sum_{i,j} gs(i) J_j^L J_{ij} \theta_{ilt} x_i^e - \sum_{i,j} gs(i) J_j^L J_{ij} \omega_{ilt} x_i^e + m'_{lt} - m'_{l(t-1)} = M_{lt}^L \quad \forall l \in \mathcal{SB}, t$$

Demand Constraints (for Event LPs)

$$\sum_{i \in \text{ itin}(i',f,e)} z^e_{fii'} + \mu^e_{fi'} = Z^{e-1}_{fi'} \quad \forall i' \in J^{e-1}, f$$

Capacity Constraints (weight and area)

$$\sum_f a_{fih} z^e_{fi} - K^m_{ih} x^e_i \leq 0 \quad \forall i, h \in legs(i)$$

$$\sum_f a_{fih} \chi_{fi} z^e_{fi} - K^a_{ih} x^e_i \leq 0 \quad \forall i, h \in legs(i)$$

Capacity Constraints (passengers)

$$\sum_f a_{fih} \rho_{fi} z^e_{fi} - K^p_{ih} x^e_i \leq 0 \quad \forall i, h \in legs(i)$$

Transshipment Constraints

$$\sum_{i \in civa(l,t)} z^e_{fi} + u_{fl(t-1)} - \sum_{i \in mild(l,t)} a_{fihop(f,i)} z^e_{fi} - u_{flt} = 0 \quad \forall l \in \mathcal{TS}, t, f (\text{s.t. dest(f)} \neq l, \text{ orig(f)} \neq l)$$

Recourse Itineraries Flow Constraint

$$\sum_{i \in \text{ reci}(i',e)} x^e_i = X^{e-1}_{i'} \quad \forall i' \in I^e$$

$$x^e_{i'} = X^{e-1}_{i'} \quad \forall i' \in I'^e$$

7.7.6 Stage 2

In this section we describe our approach to solving the stage 2 *execution* problem. We use the stage-1 solution as parameters for the sequential LPs that arise in each scenario s. In each scenario s an Stage 2 LP is generated for each disruption event e'_s at time $t^{e_s}_a$, which is the time at which event e_s becomes known. If any of the selected itineraries is affected by the disruption, a set of disrupted itineraries are generated for each of the affected itineraries. The part of the affected itinerary before time $t^{e_s}_a$, has already been executed and hence the disrupted itineraries differ only in their schedule after time $t^{e_s}_a$. The resource constraints in Stage 2 LP are written for time $t > t^{e_s}_a$. In case the aircraft is flying (i.e. is in air) at $t^{e_s}_a$, the corresponding demand variable ($u^{e_s}_{fi}$) will correspond to the demand that it carries after it's next landing location. Similarly, the capacity and area constraint, for such aircraft, are to be written for the legs following the current leg the aircraft are flying. Note that the demands will get reassigned at each such disruption event, which is a costly affair as it

requires unloading and loading of cargo. Therefore, either we do not allow cargo reassignment by restricting the cargo only to the recourse itineraries of the affected itinerary or we can penalize unloading and loading of cargo if assignment is allowed to other itineraries as well.

Note that there are flow constraints for each itinerary i.e. the number of planes flying a particular itinerary are distributed only to its disrupted itineraries. Therefore, if after the disruption, we want to fly more aircraft for a particular route, that route has to be formed as a disrupted itinerary of the itineraries that can fly that route. When an event LP is solved, the demand being served by a affected itinerary is distributed amongst all the recourse itineraries that can fly that demand.

In a Stage 2 scenario, disruption events in a scenario are applied to the itineraries selected in Stage 1. All the event LPs are aggregated to form a single monolith LP (as described in Section 7.6.3). The events are selected in the order of the time they become known, that is their t_{aware} time. The events with the same t_{aware} are collected together to form a set called as Current Event Set (CES) and is applied to the currently executing schedule at the same time (t_{aware}). We define Current Itinerary Set (CuIS) as the set of itineraries flying prior to the application of the CES. Current itineraries affected (directly or indirectly) by the CES are identified and their recourse itineraries are generated. The resulting set of itineraries composed of the recourse itineraries and the unaffected itineraries from the CuIS, are called as the candidate itineraries or the Candidate Itinerary Set (CaIS). The linear program for the CES then selects the best set of itineraries for flying from the CaIS and the selected set of itineraries form the Selected Itinerary Set (SIS). The selected itinerary set then becomes CuIS for the next event. For the first event set in time, the CuIS is the set of itineraries selected in Stage 1.

In two-stage stochastic optimization, the decision made in the Stage 1, selected itineraries in our case, is fed as input to the Stage 2 for evaluation. In order for Bender's method to converge, it is necesaary that the Stage 2 coefficient matrix and objective function should not change across iterations. In order to meet this requirement, it is necessary that all the candidate itineraries in the Stage 1 linear program are considered as the CuIS in the first event set of Stage 2, even though not all will be present in the actual CuIS in any given Stage 2 optimization. Similarly, in the successive event linear programs, all the itineraries in

the CAIS from the previous event form the CUIS of the CES. Although this means that the size of the successive event linear programs will grow rapidly, but this is required in order to keep the Stage 2 linear program identical across iterations. The Stage 1 decision during the iterations is fed as right hand side to the constraints of the Stage 2 linear program (*Recourse Itineraries Flow Constraints* in Section 7.7.9). These constraints ensure that recourses only to the itineraries selected in Stage 1 are selected in this event LP. Similarly, in successive event LPs , the itinerary variables from the previous event LP form the right hand side of the *Recourse Itineraries Flow Constraints*.

7.7.7 Input parameters

In addition to the parameters that are listed in section 7.7.2, we have the following

s	: refers to a scenario; a scenario s has N_s events that are labeled e_s
$civa(l,t)$	: set of civilian itineraries that arrive at location l at time t
$mild(l,t)$	: set of military itineraries that depart from location l at time t
K^m_{ih}	: capacity (in units of mass: tonnes) of itinerary i's aircraft along leg or 'hop' h
K^a_{ih}	: capacity (in units of area: sqft) of itinerary i's aircraft along leg h
P_f	: penalty for every unshipped unit of demand f (units of mass)
P^{replan}	: penalty adding or removing an itinerary to the schedule
P^{reload}_f	: penalty for unloading and reloading a unit load from demand f
$\zeta^{e_s}_i$	: cost of flying itinerary i at disruption event e_s
$reci(i,e_s)$	: set of recourse itineraries of itinerary i at the time of of awareness of event e_s
$\mathcal{I}$	: the set of all possible itineraries that can be chosen
J^{e_s}	: set of itineraries selected during scheduling at the time of awareness of event e_s
I^{e_s}	: set of currently scheduled itineraries that are affected by all events whose $t_{aware} < t^{e_s}_{aware}$ and $t_{impact} \geq t^{e_s}_{aware}$
I'^{e_s}	: set of itineraries that are not affected by the awareness of event e_s $J^{e_s-1} = I^{e_s} + I'^{e_s}$
$itin(i,f,e_s)$	: set of recourse itineraries at the time of awareness of event e_s

that can carry demand f currently being flown by itinerary i

$X_i^{e_s}$: equal to the optimal $y_i^{e_s}$ decided at event e_s and are then (fixed) parameters for the replanning problem at event $e_s + 1$

$Z_i^{e_s}$: equal to the optimal $u_i^{e_s}$ decided at event e and are then (fixed) parameters for the replanning problem at event $e_s + 1$

7.7.8 Variables

$y_i^{e_s}$: ith itinerary variable for replanning at disruption event e_s in scenario s

a_{jlt}^u : unused aircraft of type j at location l after time t

c_{klt} : unassigned crew of type k at location l after time t

f_{lt} : leftover fuel at location l at time t

m_{lt} : unoccupied combined MOG for at location l after time t

m'_{lt} : unoccupied MOG for widebody (large) aircraft at location $l(\in \mathcal{SB})$ after time t

v_{ihl} : integer variable, the number of tankers to refuel itinerary i on leg h that fly from location l

t_{lt} : number of unused tanker aircraft at location l and time t

$u_{fi}^{e_s}$: units of cargo (mass) from demand f carried by itinerary i after awareness of event e

$u_{fii'}^{e_s}$: units of cargo (mass) from demand f carried by itinerary i after awareness of event e that was previously being flown by itinerary $i^{'}$

$\mu_{if}^{e_s}$: undelivered cargo of demand f that was being flown by itinerary i before awareness of event e

μ_f : units of cargo (mass) from demand f that remain unshipped after scheduling

υ_{flt} : units of cargo of demand f at location l ($l \in \mathcal{TS}$) that remain unshipped at time t

η_i : represents the amount of deviation (adding when not originally present or removing from schedule when originally present) in itinerary i in replanning

β_{fi} : represents the amount of deviation in the load from demand f that is on itinerary i

7.7.9 Linear Program

Objective Function

$$\text{minimize} \ \sum_i \zeta_i^{e_s} y_i^{e_s} + \sum_{i,l,t} \Gamma_l^f F_{ilt} y_i^{e_s} + P^{\text{replan}} \sum_i \eta_i + \sum_{f,i} P_f^{\text{reload}} \beta_{fi}$$

Aircraft Constraints

$$\sum_{i,k} \phi_{ijklt} y_i^{e_s} - \sum_{i,k} \psi_{ijkl(t-R'_j)} y_i^{e_s} + a^u_{jlt} - a^u_{jl(t-1)} = N_{jlt} \quad \forall j,l,t$$

Crew Constraints

$$\sum_{i,j} \eta_{ijklt} y_i^{e_s} - \sum_{i,j} \gamma_{ijkl(t-R_k)} y_i^{e_s} + c_{klt} - c_{kl(t-1)} = C_{klt} \quad \forall k,l,t$$

Fuel Constraints

$$\sum_i F_{ilt} y_i^{e_s} + f_{lt} - f_{l(t-1)} + \sum_{i,h \in rhop(i)} F'_{ih} rdep(l,i,h,t) v_{ihl} = G_{lt} \quad \forall l,t$$

Refueling constraints

$$\sum_{l \in rloc(i,h)} v_{ihl} - \nu_{ih} y_i^{e_s} = 0 \quad \forall i, h \in rhop(i)$$

$$\sum_{i,h \in rhop(i)} rdep(l,i,h,t) v_{ihl} - \sum_{i,h \in rhop(i)} rarr(l,i,h,t) v_{ihl} + t_{lt} - t_{l(t-1)} = N_{jlt} \quad \forall l,t,j = \text{tanker-type}$$

Maximum on Ground

$$\sum_i gs(i) \theta_{ilt} y_i^{e_s} - \sum_i gs(i) \omega_{ilt} y_i^{e_s} + m_{lt} - m_{l(t-1)} = M_{lt} \quad \forall l,t$$

$$\sum_{i,j} gs(i) J_j^L J_{ij} \theta_{ilt} y_i^{e_s} - \sum_{i,j} gs(i) J_j^L J_{ij} \omega_{ilt} y_i^{e_s} + m'_{lt} - m'_{l(t-1)} = M^L_{lt} \quad \forall l \in \mathcal{SB}, t$$

Demand Constraints (for Event LPs)

$$\sum_{i \in \ \text{itin}(i',f,e)} u^e_{fii'} + \mu^e_{fi'} = Z^{e-1}_{fi'} \quad \forall i' \in J^{e-1}, f$$

Capacity Constraints (weight and area)

$$\sum_f a_{fih} u^{e_s}_{fi} - K^m_{ih} y_i^{e_s} \leq 0 \quad \forall i, h \in legs(i)$$

$$\sum_f a_{fih} \chi_{fi} u^{e_s}_{fi} - K^a_{ih} y_i^{e_s} \leq 0 \quad \forall i, h \in legs(i)$$

Capacity Constraints (passengers)

$$\sum_f a_{fih} \rho_{fi} u^{e_s}_{fi} - K^p_{ih} y_i^{e_s} \leq 0 \quad \forall i, h \in legs(i)$$

Transshipment Constraints

$$\sum_{i \in civa(l,t)} u_{fi}^{e_s} + v_{fl(t-1)} - \sum_{i \in mild(l,t)} a_{fihop(f,i)} u_{fi}^{e_s} - v_{flt} = 0 \qquad \forall l \in \mathcal{TS}, t, f(\text{s.t. dest(f)} \neq l, \text{orig(f)}$$

Recourse Itineraries Flow Constraint

$$\sum_{i \in \text{ reci}(i', e_s)} y_i^{e_s} = X_{i'}^{e_s - 1} \qquad \forall i' \in I^{e_s}$$

$$y_{i'}^{e_s} = X_{i'}^{e_s - 1} \qquad \forall i' \in I'^{e_s}$$

Deviation Constraints

$$\eta_i \geq y_i^{e_s} - y_i^{e_s - 1} \quad \forall i \in \mathcal{I}$$

$$\eta_i \geq y_i^{e_s - 1} - y_i^{e_s} \quad \forall i \in \mathcal{I}$$

$$\beta_{fi} \geq u_{fi}^{e_s} - u_{fi}^{e_s - 1} \quad \forall i \in \mathcal{I},\ f$$

$$\beta_{fi} \geq u_{fi}^{e_s} - u_{fi}^{e_s - 1} \quad \forall i \in \mathcal{I},\ f$$

7.8 Implementation Details

7.8.1 Recourse Generation

In this section, we describe some of the rules that were used for generating recourses for the itineraries that are affected either directly or indirectly by the disruption. The directly affected itineraries are identified as:

- For onground delay event, the affected itinerary is directly specified in the event description.
- For base disruption event, all the itineraries that are either landing and/or taking-off from the affected base during the period that the base is disrupted are tagged as the directly affected itineraries.

Once the directly affected itineraries (or the 1^{st} order affected itineraries) have been identified, the indirectly affected itineraries ($2^{nd}, 3^{rd}$ order, and so on) are identified. $N^{th}(N \geq 2)$ order affected itineraries are identified as follows. Let $\mathcal{L}$ be the set of locations visited by the itineraries in the P^{th} order affected itinerary set, where $P < N$, during or after the disruption event. Then, if an itinerary has not already been identified as an affected itinerary

and visits any of the locations in $\mathcal{L}$ during or after the t_{aware} time, that itinerary is tagged as a N^{th} order affected itinerary.

We use the following rules to generate multiple recourses for the directly and indirectly affected itineraries:

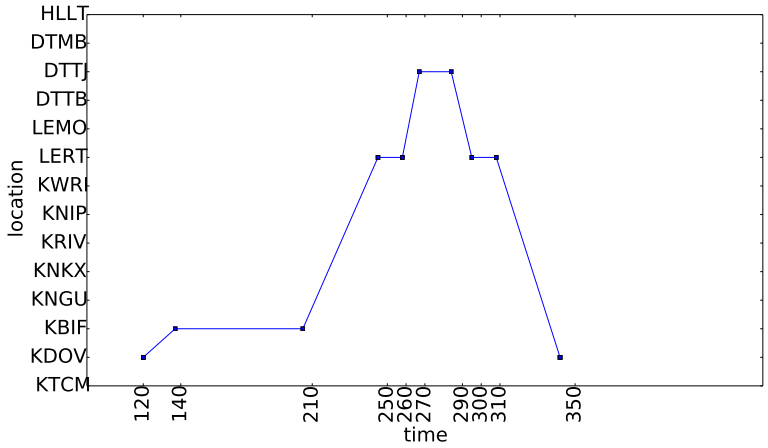
- The affected itinerary is delayed on the ground at least until the event is over. The recourse itineraries take-off every t_{delay} units once the event is over, up to t_{max} maximum delay.
- The affected itinerary is redirected to a nearby location. Once the disruption event at the affected base is over, or the repair for the aircraft has been done at this alternate location, the itinerary flies back to the original location.
- This recourse is similar to the previous recourse except that instead of returning to the original location in the itinerary, the recourse itinerary moves on to the next stop in the original itinerary.
- In case of a base disruption event when the itinerary has not yet reached the affected base, take-off of the itinerary is delayed at one of the large bases (that is having large MOG) that precedes the affected base in the original itinerary.

Figure 7.3 shows example of some sample recourse itineraries for an itinerary affected by a weather event.

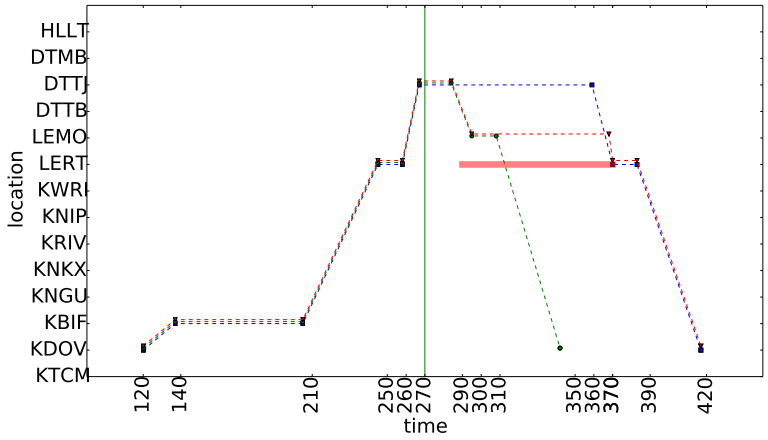
Description of techniques to generate good recourse itineraries that ensure LP feasibility and have shortest paths is beyond the scope of this work and is left for future work.

7.9 Experimental Setup

In this section, we discuss the various details of our experimental setup to evaluate the proposed stochastic integer programming approach for the DMR problem. We first discuss the source of the input data (Section 7.9.1) used in our experiments. This is followed by some alternative approaches for DMR optimization in Section 7.9.2. We compare the proposed



(a) Original Itinerary



(b) Recourse Itineraries

Figure 7.3: (a) An itinerary with stops on y-axis and time on x-axis. This itinerary is directly affected by a weather event that becomes known at time 270 (shown by vertical green line at time 270) and makes location LERT unavailable for landing and take-off from time 290 to time 370 (shown by horizontal thick red line). In (b), three recourse itineraries are shown for the affected itinerary. In the first recourse itinerary (shown in blue color), the take-off of original itinerary is delayed at the current location DTTJ such that it reaches LERT by the time the weather event is over. In the second recourse itinerary (shown in green color), the itinerary is diverted to an alternate location LEMO and from there the itinerary carries on to its next destination KDOV. On the other hand, in the third recourse itinerary (shown in red color), the itinerary first goes back to LERT, which was on its original schedule, once the weather is clear and then continues to its next destination from there.

stochastic optimization approach with these alternative approaches in Section 7.10 using the simulation setup given in Section 7.9.3 and performance metrics given in Section 7.9.4.

7.9.1 Model Data

The data for mission requirements comes from the strategic airlift portion of the AT21D TPFDD. We anticipate expanding to other sources in the future. Following the advice to Chairman Joint Chiefs of Staff (CJCS Guide) 3122, Unit Line Number (ULNs) with less than 15 tons or 100 passengers are combined into larger ULNs with nearby Aerial Port Of Debarkation (APODs), Aerial Port Of Embarkation (APOEs), Available to Load Date (ALDs), and Latest Arrival Date (LADs) (in order of priority). Load variations are not available for the Automated Transportation for the 21st Century (AT21) TPFDD, so we derived representative variations based on historical data from September 2006.

Mission delay data is taken from the 2010 GDSS archive. The delay codes (by aircraft type) were split into four categories:

- command and control (C2),
- airfield and weather,
- cargo, and
- aircraft and crew.

Aircraft and crew delays vary significantly by aircraft type. C5s are almost 3 times as likely to incur delays as C17s (probability .177, .179, and .061 for C5B, C5A, and C17, respectively in the current model), and those delays average 2.5 to 3 times longer (30.3, 28.7, 11.8 hours for C5B, C5A, and C17, respectively in the current model). This data is used to generate future scenarios that must be considered when planning/replanning missions. We generate future scenarios by sampling from each of these four categories as appropriate. For example, cargo delays are not normally associated with enroute bases, and weather delays vary markedly by airfield. The architecture of the stochastic optimization allows for easy refinement of this data as more and better data becomes available.

7.9.2 Alternative DMR Optimization Approaches

There are other approaches that can be used to solve the dynamic mission replanning problem. We discuss two such approaches here, and compare them with the proposed stochastic optimization approach in the results section (Section 7.10).

Myopic optimization

Because of the large scale and high complexity of the problem, the TACC duty officer often adjusts the CuIS to come up with an execution schedule that is feasible to the current disruption. Any disruptions in the future are handled as and when they become known. Myopic Optimization (MyOp) approach resembles the current approach taken by TACC except in the way the recourse itineraries are choosen. Currently, TACC officers use some visualization aids to determine and select recourse itineraries, while in MyOp we have a global optimization routine, that is the Stage 1 integer program, to select optimal recourse itineraries. In other words, in MyOp there are no Stage 2 evaluations/feedback and only Stage 1 optimization is done to determine the new execution schedule. This approach is same as the naïve approach described in Section 7.6.1.

Deterministic optimization

This approach falls in between the MyOp and the Stochastic Optimization (StOp) approach. In this approach, the Stage 1 decisions are made while assuming a nominal future scenario. A nominal scenario consists of expected future disruptions and does not take into account the uncertainty in the future disruption events. Deterministic Optimization (DetOp) is same as StOp with just one scenario.

7.9.3 Simulation Setup

We evaluate the StOp approach against MyOp and DetOp by evaluating these approaches for a weather disruption event that disables any landings and take-offs from a base. Sixteen future disruption scenarios were generated, where each scenario has two weather events

that affects two locations for varying periods of time. Different probabilities are assigned to each scenario, such that they add up to one. Our evaluation setup simulates a real-life (although at a small scale) execution period during which TACC has to handle multiple disruption events, a total of three in this case. We simulate all sixteen scenarios event by event, with the execution solution from one event passed on as input to the next event. At each disruption event, the new execution schedule is obtained by solving the optimization problem of the corresponding optimization approach. The simulation setup is picturized in Figure 7.4. For ease of demonstration, we have designed a simple setup in which all the scenarios have the same number of events but this is not a requirement for our setup. We now explain the optimization problems solved at each disruption event in each of the three approaches:

- MyOp: At each disruption event only the Stage 1 integer program is optimized to obtain the new execution schedule.
- DetOp: A nominal scenario is generated by taking the probability weighted average of all possible durations of the remaining weather events in the scenarios. At event 0, Stage 2 nominal scenario is generated by taking the respective probability weighted average of the durations of weather events, event 1 and event 2, across all scenarios. Similarly at event 1, Stage 2 nominal scenario is generated by taking probability weighted average of possible durations of weather event, event 2. Finally, at event 2, there is no Stage 2 scenario, as this is the last event in all the scenarios.
- StOp: At event 0, a stochastic program with 16 scenarios in Stage 2 is optimized to obtain the new execution schedule. At event 1, a stochastoc program with 4 scenarios in Stage 2 is solved. Finally, at event 2 only Stage 1 optimization is done as there are no more events remaining in any scenario.

7.9.4 Key Performance Indicators (KPIs)

The following performance indicators are calculated to measure and compared the performance of each of the optimization approaches. These metrics are obtained by calculating

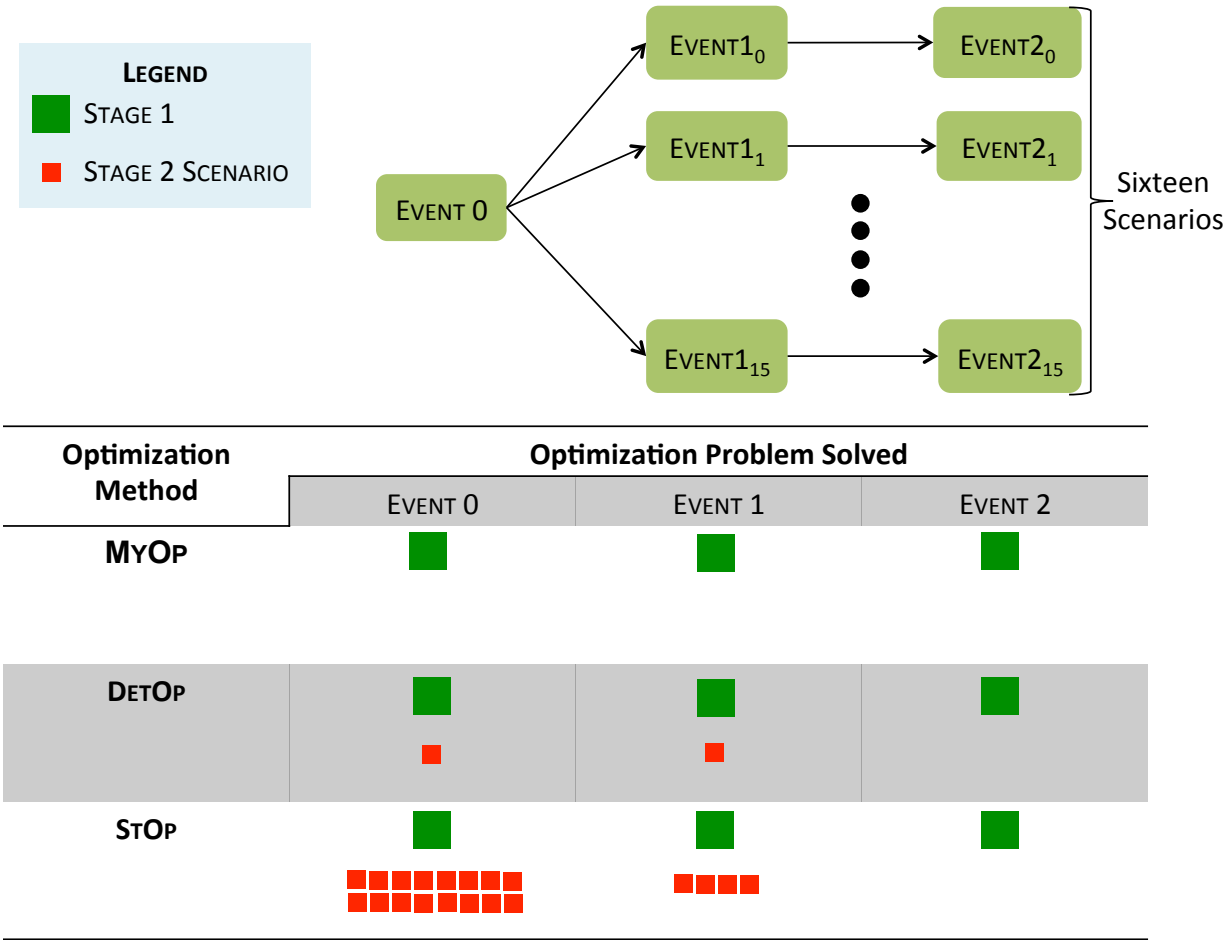


Figure 7.4: Simulation setup. There are sixteen scenarios in total. Event 0 in each scenario is common across all scenarios. EVENT1$_0$ is event 0 of scenario 0, EVENT1$_1$ is event 0 of scenario 1, and so on. All sixteen scenarios are simulated. The diagram also shows the optimization problem solved at each disruption event in each of the optimization methods. Absence of Stage 2 scenario means that only the Stage 1 IP optimization is done to obtain the new execution schedule. Presence of more than one Stage 2 scenario means stochastic optimization is done to obtain the new execution schedule, while only one Stage 2 scenario means that a deterministic problem is solved with a nominal Stage 2 scenario.

their probability weighted average across all the scenarios.

- Average Cost (AvgCost): This corresponds to the total cost due to replanning. It is calcualted as the sum of the objective function values of the Stage 1 programs at each disruption event.

- Average Itinerary Delay (AvgItinDel): This metric gives the sum total of the number of hours by which the itineraries got delayed. The number of hours by which an itinerary is delayed is the difference in the time at which it reaches its final desitination in the final schedule and the time at which it would have reached in the original schedule.

- Demand Penalty (AvgDemPenalty): This is the late and undelivered demand penalty due to replanning.

- Average Notification Time (AvgNotificationTime): This computes how much in advance of the actual change in the itinerary was the itinerary notified of the change. in advance of the actual change in the itinerary was the itinerary notified of the change.....

- Average Number of Changes (AvgNumChanges): This counts the total number of changes done to the itineraries during the course of replanning across all the disruption events.

7.10 Results

In this section, we compare the performance of various optimization methods.

Table 7.12 compares the performance of various KPIs across the three optimization approaches. To compare the three approaches across wide variety of real life situations, a total of six simulations of the setup described in Figure 7.4 were performed. The six simulations are categorized into three types - High Variation (HighVar), Medium Variation (MedVar), Low Variation (LowVar). Each of these categories has two simulations each. Scenarios in

HighVar simulation have high variation in weather event durations across its scenarios, while the scenarios in LowVar simulation have very low variation across weather event durations acorss its scenarios. StOp outperforms MyOp and DetOp in all the cases while giving as much as up to 80% reduction in costs over MyOp. These results establish the superior qualtiy of solutions obtained from StOp. StOp outperforms MyOp in all the cases.

Table 7.12: Comparison of various KPIs across different optimization approaches

Simulation Scenario	KPI	Optimization Approach			StOp vs MyOp (%)
		MyOp	DetOp	StOp	
High Variation	AvgItinDel	190	156	143	24.74
	AvgCost	7344999	12500640	1431317	80.51
	AvgItinDelCost	3423945	689215	779025	77.25
	AvgDemPenalty	3921054	11811425	652292	83.36
Medium Variation	AvgItinDel	200	176	117	41.5
	AvgCost	1706082	12587809	1308470	23.31
	AvgItinDelCost	1060075	793240	636600	39.95
	AvgDemPenalty	646007	11794569	671870	-4.0
Low Variation	AvgItinDel	87	89	62	28.74
	AvgCost	759857	11813295	686096	9.71
	AvgItinDelCost	463237	340120	347267	25.03
	AvgDemPenalty	296620	11473175	338828	-14.23

CHAPTER 8

Conclusion and Future Work

Our research shows that high performance computing can enable optimization of stochastic programs that are otherwise intractable on single/multi-core machines. We identified decomposable structures in two-stage stochastic optimization problems. This led to significant improvements in the convergence rate of benders method for stochastic optimizations. We also identified the interdependencies amongst these decomposed structures that led to highly parallel and scalable designs for stochastic linear/integer program optimizations. Finally, in the second part of the thesis we provide a computational engine for some dynamic and real-time problems faced by US Air Mobility Command. The resulting framework leads to significant cost savings as compared to the currently used approach by the Air Mobility Command. In particular, the contributions of this dissertation are:

- Cut-retirement schemes and scenario clustering for reducing the Stage 1 and Stage 2 computation required in Bender's method for two-stage stochastic optimization.
- A **p**arallel **s**tochastic **i**nteger **p**rogram **s**olver, PSIPS, that exploits branch-and-bound parallelism and nested parallelism to achieve high parallel efficiencies. State-sharing amongst the vertices reduces the solution time.
- Strong scaling of stochastic programs to hundreds of cores.
- A Split-and-Merge (SAM) method for accelerating convergence of stochastic programs with large number of scenarios.

- A Lagrangean Decomposition and Merge (LDAM) method for accelerating convergence of stochastic programs with large complexities.
- Stochastic formulation of the military aircraft allocation problem with consideration of disaster management that gives up to 35% savings in costs as compared to deterministic optimization.
- A stochastic integer programming approach for Dynamic Mission Replanning by the Air Mobility Command. The resulting framework responds to disruptive events at execution time and gives significantly better schedules as compared to other approaches.

Our attempts result in strong scaling to hundreds of cores. We believe similar results are not common in literature, and that our experiences will feed usefully into further research on this topic. We believe that this work will provide the springboard for more robust problem solving with HPC in many logistics and planning problems.

APPENDIX A

Stochastic Formulation of Military Aircraft Allocation Problem

In this appendix, we give detailed description of the stochastic formulation of a military aircraft allocation problem. The following subsections contain the descriptions of the indices and index sets, the input data to the model, and the variables in the model. This is followed by Stage 1 and Stage 2 linear programs of the two-stage stochastic formulation of the problem.

A.1 Indices and Index Sets

Description of the inidces and index sets are given below.

- $t \in \mathcal{T}$: Time periods. We discretize time into days.
- $s \in \mathcal{S}$: Stage 2 scenarios.
- $m \in \mathcal{M}$: Mission types. Channel, Contingency, and Special assignment airlift missions are indexed with 'ch,' 'co,' and 'sa,' respectively. Each mission type is subject to uncertainty, realized either by cargo demand (channel, contingency) or aircraft needed (Special assignment airlift missions).

Cargo

- $i \in \mathcal{I}$: Cargo demand identifiers. This index is overloaded to accommodate the precise definitions of the different mission types. For channel missions, i represents two-way **od!** (**od!**) pairs with daily cargo delivery demands. For contingency missions, i represents a specific cargo delivery demand between **od!** pairs at a specific time. For special assignment airlift missions, i serves as an index of aircraft charter demands.

- $k \in \mathcal{K}$: Cargo types. We generalize cargo into four types. Bulk cargo consists of small items consolidated into aircraft pallets that fit on all cargo aircraft. Oversize cargo consists of items such as rolling stock that fit on some civilian and most military aircraft. Outsize cargo consists of items such as tanks or helicopters, which fit only on wide-body military aircraft. Passengers may be carried on all aircraft equipped with seating. This set is overloaded with and additional index 'sam' that denotes a Special assignment airlift mission demand, which is independent of cargo type. Unless explicitly indicated, 'sam' is excluded from summation and domain expressions.

- $K_j \subset \mathcal{K}$: Subset of cargo types that can be carried by aircraft type j.

- $\mathcal{TS} \subset \mathcal{I}$: The subset of $\mathcal{I}$ that requires transshipment, which can occur when civilian aircraft cannot be flown into regions of conflict.

Aircraft

- $j \in \mathcal{J}$: Aircraft (jet) types, civilian and military. Aircraft types differ in their infrastructure requirements, capacity, operating cost, etc.

- $J_{mil}, J_{civ} \subset \mathcal{J}$: Subsets of military and civilian aircraft, respectively.

- $J_1, J_2 \subset \mathcal{J}$: Subsets of aircraft that are allocated in Stage 1 and Stage 2, respectively. Note that only civilian aircraft (short-notice rentals) are elements of J_2.

- $JT \subset \mathcal{J}$: Subset of aircraft that can serve as tankers or airlifters.

- $JS_i \subset \mathcal{J}$: Subset of aircraft requested by special assignment airlift mission i.

Locations

- $l \in \mathcal{L}$: Locations. These may be aircraft home bases, cargo origin or destination bases, enroute bases, or aerial locations used for inflight refueling.
- $LA \subset \mathcal{L}$: Subset of locations that are air refueling locations.

Routes

- $r \in \mathcal{R}$: Routes. Each cargo route begins at an aircraft home base and transits a cargo origin and destination or transshipment location (both inbound and outbound). It may also transit one or more enroute locations for refueling. For example, a route may be of the type: home-base — origin — enroute — destination — origin — home-base, or home-base — origin — air-refueling — destination — home-base. In some cases the home base and origin are co-located. Air refueling routes begin at an aircraft home base and visit an air refueling location to deliver fuel to another aircraft.
- $S1_i \subset \mathcal{R}$: Subset of routes which constitute the first portion of a transshipped delivery for demand i. These routes are flown by civilian aircraft.
- $S2_i \subset \mathcal{R}$: Subset of routes which constitute the second portion of a transshipped delivery for demand i. These routes are flown by military aircraft.
- $Q1_l \subset \mathcal{R}$: Subset of airlift routes transiting air refueling location l.
- $Q2_l \subset \mathcal{R}$: Subset of tanker routes servicing air refueling location l.
- $S_i \subset \mathcal{R}$: Subset of routes serving demand i.
- $O_r \subset \mathcal{L}$: Home base location l (origin) of route r (1 element per subset).

A.2 Input Data

This section contains the description of the input data to the model.

- $A_{j,t}$: Number of hours an aircraft j is available for flying in period t.

- $C_{r,j}$: Capacity of aircraft j when flying route r. $C_{r,j}$ will depend on the distance of each leg, fuel requirements and other factors.

- $C^k_{r,j}$: Capacity of aircraft j for carrying cargo type k when flying route r. This accommodates different space requirements of bulk cargo, oversize cargo, outsize cargo, and passengers.

- $\tilde{C}_{r,j}$: Surplus capacity on aircraft j from a special assignment airlift mission to carry channel cargo.

- $D_{m,i,k,t}(s)$: Demand of cargo type k for requirement i of mission m in period t as realized in scenario s. The demand is modeled as a random variable. The units for the demand are tons for channel and contingency missions, and aircraft for special assignment airlift missions.

- $E_{r,j}$: Operational expenses incurred for flying aircraft j on route r.

- $H_{i,k,t}$: Maximum unmet channel cargo demand for requirement i, cargo type k at time t. Unmet demand greater than H is penalized at a higher rate.

- $Opt(\mathbf{y}, \mathbf{s})$: Optimal Stage 2 objective function value for scenario s, given allocation vector $\mathbf{y}$; $\mathbf{y}^*$ denotes an incumbent solution.

- $P^1_{k,m}$ ($P^2_{k,m}$): Penalty per unit weight for late (very late) delivery of a cargo of type k for mission m.

- R_j: Per period cost of civilian aircraft j if leased well in advance. This is used in Stage 1.

- $\hat{R}_j$: Per period rental cost of civilian aircraft j if rented on short notice. This is used in Stage 2.

- RDD_i: The required delivery date for contingency requirement i.

- $T_{m,r,j,t,t'}(s)$: Hours required in period t to complete route r with aircraft j when launched in period t' for mission m as realized in scenario s.

- $T^{'}_{m,r,j}$: Flying hours of aircraft j to complete route r while flying mission type m.
- $TR_{r,j,l}$: Tankers required (baselined by KC10 equivalents, which is a large tanker) by aircraft j flying route r at air refueling location l
- μ_j: Permissible flying time utilization of aircraft of type j.
- $\Delta_{i,r,j}$: Time periods needed to reach the destination or transshipment base for requirement i on route r using aircraft type j (channel and contingency). For special assignment airlift missions, it denotes the time periods required to reach the initial special assignment airlift mission location.
- $\Delta_{r,j}$: time periods needed to complete route r using aircraft type j. As used in the air refueling constraint, it denotes the number of lag periods between an airlift mission launch on route r by aircraft j, and the air refueling event.
- $Y_{j,l}$: Number of aircraft of type j available for allocation at location l.
- π_s: Probability of scenario s.

A.3 Variables

This section contains the description of variables in Stage 1, and Stage 2 linear programs. All variables are continuous, non-negative, and used in Stage 2 unless otherwise noted.

- $u^1_{i,k,m,t}, u^2_{i,k,m,t}$: Unmet demand of cargo type k for requirement i of mission m in period t. The penalty for unmet demand grows linearly when cargo has not been delivered for $\tau_{i,m}$ days. Beyond that the penalty is increased. To model this for channel missions, we divide the unmet demand into two parts. $u^1_{i,k,ch,t}$ is the unmet demand that is less than $\tau_{i,ch}$ days old and $u^2_{i,k,ch,t}$ is the unmet demand that is more than $\tau_{i,ch}$ days old. We define the threshold H as follows:

$$H_{i,k,t} = \sum_{\Gamma=t-\tau_{i,ch}}^{t} D_{ch,i,k,\Gamma}(s)$$

Late cargo for contingency missions is defined using the parameter RDD_i. Cargo delivered on or before the RDD_i is unpenalized. Cargo delivered one to $\tau_{i,co}$ days late is penalized at rate $P^1_{k,co}$. Thereafter, cargo is penalized at rate $P^2_{k,m}$. Late special assignment airlift missions are disallowed: they are either flown on the requested day or a penalty is imposed.

- $x_{m,r,j,t}$: Number of type j aircraft launched on route r in time t supporting mission m.

- $y_{j,l,m}$ (general integer, Stage 1 variable): Number of Stage 1 aircraft j allocated to (base) location l for mission m. This is the principal output of the program. Stage 1 allocations include all military aircraft and civilian aircraft on advanced (more than one month prior) lease. We use $\mathbf{y}$ to denote the allocation vector, and $\mathbf{y}^*$ for an allocation vector obtained from Stage 1 optimization.

- $\hat{y}_{j,l,m,t}$: Number of stage 2 rented aircraft (short-notice, high-cost) j allocated at (base) location l for mission m at time t. This is also a principal output of the program, and is approximated with a linear variable.

- $z_{m,i,j,k,r,t}$: Tons of of type k cargo for requirement i, mission m transported on aircraft j using route r in period t.

- θ_s (Stage 1 variable): Stage 2 cut support for scenario s.

- $v^s_{\cdot}$: Optimal dual variables for scenario 's' obtained by solving Stage 2 of the model. The (.) subscript denotes the constraint block number and variable domain.

A.4 Stage 1 Linear Program

Objective Function

$$\sum_{j \in J_{Civ} \cap J_1, l, m} R_j y_{j,l,m} + \sum_s \pi_s \theta_s$$

Feasible Allocation

$$\sum_{m} y_{j,l,m} \leq Y_{j,l} \qquad \forall j \in J_1, l$$

Stage 2 Cuts

$$\theta_s \geq Opt(\mathbf{y}^*, \mathbf{s}) + \sum_{j \in J_1, l, m} (\sum_{t} A_{j,t} v^s_{7,j,l,m,t} + \mu_j v^s_{9,j,l,m})(y_{j,l,m} - y^*_{j,l,m}) \quad \forall s$$

Variable range

$$y_{j,l,m} \in \{0, 1, 2, ...\} \qquad \forall j, l, m$$

A.5 Stage 2 Linear Program

Objective Function

$$Opt(\mathbf{y}, \mathbf{s}) = \min \sum_{j \in J_{civ}, l, m, t} \hat{R}_j \hat{y}_{j,l,m,t} + \sum_{i,k,t} P^1_{k,ch} u^1_{i,k,ch,t}$$
$$+ \sum_{i,k,t=RDD_i}^{RDD_i + \tau_{i,co}} P^1_{k,co} u^1_{i,co,t} + \sum_{i,k,m,t} P^2_{k,m} u^2_{i,k,m,t}$$
$$+ \sum_{r,j \in J_{mil}, m, t} E_{r,j} x_{m,r,j,t}$$

Channel Missions, Contingency Missions ($m \in \{ch, co\}$)

Demand (1)

$$-(u^1_{i,k,m,t-1} + u^2_{i,k,m,t-1}) + \sum_{r \in S_i} \sum_{j} z^{ch}_{m,i,j,k,r,t}$$
$$+u^1_{i,k,m,t} + u^2_{i,k,m,t} = D_{m,i,k,t}(s) \qquad \forall i \notin \mathcal{TS}, k, t$$

TS Demand (2)

$$-(u^1_{i,k,m,t-1} + u^2_{i,k,m,t-1}) + \sum_{r \in S_i} \sum_{j \in J_{mil}} z_{m,i,j,k,r,t}$$
$$+ \sum_{r \in S1_i} \sum_{j \in J_{civ}} z_{m,i,j,k,r,t} + u^1_{i,k,m,t} + u^2_{i,k,m,t} = D_{m,i,k,t}(s) \qquad \forall i \in \mathcal{TS}, k, t$$

Transshipment (3)

$$\sum_{r \in S1_i} \sum_{j \in J_{civ}} z_{m,i,j,k,r,(t-\Delta^{i,j}_r)} - \sum_{r \in S2_i} \sum_{j \in J_{mil}} z_{m,i,j,k,r,t} = 0 \qquad \forall i \in \mathcal{TS}, k, t$$

Aggregate capacity, Channel Missions (4a)

$$\sum_{k \in K_j} z_{ch,i,j,k,r,(t+\Delta^{i,j}_r)} - C_{r,j} x_{ch,r,j,t} - \tilde{C}_{r,j} x_{sa,r,j,t} \leq 0 \qquad \forall i, j, r, t$$

Aggregate capacity, Contingency Missions (4b)

$\sum_{k \in K_j} \sum_{i: r \in S_i} z_{co,i,j,k,r,(t+\Delta_r^{i,j})} - C_{r,j} x_{co,r,j,t}$	≤ 0	$\forall j, r, t$
Specific capacity (5)		
$\sum_{i: r \in S_i} z_{m,i,j,k,r,(t+\Delta_{i,r,j})} - C_{r,j,k} x_{m,r,j,t}$	≤ 0	$\forall j, k, r, t$
Price Break (6)		
$0 \leq u^1_{i,k,1,t-1}$	$\leq H_{i,k,t}$	$\forall i, k, t$

Special Assignment Airlift Missions

Demand (1b)		
$\sum_{r \in S_i} x^{sa}_{r,j,(t-\Delta_r^{i,j})} + u^2_{i,sam,sa,t}$	$= D^{sa}_{i,j,t}(s)$	$\forall i, j \in JS_i, t$

Aircraft usage

Mission times, (7, 8)		
$\sum_{t'} \sum_{r: l \in O_r} T_{m,j,r,t,t'}(s) x_{m,r,j,t'}$	$\leq A_{j,t} y^*_{j,l,m}$	$\forall j \in J_1, l, m, t$
$\sum_{t'} \sum_{r: l \in O_r} T_{m,j,r,t,t'}(s) x_{m,r,j,t'} - A_{j,t} \hat{y}_{j,l,m,t}$	≤ 0	$\forall j \in J_2, l, m, t$
Flying times (9):		
$\sum_{t} \sum_{r: l \in O_r} T^{'}_{m,r,j} x_{m,r,j,t}$	$\leq \sum_t \mu_j y^*_{j,l,m}$	$\forall j \in J_1, l, m$
Air Refueling (10)		
$\sum_{r \in Q1_l} \sum_{j} TR_{r,j,l} x_{m,r,j,(t-\Delta_r^j)} =$	$\sum_{r \in Q2_l} \sum_{j \in JT} x_{m,r,j,t}$	$\forall l \in LA, t, m$
Variables Range		
$u^1_{i,k,m,t}$, $u^2_{i,k,m,t}$	≥ 0	$\forall i, k, m, t$
$x_{m,r,j,t}$	≥ 0	$\forall r, j, m, t$
$z_{m,i,j,k,r,t}$	≥ 0	$\forall i, j, k, m, r, t$
$\hat{y}_{j,l,m,t}$	≥ 0	$\forall j, l, m, t$

APPENDIX B

A small example of Dynamic Mission Replanning (DMR)

B.1 Demands and chosen itineraries

Below is a sample set of demands.

Table B.1: A sameple set of demands to be carried by an execution schedule

orig	dest	alt	eat	lat	type	area	mass	efficiency	undel-penalty	late-penalty
1	7	0	0	192	bulk	1500	51.5	1	32238	8060
1	7	0	0	192	over	1800	45.8	1	32238	8060
1	7	0	0	192	pax	0	13.2		2437	609
1	7	0	96	480	bulk	40	0.8	0.9	32238	8060
1	7	0	96	480	over	2100	51.8	0.8	32238	8060
3	8	0	0	384	pax	0	68.8		2564	641
3	8	0	0	384	bulk	3200	95.5	0.9	25030	6258
3	8	0	0	384	over	500	12	1	25030	6258
3	8	0	0	384	out	1750	54.5	1	25030	6258

where, orig is the port of embarkation, dest is port of disembarkation, ald is the available

to load time, eat is the earliest arrival time at the port of disembarkation, lat is the latest arrival time, type stands for the type of cargo (bulk, oversized, passenger, outsized), area is the floor area requirements of the cargo, mass is the mass of the cargo in tons, efficiency is the ratio of the area occupied by the cargo in the aircraft to the specified area requirements of the cargo, undel-penalty stands for the undelivery penalty (in USD) per ton of the demand, and late-penalty stands for the late delivery penalty (in USD) per day per ton of the demand. For these demands, one of the itineraries chosen to be flown is given below (also depicted in Figure B.1). We call this itinerary as $\mathcal{I}$. We shall use this itinerary to describe how it is affected by a scenario and what recourse decisions it may prompt.

```
"atype" :  0,
"stops" :
[ 2, 0, 0, 105207.67, 70.0, 0 ],
[ 1, 18, 86, 226086.70, 70.0, 4.57 ],
[ 5, 126, 139, 103526.25, 70.0, 9.83 ],
[ 7, 147, 164, 0, 70.0, 2 ],
[ 5, 174, 187, 197929.05, 70.0, 2.5 ],
[ 2, 221, 221, 0, 70.0, 8.61 ]
```

Description of each stop in an itinerary consists of - stop/base number, landing time at that stop, take-off time from that stop, fuel requirements at that stop, max payload on the next leg, flying time to this stop from the previous stop, in that order. We adopt a shorthand notation for an itinerary which only refers to the sequence of stops: $2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$. In this notation, the symbol $\longrightarrow$ simply means a 'normal continuation' from one location to the next. This 'normal continuation' includes such things as fuel intake, length of flight-time, length of time on ground, etc. When needed, we augment this shorthand notation with other self-explanatory features such as a 'start time' or various 'ground delays'.

B.2 A sample scenario

We now describe a scenario i.e., a specific set of disruptive events.

- Weather events:
  1. $t_{\text{aware}} = 0$, $t_{\text{start}} = 100$, $t_{\text{end}} = 170$, $l = 5$.
  2. $t_{\text{aware}} = 300$, $t_{\text{start}} = 350$, $t_{\text{end}} = 500$, $l = 5$.
- Breakdown events:
  1. itinerary = 7, leg = 3, $t_{\text{repairs}} = 200$.
- Demand events:
  1. A 20% increase in demand 4.
  2. A 15% decrease in demand 6.

B.2.1 Recourses for a specific itinerary

In the discussion below, we outline the various recourse itineraries for $\mathcal{I}$ that we consider as the scenario unfolds.

1. At time 0, we become aware of Weather-1 i.e., the fact that base 5 is unavailable from time 100 to time 170. Therefore we consider the following recourse actions:
   (a) Delay the start of the itinerary from time 0 to time $170-126 = 44$. In other words, change the itinerary from $2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$ to Delay-44 $\longrightarrow 2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$.
   (b) Divert the itinerary to base 6, which is expected to be unaffected by the weather (per information available at time 0), then continue on to the next scheduled stop. In other words, alter the sequence of stops from $2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$ to $2 \longrightarrow 1 \longrightarrow 6 \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$

(c) Divert the itinerary to base 6, then return to base 5, after the weather has cleared at time 170. In effect alter the sequence of stops from $2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$ to $2 \longrightarrow 1 \longrightarrow 6 \longrightarrow 5 \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$. Note that in this case an additional ground delay is introduced at base 6.

2. Subsequently, the scenario has Breakdown-1 on the third leg of the original itinerary (or whatever recourse is chosen) and another weather-related disruption, Weather-2. Depending on the recourse actions chosen before, the order in which these disruptions occur can change. In our example, the numbers are such that Breakdown-1 occurs **before** we become aware of Weather-2 which is imminent in the scenario. We therefore list our sets of recourse choices under Breakdown-1 for each of the recourse decisions above:

   (a) We have the situation Delay-44 $\longrightarrow 2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 7 \longrightarrow$ Breakdown-1 .... The recourses from this are:

      i. Delay-44 $\longrightarrow 2 \longrightarrow 1 \longrightarrow 5 \longrightarrow 7 \longrightarrow$ dmr:Breakdown-1/Delay-$(200 + x) \longrightarrow 5 \longrightarrow 2$, for some $x \geq 0$. Note that in these recourses, the only choice is in the value of x.

   (b) We have the situation $2 \longrightarrow 1 \longrightarrow 6 \longrightarrow 7 \longrightarrow$ Breakdown-1 .... The recourses from this are analogous to the above, except that take-off and landing times are shifted accordingly (due to the earlier redirection).

   (c) We have the situation $2 \longrightarrow 1 \longrightarrow 6 \longrightarrow 5 \longrightarrow$ Breakdown-1 .... The recourses from this are:

      i. $2 \longrightarrow 1 \longrightarrow 6 \longrightarrow 5 \longrightarrow$ Breakdown-1/Delay-$(200 + x) \longrightarrow 7 \longrightarrow 5 \longrightarrow 2$.

3. Finally, the second weather event (outage of base-5 from time 350 to time 500), which we become aware of at time 300, affects each of our recourse choices. In effect, as a scenario unfolds, we generate choices for recourse decisions. However, the available choices, naturally, depend on our previous decisions.

All the recourses above are also depicted in Figure B.2.

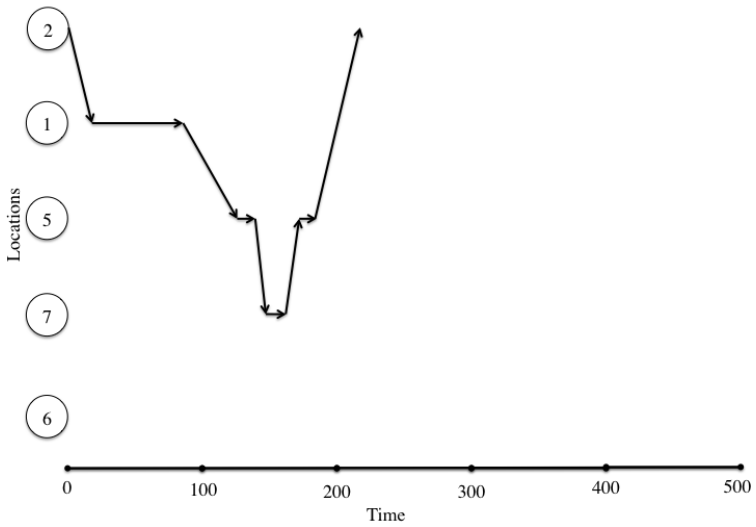
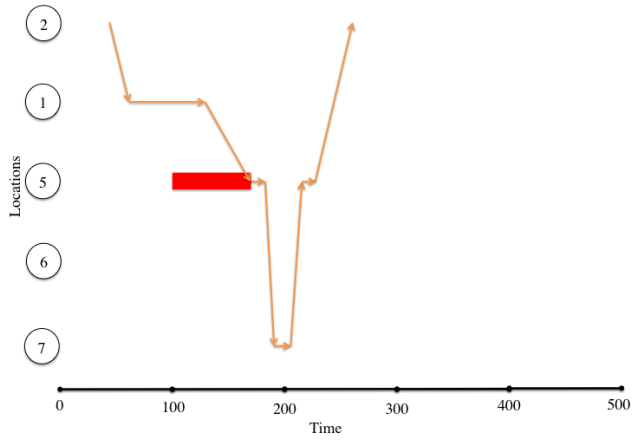
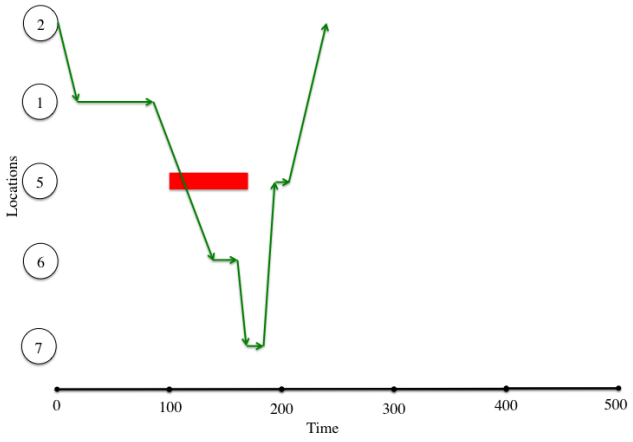


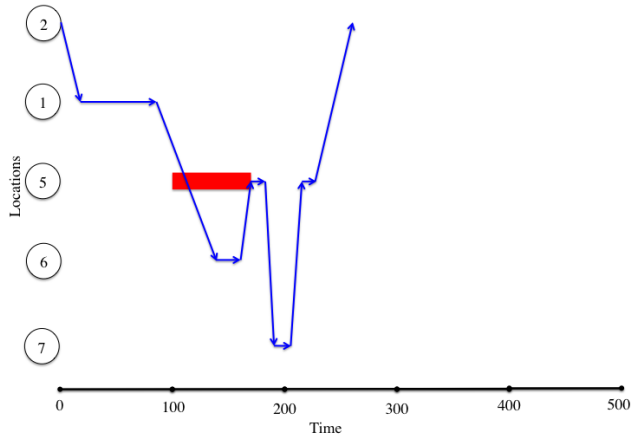
Figure B.1: A sample itinerary $\mathcal{I}$



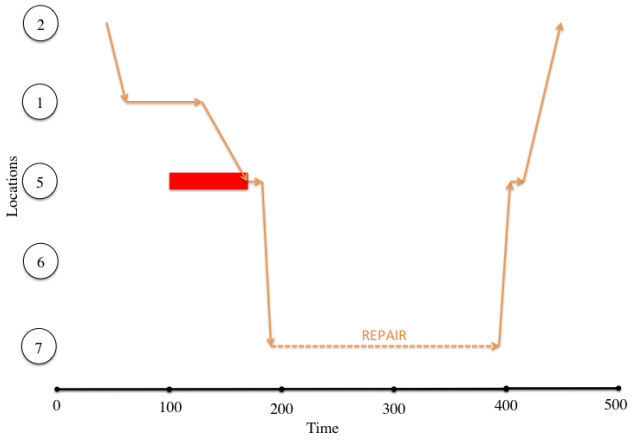
(a) Recourse action 1a of itinerary 7 due to Weather-1



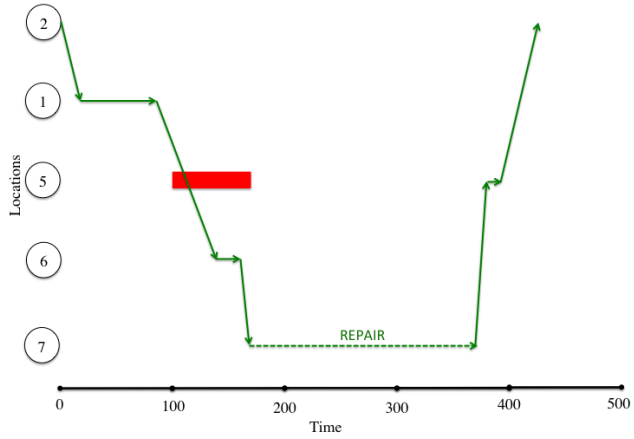
(b) Recourse action 1b of itinerary 7 due to Weather-1



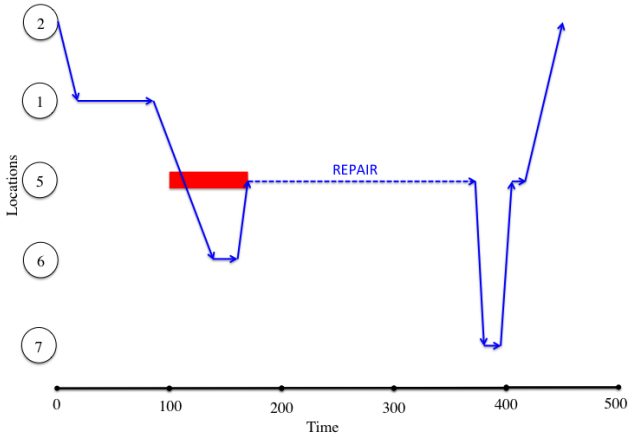
(c) Recourse action 1c of itinerary 7 due to Weather-1



(d) Recourse action 2a of itinerary 7 due to Breakdown-2



(e) Recourse action 2b of itinerary 7 due to Breakdown-2



(f) Recourse action 2c of itinerary 7 due to Breakdown-2

Figure B.2: Itinerary $\mathcal{I}$'s recourses for the given scenario

REFERENCES

[1] J. Benders, "Partitioning Procedures for Solving Mixed Variables Programming Problems," *Numerische Mathematik 4*, pp. 238–252, 1962.

[2] R. M. Van Slyke and R. Wets, "L-shaped Linear Programs with Applications to Optimal Control and Stochastic Programming," *SIAM Journal on Applied Mathematics*, vol. 17, no. 4, pp. 638–663, 1969.

[3] J. Birge and F. Louveaux, "A Multicut Algorithm for Two-stage Stochastic Linear Programs," *European Journal of Operational Research*, vol. 34, no. 3, pp. 384–392, 1988.

[4] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-passing Interface*. MIT press, 1999, vol. 1.

[5] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, and L. Kale, "Parallel Programming with Migratable Objects: Charm++ in Practice," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. New York, NY, USA: ACM, 2014.

[6] G. B. Dantzig, "Linear programming under uncertainty," *Management science*, vol. 1, no. 3-4, pp. 197–206, 1955.

[7] G. B. Dantzig and A. Madansky, "On the solution of two-stage linear programs under uncertainty," in *Proceedings of the fourth berkeley symposium on mathematical statistics and probability*, vol. 1. University of California Press Berkeley, CA, 1961, pp. 165–176.

[8] J. Birge and H. Tang, "L-shaped Method for Two Stage Problems of Stochastic Convex Programming," *Ann Arbor*, vol. 1001, pp. 48 109–2117, 1993.

[9] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*. Springer Science + Business Media, 2011.

[10] R. Schultz, "Continuity properties of expectation functions in stochastic integer programming," *Mathematics of Operations Research*, vol. 18, no. 3, pp. 578–589, 1993.

[11] R. Schultz, "On structure and stability in stochastic programs with random technology matrix and complete integer recourse," *Mathematical Programming*, vol. 70, no. 1-3, pp. 73–89, 1995.

[12] R. Schultz, L. Stougie, and M. v. d. Vlerk, "Two-stage stochastic integer programming: a survey," *Statistica Neerlandica*, vol. 50, no. 3, pp. 404–416, 1996.

[13] S. Ahmed, "Two-stage stochastic integer programming: A brief introduction," *Wiley Encyclopedia of Operations Research and Management Science*, 2010. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/9780470400531.eorms0092/full

[14] L. Stougie and M. V. D. Vlerk, "Approximation in stochastic integer programming," no. April 2003, pp. 1–43, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.6454&rep=rep1&type=pdf

[15] S. Ahmed, "Stochastic Integer Programming An Algorithmic Perspective," *Citeseer*, 2004. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Stochastic+Integer+Programming+An+Algorithmic+Perspective#0

[16] M. Dempster, M. Fisher, L. Jansen, B. Lageweg, J. Lenstra, and A. Kan, "Analysis of Heuristics for Stochastic Programming: Results for Hierarchical Scheduling Problems," *Mathematics of Operations Research*, vol. 8, no. 4, pp. 525–537, 1983.

[17] W. K. K. Haneveld and M. H. van der Vlerk, *Optimizing electricity distribution using two-stage integer recourse models*. Springer, 2001.

[18] G. Laporte, F. Louveaux, and H. Mercure, "The vehicle routing problem with stochastic travel times," *Transportation science*, vol. 26, no. 3, pp. 161–170, 1992.

[19] H. Gangammanavar, S. Sen, and V. Zavala, "Stochastic optimization of sub-hourly economic dispatch with wind energy," *Power Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–11, 2015.

[20] F. D. Munoz and J.-P. Watson, "A scalable solution framework for stochastic transmission and generation planning problems," *Computational Management Science*, pp. 1–28, 2015.

[21] Y. Tong, L. Zhao, L. Li, and Y. Zhang, "Stochastic programming model for oversaturated intersection signal timing," *Transportation Research Part C: Emerging Technologies*, 2015.

[22] H. Park and R. Baldick, "Stochastic generation capacity expansion planning reducing greenhouse gas emissions," *Power Systems, IEEE Transactions on*, vol. 30, no. 2, pp. 1026–1034, March 2015.

[23] S. Ahmed, A. King, and G. Parija, "A multi-stage stochastic integer programming approach for capacity expansion under uncertainty," *Journal of Global Optimization*, vol. 26, no. 1, pp. 3–24, 2003.

[24] K. Kim and S. Mehrotra, "A Two-Stage Stochastic Integer Programming Approach to Integrated Staffing and Scheduling with Application to Nurse Management," *Optimization-Online.Org*, pp. 1–32, 2015. [Online]. Available: http://www.optimization-online.org/DB_FILE/2014/01/4200.pdf

[25] K. Ariyawansa and A. J. Felt, "On a new collection of stochastic linear programming test problems," *INFORMS Journal on Computing*, vol. 16, no. 3, pp. 291–299, 2004.

[26] S. Ahmed, R. Garcia, N. Kong, L. Ntaimo, G. Parija, F. Qiu, and S. Sen, "Siplib: A stochastic integer programming test problem library," 2015. [Online]. Available: http://www.isye.gatech.edu/~sahmed/siplib

[27] H. Gassmann, S. W. Wallace, and W. T. Ziemba, *Stochastic Programming: Applications in Finance, Energy, Planning and Logistics*. World Scientific, 2013, vol. 4.

[28] G. Guo, G. Hackebeil, S. M. Ryan, J.-P. Watson, and D. L. Woodruff, "Integration of progressive hedging and dual decomposition in stochastic integer programs," *Operations Research Letters*, vol. 43, no. 3, pp. 311–316, 2015.

[29] R. T. Rockafellar and R. J.-B. Wets, "Scenarios and Policy Aggregation in Optimization Under Uncertainty," *Mathematics of operations research*, vol. 16, no. 1, pp. 119–147, 1991.

[30] C. C. CarøE and R. Schultz, "Dual decomposition in stochastic integer programming," *Operations Research Letters*, vol. 24, no. 1, pp. 37–45, 1999.

[31] E. Beier, S. Venkatachalam, L. Corolli, and L. Ntaimo, "Stage-and scenario-wise fenchel decomposition for stochastic mixed 0-1 programs with special structure," *Computers & Operations Research*, 2015.

[32] B. Kawas, A. Koc, M. Laumanns, C. Lee, R. Marinescu, M. Mevissen, N. Taheri, S. van den Heever, and R. Verago, "Unified framework and toolkit for commerce optimization under uncertainty," *IBM Journal of Research and Development*, vol. 58, no. 5/6, pp. 12–1, 2014.

[33] A. B. D. Becker, "Decomposition Methods for Large Scale Stochastic and Robust Optimization Problems," Ph.D. dissertation, Massachusetts Institute of Technology, 2011.

[34] G. B. Dantzig and P. W. Glynn, "Parallel Processors for Planning Under Uncertainty," *Annals of Operations Research*, vol. 22, no. 1, pp. 1–21, 1990.

[35] J. Gondzio and R. Kouwenberg, "High-performance computing for asset-liability management," *Operations Research*, vol. 49, no. 6, pp. 879–891, 2001.

[36] M. Avriel, G. Dantzig, and P. Glynn, "Decomposition and Parallel Processing for Large-Scale Electric Power System Planning Under Uncertainty," 1989.

[37] J. Linderoth and S. Wright, "Decomposition algorithms for stochastic programming on a computational grid," *Computational Optimization and Applications*, vol. 24, no. 2, pp. 207–250, 2003.

[38] J. Linderoth and S. Wright, "Computational Grids for Stochastic Programming," *Applications of stochastic programming*, vol. 5, pp. 61–77, 2005.

[39] S. M. Ryan, R.-B. Wets, D. L. Woodruff, C. Silva-Monroy, and J.-P. Watson, "Toward scalable, parallel progressive hedging for stochastic unit commitment," in *Power and Energy Society General Meeting (PES), 2013 IEEE*. IEEE, 2013, pp. 1–5.

[40] A. Papavasiliou, S. Oren, and B. Rountree, "Applying high performance computing to transmission-constrained stochastic unit commitment for renewable energy integration," *Power Systems, IEEE Transactions on*, vol. 30, no. 3, pp. 1109–1120, May 2015.

[41] "Gurobi Optimization Inc. Software, 2014," http://www.gurobi.com/.

[42] "Parallel Distributed-Memory Simplex for Large-scale Stochastic LP Problems. Preprint ANL/MCS-P2075-0412, Argonne National Laboratory, Argonne, IL. April 2012," www.optimization-online.org/DB_HTML/2012/04/3438.html.

[43] M. Lubin, C. G. Petra, M. Anitescu, and V. Zavala, "Scalable Stochastic Optimization of Complex Energy Systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063470 pp. 64:1–64:64.

[44] "Air Mobility Command. Air Mobility Command Almanac 2009. Retrieved 12 Sep 2011," *http://www.amc.af.mil/shared/media/document/AFD-090609-052.pdf*.

[45] J. Benders, "Partitioning Procedures for Solving Mixed-variables Programming Problems," *Numerische mathematik*, vol. 4, no. 1, pp. 238–252, 1962.

[46] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng, "Programming Petascale Applications with Charm++ and AMPI," in *Petascale Computing: Algorithms and Applications*, D. Bader, Ed. Chapman & Hall / CRC Press, 2008, pp. 421–441.

[47] L. Kale, A. Arya, A. Bhatele, A. Gupta, N. Jain, P. Jetley, J. Lifflander, P. Miller, Y. Sun, R. Venkataraman, L. Wesolowski, and G. Zheng, "Charm++ for productivity and performance: A submission to the 2011 HPC class II challenge," Parallel Programming Laboratory, Tech. Rep. 11-49, November 2011.

[48] C. Kim, "LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies," *IEEE Transactions on Computers*, vol. 50, no. 12, 2001.

[49] J. Hartigan and M. Wong, "Algorithm AS 136: A K-means clustering algorithm," *Applied Statistics*, pp. 100–108, 1979.

[50] F. Louveaux and R. Schultz, "Stochastic Integer Programming," *Handbooks in operations research and mgmt science*, vol. 10, pp. 213–266, 2003.

[51] N. Sahinidis, "Optimization Under Uncertainty: State-of-the-art and Opportunities," *Computers & Chemical Engg*, vol. 28, no. 6, pp. 971–983, 2004.

[52] Y. Xu, T. Ralphs, L. Ladányi, and M. Saltzman, "Computational Experience with a Software Framework for Parallel Integer Programming," *INFORMS Journal on Computing*, vol. 21, no. 3, pp. 383–397, 2009.

[53] T. Koch, T. Ralphs, and Y. Shinano, "Could we use a Million Cores to Solve an Integer Program?" *Mathematical Methods of Operations Research*, pp. 1–27, 2012.

[54] A. Sinha and L. Kale, "A load balancing strategy for prioritized execution of tasks," in *Workshop on Dynamic Object Placement and Load Balancing, in co-operation with ECOOP's 92*, Utrecht, The Netherlands, April 1992.

[55] G. Bitran, E. Haas, and H. Matsuo, "Production Planning of Style Goods with High Setup Costs and Forecast Revisions," *Operations Research*, vol. 34, no. 2, pp. 226–236, 1986.

[56] M. H. van der Vlerk, "Stochastic Integer Programming Bibliography," http://www.eco.rug.nl/mally/biblio/sip.html, 1996-2007.

[57] L. Escudero, M. Araceli Garín, G. Pérez, and A. Unzueta, "Scenario Cluster Decomposition of the Lagrangian Dual in Two-stage Stochastic Mixed 0-1 Optimization," *Computers & Operations Research*, 2012.

[58] "IBM CPLEX Optimization Studio. Software, 2012," http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/.

[59] J. Watson, D. Woodruff, and W. Hart, "PySP: Modeling and Solving Stochastic Programs in Python," *Mathematical Programming Computation*, pp. 1–41, 2011.

[60] "PySp: Python-based Stochastic Programming Modeling and Solving Library, 2012," https://software.sandia.gov/trac/coopr/wiki/PySP.

[61] M. Lubin, K. Martin, C. Petra, and B. Sandıkçı, "On Parallelizing Dual Decomposition in Stochastic Integer Programming," *Operations Research Letters*, 2013.

[62] L. Kalé and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proceedings of OOPSLA'93*, A. Paepcke, Ed. ACM Press, September 1993, pp. 91–108.

[63] L. Kale, A. Arya, N. Jain, A. Langer, J. Lifflander, H. Menon, X. Ni, Y. Sun, E. Totoni, R. Venkataraman, and L. Wesolowski, "Migratable objects + active messages + adaptive runtime = productivity + performance a submission to 2012 HPC class II challenge," Parallel Programming Laboratory, Tech. Rep. 12-47, November 2012.

[64] A. Langer, R. Venkataraman, U. Palekar, L. V. Kale, and S. Baker, "Performance Optimization of a Parallel, Two Stage Stochastic Linear Program: The Military Aircraft Allocation Problem," in *Proceedings of the 18th International Conference on Parallel and Distributed Systems (ICPADS 2012)*, Singapore, December 2012.

[65] T. L. Magnanti and R. T. Wong, "Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria," *Operations Research*, vol. 29, no. 3, pp. 464–484, 1981.

[66] A. Ruszczyński, "A regularized decomposition method for minimizing a sum of polyhedral functions," *Mathematical programming*, vol. 35, no. 3, pp. 309–333, 1986.

[67] X. Li, A. Tomasgard, and P. Barton, "Decomposition Strategy for the Stochastic Pooling Problem," *Journal of Global Optimization*, vol. 54, no. 4, pp. 765–790, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10898-011-9792-0

[68] A. B. D. Becker, "Decomposition Methods for Large scale Stochastic and Robust Optimization Problems," Ph.D. dissertation, Massachusetts Institute of Technology, 2011.

[69] Z. Guan and A. Philpott, "A Multistage Stochastic Programming Model for the New Zealand Dairy Industry," *International Journal of Production Economics*, vol. 134, no. 2, pp. 289 – 299, 2011, robust Supply Chain Management. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925527309004058

[70] M. Held and R. Karp, "The Traveling-salesman Problem and Minimum Spanning Trees," *Operations Research*, pp. 1138–1162, 1970.

[71] M. Held, P. Wolfe, and H. Crowder, "Validation of Subgradient Optimization," *Mathematical programming*, vol. 6, no. 1, pp. 62–88, 1974.

[72] N. K. Shah and M. G. Ierapetritou, "Lagrangian decomposition approach to scheduling large-scale refinery operations," *Computers & Chemical Engineering*, vol. 79, pp. 1–29, 2015.

[73] H. Hosni, J. Naoum-Sawaya, and H. Artail, "The shared-taxi problem: Formulation and solution methods," *Transportation Research Part B: Methodological*, vol. 70, pp. 303–318, 2014.

[74] B. Ghaddar, J. Naoum-Sawaya, A. Kishimoto, N. Taheri, and B. Eck, "A lagrangian decomposition approach for the pump scheduling problem in water networks," *European Journal of Operational Research*, vol. 241, no. 2, pp. 490–501, 2015.

[75] G. R. Raidl, "Decomposition based hybrid metaheuristics," *European Journal of Operational Research*, 2014.

[76] M. Leitner and G. R. Raidl, "Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks," in *Hybrid Metaheuristics*. Springer, 2008, pp. 158–174.

[77] M. Leitner and G. R. Raidl, "Combining lagrangian decomposition with very large scale neighborhoood search for capacitated connected facility location," in *Post-Conference Book of the Eight Metaheuristics International Conference–MIC*, 2009.

[78] D. Thiruvady, G. Singh, and A. T. Ernst, "Hybrids of integer programming and aco for resource constrained job scheduling," in *Hybrid Metaheuristics*. Springer, 2014, pp. 130–144.

[79] M. Boschetti, V. Maniezzo, and M. Roffilli, "Decomposition techniques as metaheuristic frameworks," in *Matheuristics*. Springer, 2010, pp. 135–158.

[80] S. Mouret, I. E. Grossmann, and P. Pestiaux, "A new lagrangian decomposition approach applied to the integration of refinery planning and crude-oil scheduling," *Computers & Chemical Engineering*, vol. 35, no. 12, pp. 2750–2766, 2011.

[81] C. H. Rosa and A. Ruszczyński, "On augmented lagrangian decomposition methods for multistage stochastic programs," *Annals of Operations Research*, vol. 64, no. 1, pp. 289–309, 1996.

[82] A. Ruszczyński, "Decomposition methods in stochastic programming," *Mathematical programming*, vol. 79, no. 1-3, pp. 333–353, 1997.

[83] A. Ruszczynski, "Some advances in decomposition methods for stochastic linear programming," *Annals of Operations Research*, vol. 85, pp. 153–172, 1999.

[84] S. Ketabchi and M. Behboodi-Kahoo, "Augmented lagrangian method within l-shaped method for stochastic linear programs," *Applied Mathematics and Computation*, vol. 266, pp. 12–20, 2015.

[85] "Tabker Airlift Control Center, 618th Air and Space Operations Center (TACC) fact sheet." [Online]. Available: http://www.618tacc.amc.af.mil

[86] R. Glenn Richey Jr, M. Natarajarathinam, I. Capar, and A. Narayanan, "Managing supply chains in times of crisis: a review of literature and insights," *International Journal of Physical Distribution & Logistics Management*, vol. 39, no. 7, pp. 535–573, 2009.

[87] N. Altay and W. G. Green, "Or/ms research in disaster operations management," *European journal of operational research*, vol. 175, no. 1, pp. 475–493, 2006.

[88] U. Paulsson, "Supply chain risk management," *Supply chain risk: A reader*, pp. 79–96, 2004.

[89] S.-C. O. A. Haghani, "Testing and evaluation of a multi-commodity multi-modal network flow model for disaster relief management," *Journal of Advanced Transportation*, vol. 31, no. 3, pp. 249–282, 1997.

[90] G. Barbaroso&gcaron et al., "A two-stage stochastic programming framework for transportation planning in disaster response," *Journal of the Operational Research Society*, vol. 55, no. 1, pp. 43–53, 2004.

[91] M. Goh, J. Y. Lim, and F. Meng, "A stochastic model for risk management in global supply chain networks," *European Journal of Operational Research*, vol. 182, no. 1, pp. 164–173, 2007.

[92] H. K. Rappoport, L. S. Levy, B. L. Golden, and K. J. Toussaint, "A planning heuristic for military airlift," *Interfaces*, vol. 22, no. 3, pp. 73–87, 1992.

[93] B. M. Beamon and S. A. Kotleba, "Inventory modelling for complex emergencies in humanitarian relief operations," *International Journal of Logistics: Research and Applications*, vol. 9, no. 1, pp. 1–18, 2006.

[94] G. Barbarosoğlu, L. Özdamar, and A. Cevik, "An interactive approach for hierarchical analysis of helicopter logistics in disaster relief operations," *European Journal of Operational Research*, vol. 140, no. 1, pp. 118–133, 2002.

[95] D. Teodorovic, *Airline operations research*. Taylor & Francis, 1988, vol. 10.

[96] G. Yu and B. Thengvall, "Airline optimization," *Encyclopedia of Optimization*, pp. 26–30, 2009.

[97] C. Barnhart, P. Belobaba, and A. R. Odoni, "Applications of Operations Research in the Air Transport Industry," *Transportation science*, vol. 37, no. 4, pp. 368–391, 2003.

[98] O. R. N. Labs, "Consolidated Air Mobility Planning System (CAMPS): An Air Mobility Planning and Scheduling System, Research Brief," *Center for Transportation Analysis*. [Online]. Available: http://cta.ornl.gov

[99] S. F. Baker, D. P. Morton, R. E. Rosenthal, and L. M. Williams, "Optimizing Military Airlift," *Operations Research*, vol. 50, no. 4, pp. 582–602, 2002.

[100] J. Salmeron, R. K. Wood, and D. P. Morton, "A Stochastic Program for Optimizing Military Sealift Subject to Attack," *Military Operations Research*, vol. 14, no. 2, pp. 19–39, 2009.

[101] D. A. Goggins, "Stochastic modeling for airlift mobility," Ph.D. dissertation, Monterey, California. Naval Postgraduate School, 1995.

[102] C. Barnhart, "Analyzing passenger travel disruptions in the National Air Transportation System," no. 2005, 2010.

[103] C. Barnhart, D. Fearing, A. Odoni, and V. Vaze, "Demand and capacity management in air transportation," *EURO Journal on Transportation and Logistics*, vol. 1, no. 1-2, pp. 135–155, 2012.

[104] C. Barnhart, D. Fearing, and V. Vaze, "Modeling Passenger Travel and Delays in the National Air Transportation System," *Operations Research*, vol. 62, no. 3, pp. 580–601, 2014. [Online]. Available: http://0-pubsonline.informs.org.pugwash.lib.warwick.ac.uk/doi/abs/10.1287/opre.2014.1268

[105] C. S. Pun and D. Klabjan, "Air Cargo Allotment Planning," pp. 1–25, 2010.

[106] M. Sohoni, Y.-C. Lee, and D. Klabjan, "Robust Airline Scheduling Under Block-Time Uncertainty," *Transportation Science*, vol. 45, no. 4, pp. 451–464, 2011.

[107] R. Sandhu and D. Klabjan, "Integrated Airline Fleeting and Crew-Pairing Decisions," *Operations Research*, vol. 55, no. 3, pp. 439–456, 2007.

[108] S. Shebalov and D. Klabjan, "Robust Airline Crew Pairing: Move-up Crews," *Transportation Science*, vol. 40, no. 3, pp. 300–312, 2006.

[109] D. Klabjan, E. L. Johnson, and G. L. Nemhauser, "Airline Crew Scheduling with Time Windows and Plane Count Constraints," pp. 1–21, 2001.

[110] D. Klabjan, E. L. Johnson, G. L. Nemhauser, E. Gelman, and S. Ramaswamy, "Airline Crew Scheduling with Time Windows and Plane-Count Constraints," *Transportation Science*, vol. 36, no. 3, pp. 337–348, 2002.

[111] D. Klabjan, E. L. Johnson, and G. L. Nemhauser, "Solving Large Airline Crew Scheduling Problems : Random Pairing Generation and Strong Branching," pp. 1–23, 2000.

[112] D. Klabjan, E. L. Johnson, G. L. Nemhauser, E. Gelman, and S. Ramaswamy, "Airline Crew Scheduling with Regularity," *Transportation Science*, vol. 35, no. 4, pp. 359–374, 2001.

[113] A. Makri and D. Klabjan, "A New Pricing Scheme for Airline Crew Scheduling," Urbana, IL.

[114] C. A. Hane, C. Barnhart, E. L. Johnson, R. E. Marsten, G. L. Nemhauser, and G. Sigismondi, "The fleet assignment problem: solving a large-scale integer program," *Mathematical Programming*, vol. 70, no. 1-3, pp. 211–232, 1995.

[115] A. J. Schaefer, E. L. Johnson, A. J. Kleywegt, and G. L. Nemhauser, "Airline crew scheduling under uncertainty," *Transportation science*, vol. 39, no. 3, pp. 340–348, 2005.

[116] J. M. Rosenberger, E. L. Johnson, and G. L. Nemhauser, "Rerouting aircraft for airline recovery," *Transportation Science*, vol. 37, no. 4, pp. 408–421, 2003.

[117] L. Clarke, E. Johnson, G. Nemhauser, and Z. Zhu, "The aircraft rotation problem," *Annals of Operations Research*, vol. 69, pp. 33–46, 1997.

[118] L. W. Clarke, C. A. Hane, E. L. Johnson, and G. L. Nemhauser, "Maintenance and crew considerations in fleet assignment," *Transportation Science*, vol. 30, no. 3, pp. 249–260, 1996.

[119] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser, "Airline crew scheduling: A new formulation and decomposition algorithm," *Operations Research*, vol. 45, no. 2, pp. 188–200, 1997.

[120] C. Barnhart, A. M. Cohn, E. L. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance, "Airline crew scheduling," in *Handbook of transportation science*. Springer US, 2003, pp. 517–560.

[121] C. Barnhart, N. L. Boland, L. W. Clarke, E. L. Johnson, G. L. Nemhauser, and R. G. Shenoi, "Flight string models for aircraft fleeting and routing," *Transportation science*, vol. 32, no. 3, pp. 208–220, 1998.

[122] J. M. Rosenberger, A. J. Schaefer, D. Goldsman, E. L. Johnson, A. J. Kleywegt, and G. L. Nemhauser, "Air transportation simulation: Simair: a stochastic model of airline operations," in *Proceedings of the 32nd conference on Winter simulation*. Society for Computer Simulation International, 2000, pp. 1118–1122.

[123] D. Espinoza, R. Garcia, M. Goycoolea, G. L. Nemhauser, and M. W. Savelsbergh, "Per-seat, on-demand air transportation part ii: Parallel local search," *Transportation Science*, vol. 42, no. 3, pp. 279–291, 2008.

[124] E.-P. Chew, H.-C. Huang, E. L. Johnson, G. L. Nemhauser, J. S. Sokol, and C.-H. Leong, "Short-term booking of air cargo space," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1979–1990, 2006.

[125] E. Pruul, G. Nemhauser, and R. Rushmeier, "Branch-and-bound and parallel computation: A historical note," *Operations Research Letters*, vol. 7, no. 2, pp. 65–69, 1988.

[126] Z. Gu, E. L. Johnson, G. L. Nemhauser, and W. Yinhua, "Some properties of the fleet assignment problem," *Operations Research Letters*, vol. 15, no. 2, pp. 59–71, 1994.

[127] D. Klabjan, E. L. Johnson, G. L. Nemhauser, E. Gelman, and S. Ramaswamy, "Airline crew scheduling with regularity," *Transportation science*, vol. 35, no. 4, pp. 359–374, 2001.

[128] P. H. Vance, A. Atamturk, C. Barnhart, E. Gelman, E. L. Johnson, A. Krishna, D. Mahidhara, G. L. Nemhauser, and R. Rebello, "A heuristic branch-and-price approach for the airline crew pairing problem," *preprint*, 1997.

[129] D. Klabjan, E. L. Johnson, G. L. Nemhauser, E. Gelman, and S. Ramaswamy, "Solving large airline crew scheduling problems: Random pairing generation and strong branching," *Computational Optimization and Applications*, vol. 20, no. 1, pp. 73–91, 2001.

[130] J. M. Rosenberger, E. L. Johnson, and G. L. Nemhauser, "A robust fleet-assignment model with hub isolation and short cycles," *Transportation science*, vol. 38, no. 3, pp. 357–368, 2004.

[131] J. M. Rosenberger, A. J. Schaefer, D. Goldsman, E. L. Johnson, A. J. Kleywegt, and G. L. Nemhauser, "A stochastic model of airline operations," *Transportation science*, vol. 36, no. 4, pp. 357–377, 2002.

[132] L. Lettovskỳ, E. L. Johnson, and G. L. Nemhauser, "Airline crew recovery," *Transportation Science*, vol. 34, no. 4, pp. 337–348, 2000.

[133] M. Ball, C. Barnhart, G. Nemhauser, and A. Odoni, "Air transportation: Irregular operations and control," *Handbooks in operations research and management science*, vol. 14, pp. 1–67, 2007.

[134] D. Klabjan, E. L. Johnson, G. L. Nemhauser, E. Gelman, and S. Ramaswamy, "Airline crew scheduling with time windows and plane-count constraints," *Transportation science*, vol. 36, no. 3, pp. 337–348, 2002.

[135] G. L. Nemhauser, "Column generation for linear and integer programming," *Optimization Stories*, vol. 20, p. 64, 2012.

[136] A. G. Loerch, N. Boland, E. L. Johnson, and G. L. Nemhauser, "Finding an optimal stationing policy for the us army in europe after the force drawdown," *Military Operations Research*, vol. 2, no. 4, pp. 39–51, 1996.

[137] J. M. Mulvey, R. J. Vanderbei, S. a. Zenios, and S. a. Z. John M Mulvey, Robert J Vanderbei, "Robust optimization of large-scale systems," pp. 264–281, 1995. [Online]. Available: http://www.jstor.org/stable/171835

[138] S. F. Baker, D. P. Morton, R. E. Rosenthal, and L. M. Williams, "Monterey, California," Tech. Rep. April, 1999.

[139] S. F. Baker, D. P. Morton, R. E. Rosenthal, and L. M. Williams, "Optimizing Military Airlift," *Operations Research*, vol. 50, no. 4, pp. 582–602, 2002.

[140] R. G. McGarvey, T. Light, B. Thomas, and R. Sanchez, "Commercial Intratheater Airlift: Cost-Effectiveness Analysis of Use in U.S. Central Command," RAND Corporation, Tech. Rep., 2013.

[141] M. J. Williams, "Column Generation Approaches to the Military Airlift Scheduling Problem," Ph.D. dissertation, Massachusetts Institute of Technology, 2014. [Online]. Available: http://hdl.handle.net/1721.1/91297

[142] C. Pflieger, "Models for the Optimization of Air Refueling Missions," 1993. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA262392

[143] G. Stojković, F. Soumis, J. Desrosiers, and M. M. Solomon, "An optimization model for a real-time flight scheduling problem," *Transportation Research Part A: Policy and Practice*, vol. 36, no. 9, pp. 779–788, 2002.

[144] K. W. Kopp, "Improvement of Air Mobility Command Airlift Scheduling," Ph.D. dissertation, Air Force Institute of Technology, 2004.

[145] L. Kramer and S. Smith, "Optimizing for change: Mixed-initiative resource allocation with the AMC Barrel Allocator," in *the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002. [Online]. Available: http://swing.adm.ri.cmu.edu/pub_files/pub4/kramer_laurence_2002_1/kramer_laurence_2002_1.pdf

[146] G. G. Brown, W. M. Carlyle, and R. F. Dell, "Optimizing Intratheater Military Airlift in Iraq and Afghanistan," *Military Operations Research*, vol. 18, no. 3, pp. 35–52, 2013. [Online]. Available: http://openurl.ingenta.com/content/xref?genre=article&issn=1082-5983&volume=18&issue=3&spage=35

[147] R. a. Rushmeier, "Recent Advances in Exact Optimization of Airline Scheduling Problems," *New York*, 1995.

[148] S. Smith, M. Becker, and L. Kramer, "Continuous management of airlift and tanker resources: A constraint-based approach," *Mathematical and Computer Modelling*, vol. 39, no. 6-8, pp. 581–598, 2004.

[149] D. E. Wilkins, S. F. Smith, L. a. Kramer, T. J. Lee, and T. W. Rauenbusch, "Airlift mission monitoring and dynamic rescheduling," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 2, pp. 141–155, 2008.

[150] T. T. Wu, W. B. Powell, and A. Whisman, "The optimizing-simulator," *ACM Transactions on Modeling and Computer Simulation*, vol. 19, no. 3, pp. 1–31, 2009.