Empirical Analysis of Algorithms for Block-Angular Linear Programs

by

Jiarui Dang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 2007

Your file Votre référence
ISBN: 978-0-494-35122-2
Our file Notre référence
ISBN: 978-0-494-35122-2

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Jiarui Dang

Abstract

This thesis aims to study the theoretical complexity and empirical performance of decomposition algorithms. We focus on linear programs with a block-angular structure. Decomposition algorithms used to be the only way to solve large-scale special structured problems, in terms of memory limit and CPU time. However, with the advances in computer technology over the past few decades, many large-scale problems can now be solved simply by using some general purpose LP software, without exploiting the problems' inner structures. A question arises naturally, should we solve a structured problem with decomposition, or directly solve it as a whole? We try to understand how a problem's characteristics influence its computational performance, and compare the relative efficiency of algorithms with and without decomposition. Two comparisons are conducted in our research: first, the Dantzig-Wolfe decomposition method (DW) versus the simplex method (simplex); second, the analytic center cutting plane method (ACCPM) versus the interior point method (IPM). These comparisons fall into the two main solution approaches in linear programming: simplex-based algorithms and IPM-based algorithms. Motivated by our observations of ACCPM and DW decomposition, we devise a hybrid algorithm combining ACCPM and DW, which are the counterparts of IPM and simplex in the decomposition framework, to take the advantages of both: the quick convergence rate of IPM-based methods, as well as the accuracy of simplex-based algorithms. A large set of 316 instances is incorporated in our experiments, so that different dimensioned problems with primal or dual block-angular structures are covered to test our conclusions.

Acknowledgements

It has been a long journey for me to do this PhD, and I have not been successful alone. I first would like to thank my supervisors, Professor Fuller and Professor Sundarraj, for their support and guidance throughout these years. Some people say that doing research is like working in a tunnel: when you see a beam of light, the end is right in front of you; but when you are still digging in the dark and do not see where it leads to, it can be quite frustrating. I had a difficult time at the beginning of my research, where I could not see the future and was unsure about my work. It was the invaluable advice and supportive comments from my supervisors that encouraged me and helped me out of my depression. Confidence and persistence are the most important things that I have learned from my research experience.

I would like to take this chance to thank Professor Elhedhli, who has research interests related to ours, and who kindly helped me when I encountered difficulties during the experiments. Without his suggestions, I could have remained stuck on one point for a long time. I also want to express my gratitude to my committee members, Professor Calamai, Professor Anjos, and Professor Hooker, for reviewing my research papers and giving helpful feedback.

My special thanks go to Professor McKay. I worked as his teaching assistant for three years. In his class, I learned a lot about research methods and teaching techniques.

Family and friends give sparkles to my life. I treasure the moments that I have spent with you, and you strengthen all facets of my life. I affectionately thank my husband, parents, and brother, for supporting me throughout my academic pursuits.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

For over 50 years, Operations Research (OR) has been using linear programming models to aid decision-making. The problem size that can be solved has increased dramatically with the impressive development of computer technologies. Today, a variety of commercial software is available to solve general linear programs, but they still have a capacity limit for constraints and variables [28]. Unfortunately, many application formulations may greatly exceed the existing limit. There are two types of approaches available to deal with this issue [13]: the first is to partition the overall problem into manageable subproblems, linked by means of a hierarchical interactive system, and the second is to devise specialized algorithms, called *decomposition* algorithms, to exploit the structure of the problem. The first approach is not applicable for a lot of problems with a *monolithic* structure. The second approach is the most commonly used alternative to treat the problem as a unit. In this thesis, we focus mainly on the Dantzig-Wolfe decomposition method (DW), one of various available decomposition algorithms.

Another method used in our analysis is the Analytic Center Cutting Plane Method (ACCPM), which is a new and promising algorithm with good performance in both theory and practice. ACCPM uses ideas from Interior Point Methods (IPMs) to calculate analytic centers. This step also dominates the computational effort at each iteration. Moreover, due to the dual equivalence between Lagrangian relaxation and Dantzig-Wolfe decomposition, ACCPM applied in decomposition can be viewed as one variant of the DW algorithm.

1.1 Decomposition or Not

The advent of decomposition methods has solved the insufficient memory problem in computing solutions to some large models, and hence avoided the requirement of an expensive super computer. Decomposition methods also make it possible to use parallel computing with affordable multi-processors. However, although decomposition methods have the aforementioned advantages, their theoretical complexity and practical performance are often not satisfactory. Convergence can be proven, but the convergence rate is usually slow. In addition, nowadays, due to the breakthrough of software and hardware development, personal computers today have much larger processing abilities which were unimaginable in the past. So, do we really need decomposition? A common accepted argument is that for very large-scale applications, decomposition algorithms would hopefully outperform standard methods. But, how large is really large? How does one measure the scale of an instance? Are there any other factors impacting the computational effort? There have been no clear answers to these questions at present, although the complexity of decomposition has long been a concern by researchers. We will try to answer these questions in this thesis.

Based on some conclusions available in the literature, we intend to give a complete picture of the complexity of decomposition algorithms. We will then compare the complexity of decomposition algorithms against the complexity of direct solution approaches. In particular, we will consider block-angular structured linear programs, whose size can be measured by dimensions, such as the number of linking constraints, the number of blocks, and the number of rows and columns in each block. Therefore, the complexity of a problem can be described as a function of its dimensional parameters, to indicate the relationship between a problem's size and its computational effort. We try to give a few conclusions that are helpful for decision making, *i.e.*, for a specific structured problem, whether or not to use decomposition.

Two comparisons are conducted in our analysis: DW versus simplex (simplex-based algorithms, with and without decomposition), and ACCPM versus IPM (IPM-based algorithms, with and without decomposition). Both comparisons are discussed according to some theoretical conclusions and then tested empirically.

The theoretical analysis on the first comparison, DW vs. simplex, is based on simplex

methods. Since simplex has a well known exponential worst-case complexity, which rarely occurs in practice, it is not suitable to analyze in the worst-case. However, with a much better average performance, simplex is still a competitive algorithm for linear programming. Note that it is not statistically 'average', but a widely accepted 'average-case observed performance', which can reflect the real performance of the simplex method. We will use results from the average-case to analyze the overall complexity for both DW and simplex.

The theoretical analysis on the second comparison, ACCPM vs. IPM, is based on interior point theories. There are some previous results available on the complexity of ACCPM with multiple cuts, but we have gone further by analyzing the complexity of ACCPM applied in decomposition, taking into account the complexity of subproblems. All the results in this part are from the worst-case estimates.

Besides the theoretical analysis, we also do plenty of experiments to test our conclusions. Empirical analysis is as important as the theoretical ones in Operations Research, if not more. Researchers like McGeoch [65][66][67] and Hooker [49] strongly suggest empirical analysis, because there is a huge gap between *theory* (abstract model) and *practice* (physical artifact). A lot of assumptions, which are often untrue in the real world, have been made to build an ideal mathematical model. Sometimes a worst-case theoretical result of an algorithm cannot tell us how exactly it works in practice. Unfortunately, few people have realized this. We still see some papers with propositions, proofs, conclusions, and then a last page containing numerical results. This kind of approach is not enough because a few examples may not be able to represent the whole picture, and many other factors count as well.

To overcome the limit of theoretical analysis, several encouraging methods have been suggested by empirical studies. Statistical models are one type of these methods. Based on the numerical results, we used regression analysis to test our theoretical conclusions. A regression model can be built by examining the influence of each parameter, *e.g.*, the more linking constraints a problem has, the more computational effort a decomposition method usually requires. However, due to the joint contribution of various dimensional parameters, a good model can become quite difficult to develop. Under such circumstances, a pure empirical analysis approach is adopted, *i.e.*, putting theoretical results aside, testing all available problems with specific dimensional parameters, and then developing tentative

regression models by observing the tendency shown in data.

Empirical analysis usually requires a large amount of samples in order to be convincing. Instead of randomly choosing testing problems, we use a large set of test problems to make the analysis systematic. Our computational tests employ many block-angular problems with deliberately chosen variations in the dimensional parameters such as the number of linking constraints, the number of blocks, and the number of variables and constraints in each subproblem.

1.2 A Hybrid Solution Approach Combining ACCPM and DW

Convergence speed and accuracy are both important criteria in choosing an algorithm. Yet, in linear programming, these two criteria appear to be in conflict for the two main solution approaches: the simplex method and interior point methods (IPMs). The analytical center cutting plane method (ACCPM), which can be viewed as an IPM-based decomposition approach, has superior global convergence properties, but its main feature, *staying away from the boundary*, prevents it from producing an exact optimal solution. Conversely, the simplex-based Dantzig-Wolfe decomposition method (DW) achieves greater accuracy because it can reach the vertices of the feasible region. In this thesis, we propose a hybrid solution approach that seeks to combine the advantages of both. We start with ACCPM, and then switch to DW after a few iterations (usually when the current point is sufficiently close to the optimal solution). There is little computational effort required for the switch. Experiments indicate that for large problems, the hybrid approach appears to have both accuracy and a fast convergence rate.

1.3 Our Contributions and Structure of the Thesis

Our contributions are three-fold. First, we analyze how a problem's structure impacts its computational effort, deduce theoretical complexity conclusions on both decomposition and non-decomposition methods, and then empirically test them with rigorously designed

experiments. This study aims to provide useful information for decision making, *e.g.*, for a specific structured problem, solving it by decomposition or solving it as a whole, using serial or parallel computing, how to design the experiment architecture, and even predicting an algorithm's performance on a given problem. Second, while IPM has been involved in the DW algorithm (see [81], [63], and [87]), a hybrid method combining ACCPM and DW is a new attempt, and has not been proposed in the literature. Several techniques, such as the weighted versions of ACCPM [24] and DW, the constructed master problem [24], and a variant of warm-start recovery, are incorporated and make the hybrid approach competitive. Third, in order to provide full-scale numerical results, our computational tests involve primal and dual block-angular structured problems with various dimensional parameters.

The present exposition is not meant to be exhaustive, as algorithms as well as research methods are versatile, and real-world problems are greatly varied. The rest of the thesis is organized as follows:

Chapter 2 is composed of related basic principles. First, structured linear programming problems are introduced, particularly the block-angular structure, along with its applications. Then, a few solution methods are briefly introduced. Two main categories are covered: simplex-based and IPM-based methods. The theoretical and empirical analysis, the regression analysis, and experimental design are also discussed in this chapter.

Chapter 3 addresses detailed algorithms and notations. There are two main categories: one, simplex-based algorithms, including simplex, revised simplex method, and its application in large-scale systems, the Dantzig-Wolfe decomposition method; two, interior point methods and their application in decomposition, ACCPM. A comparison of these two prime solution methods is outlined from different perspectives.

Chapters 4 and 5 analyze the relative efficiency of *with* and *without* decomposition using two comparisons. Chapter 4 considers the first comparison: DW versus simplex, in the average case. We briefly recall the observed behavior and simplex-based algorithms, and then examine the real performance of both simplex and DW over the test problems. Chapter 5 considers the second comparison: ACCPM versus IPM, in the worst case. We first analyze the complexity of ACCPM and IPM, based on some results from the literature. We try to give a whole picture of the complexity for ACCPM applied in decomposition, and

then investigate the influence by each dimensional parameter that describes the problems structure. A few conclusions are made on the complexity analysis, and are tested by experiments.

Empirical studies are outlined in Chapters 4 and 5. We test our preliminary conclusions by solving the sample problems, with and without decomposition. Instead of including a 'last page empirical results' section, we design some statistical models to make our discussion more convincing. In addition, exploratory data analysis is performed in our discussion. That is, we observe the computational results, which can possibly provide some hints to build a regression model. This method is particularly effective for complex data analysis.

When we coded ACCPM and DW in MATLAB, we realized that convergence and accuracy appear to be in contradiction, which is the motivation for us to devise a hybrid decomposition approach to take advantages of both. This hybrid algorithm is presented in Chapter 6, and numerical experiments show promising results.

Chapter 7 summarizes the implementation issues, including software and hardware, as well as strategies in large-scale computing. In particular, we propose two techniques used in our ACCPM code: penalty weight factors and the dynamic update of the traditional big M.

In the last chapter, we make concluding remarks and anticipate our future research.

Chapter 2

Structured Problems and Methodology

In this chapter, we briefly introduce the block-angular structure and two real-world applications, as well as the research methods used in this thesis.

2.1 Structured Problems for Linear Programming

Large-scale problems are usually quite sparse. In this context, structure means the pattern of zero and nonzero coefficients in the constraint matrices. Several structural forms reappear frequently in real-world applications [13]. The special structures of these problems can be exploited by decomposition algorithms.

Our work focuses on linear programs with a block-angular structure. Consider a problem with the form of

$$\begin{array}{lllll} \min & c_1^T x_1 + & c_2^T x_2 + & \cdots + & c_h^T x_h & \\ s.t. & A_1 x_1 + & A_2 x_2 + & \cdots + & A_h x_h & = b \\ & B_1 x_1 & & & & = d_1 \\ & & B_2 x_2 & & & = d_2 \\ & & & \cdots & & \cdots \\ & & & & B_h x_h & = d_h \\ & x_l \geq 0, & & & l & = 1, 2, \cdots, h \end{array} \tag{2.1}$$

where $A_l \in R^{m_0 \times n_l}$, and $b \in R^{m_0 \times 1}$; each block $B_l \in R^{m_l \times n_l}$, $d_l \in R^{m_l \times 1}$, $c_l \in R^{n_l \times 1}$, and the variables $x_l \in R^{n_l \times 1}$. Denote $A = [A_1, A_2, \cdots, A_h]$, $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_h \end{bmatrix}$, $c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_h \end{bmatrix}$,

$B = \begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \cdots & \\ & & & B_h \end{bmatrix}$, and $d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_h \end{bmatrix}$. The problem becomes

$$\min \quad c^T x \tag{2.2}$$

$$s.t. \quad Ax = b \tag{2.3}$$

$$Bx = d \tag{2.4}$$

$$x \geq 0 \tag{2.5}$$

Notice that in this problem, there are many zeros in the coefficients, and the remaining nonzeros occur in clusters. The system of equations (2.4) can be separated into h completely independent subsystems, while constraints (2.3) act as a linkage between these smaller blocks. This special pattern is called *primal block-angular* structure.

2.2 Examples of Block Angular Structure

Many problems arising in application have a block-angular structure. In this thesis, we include two models (316 instances) for the numerical experiments.

2.2.1 Stochastic Financial Model (Dual Block-Angular)

We consider a two-stage stochastic portfolio problem based on a multistage model in [29]. The problem can be viewed as an event tree. For simplicity, we assume that each node occurs with the same probability. The root node represents time point 0, where w_{in} is the amount of initial funds, and n is the number of assets. Every stage is a decision process: all assets are sold and then re-invested at the next stage. Transaction cost is ignored here,

because although it is important in financial analysis, it does not affect the computational properties. Us and Vs are evaluated at end leaf nodes (stage 2) as surplus and deficit. If there is a surplus (U) above the expected return (w_{out}), the investment is successful, so a bonus scalar is assigned to multiply the Us in the objective function. On the contrary, deficit (V) is not desirable, so a penalty scalar multiplies the Vs in the objective function. We use the same bonus and penalty scalars (5 and -20 respectively) as [29].

A two-stage stochastic problem can be formulated as

$$\begin{aligned} max \quad & \sum_{l_2=1}^{L_2} 5U^{l_2} - 20V^{l_2} \\ s.t. \quad & w_{in} = C^{l_0} + \sum_{i=1}^{n} X_i^{l_0}, && l_0 = 1 \\ & r_c C^{a(l_1)} + \sum_{i=1}^{n} p_i^{l_1} X_i^{a(l_1)} = C^{l_1} + \sum_{i=1}^{n} X_i^{l_1}, && l_1 = 1, \cdots, L_1 \\ & r_c C^{a(l_2)} + \sum_{i=1}^{n} p_i^{l_2} X_i^{a(l_2)} = w_{out} + U^{l_2} - V^{l_2}, && l_2 = 1, \cdots, L_2 \\ & C^{a(l_t)}, X_i^{a(l_t)}, U^{l_2}, V^{l_2} \geq 0, \end{aligned} \tag{2.6}$$

where i is the index for assets ($i = 1, 2, \cdots, n$), t is the index for stages ($t = 1, 2$), l_t is the node index in stage t, $a(l_t)$ represents the immediate ancestor of node l_t, and L_t is the total number of nodes in stage t . Variable C^{l_t} is the amount of cash in stage t, and variable $X_i^{l_t}$ is the amount of money invested in asset i in stage t. Parameter r_c represents the rate of return for cash, and $p_i^{l_t}$ represents the rate of return for asset i in stage t. We set $r_c \in [0.8, 1.2]$ and $p_i^{l_t} \in [0.8, 1.2]$ in our experiments.

The stochastic financial model has a dual block-angular structure as shown in Figure 2.1, where the horizontal axis represents the row count, the vertical axis represents the column count, and nz stands for the number of nonzeros. Consequently, the dual problem of (2.6) has a primal block angular structure as in (2.1). The first stage variables C^{l_0} and $X_i^{l_0}$ correspond to the linking constraints in the dual problem, and the remaining variables C^{l_1}, $X_i^{l_1}$, V^{l_2} and U^{l_2} correspond to L_1 blocks of constraints in the dual problem. Since there are h branchings at each node, $L_1 = h$, and $L_2 = h^2$.

There are only two parameters that determine the problem dimensions: the number of assets (n) and the number of scenarios (h). The dual of the stochastic financial problem (2.6) has $h^2 + h + 1$ variables and $2h^2 + nh + h + n + 1$ constraints, of which $n + 1$ are linking

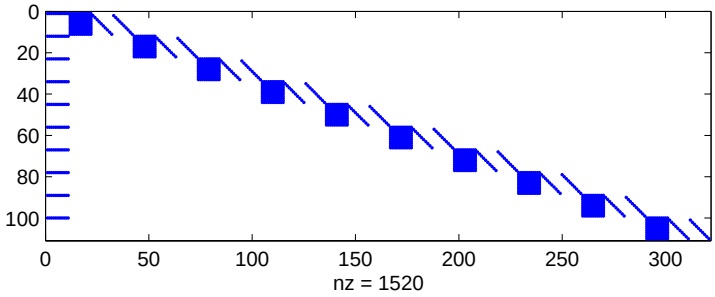


Figure 2.1: Stochastic Financial Model with Dual Block-angular Structure

constraints. Thus, the number of linking constraints is only determined by the number of assets, and is independent of the number of scenarios. As is known, in a stochastic problem, the more scenarios, the more accurate is the model. We can address the problem accurately, with a lot of scenarios, and not affect the number of linking constraints. This is a very nice feature for decomposition algorithms.

We include 100 test problems from this model in our experiments, with n from 10 to 100, h from 10 to 100, and stepsize 10. Table 2.1 shows the problem characteristics of 16 of them, with all combinations of n and h equal to 10, 40, 70, and 100. In the table, m_0 represents the number of linking constraints, and $m_0\%$ is the percentage of linking constraints to all constraints. *rows* and *cols* [1] stand for the total number of rows and columns in the whole model. nz and $nz\%$ are the number of non-zeros and their percentage in the whole model. The last two columns, row_s and col_s, represent the number of rows and columns in each subproblem. All the h subproblems are of the same size. For more information on all the 100 test problems, see Table A.1 in Appendix A for more information.

2.2.2 Multicommodity Network Flow Model (Primal Block-Angular)

The multicommodity network flow problem is a typical model with a primal block-angular structure (see Figure 2.2). Several different commodities share the same transportation network, and are shipped simultaneously from their respective sources to destinations,

[1] Refers to rows and columns in primal block angular structure (dual formulation).

Table 2.1: Stochastic Financial Problem Characteristics (16 out of 100)

Prob.	n	h	m_0	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
1	10	10	11	321	3.43%	111	1,531	4.30%	31	11
2	10	40	11	3,651	0.30%	1,641	21,691	0.36%	91	41
3	10	70	11	10,581	0.10%	4,971	65,251	0.12%	151	71
4	10	100	11	21,111	0.05%	10,101	132,000	0.06%	211	101
5	40	10	41	651	6.30%	111	5,161	7.14%	61	11
6	40	40	41	4,881	0.84%	1,641	72,121	0.90%	121	41
7	40	70	41	12,711	0.32%	4,971	216,000	0.34%	181	71
8	40	100	41	24,141	0.17%	10,101	438,000	0.18%	241	101
9	70	10	71	981	7.24%	111	8,791	8.07%	91	11
10	70	40	71	6,111	1.16%	1,641	123,000	1.22%	151	41
11	70	70	71	14,841	0.48%	4,971	368,000	0.50%	211	71
12	70	100	71	27,171	0.26%	10,101	744,000	0.27%	271	101
13	100	10	101	1,311	7.70%	111	12,421	8.54%	121	11
14	100	40	101	7,341	1.38%	1,641	173,000	1.44%	181	41
15	100	70	101	16,971	0.60%	4,971	519,000	0.62%	241	71
16	100	100	101	30,201	0.33%	10,101	1,050,000	0.34%	301	101

with the total flow of each arc not exceeding its capacity.

Given a directed graph $G(N, A)$, with N representing the set of nodes ($|N| = n$), and A representing the set of arcs in the network ($|A| = m$), a multicommodity network flow problem can be formulated as follows

$$\min \quad \sum_k \sum_{ij} c_{ij}^k x_{ij}^k \tag{2.7}$$

$$s.t. \quad \sum_j x_{ij}^k - \sum_j x_{ji}^k = b_i^k \qquad \forall i, k \tag{2.8}$$

$$0 \leq x_{ij}^k \leq u_{ij}^k \qquad \forall i, j, k \tag{2.9}$$

$$\sum_k x_{ij}^k \leq u_{ij} \qquad \forall i, j \tag{2.10}$$

$$x_{ij}^k \geq 0, \tag{2.11}$$

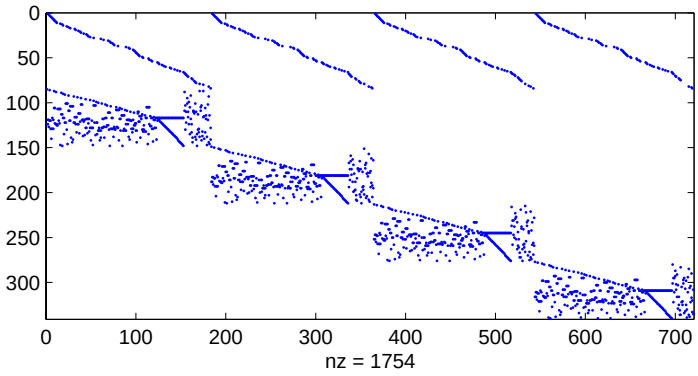


Figure 2.2: Multicommodity Network Flow Model with Primal Block-angular Structure

where $k = 1, 2, \cdots, h$ is the index of commodities, node i and $j \in N$, and arc $(i, j) \in A$. The constraint (2.8) represents the flow conservation with b_i^k as net supply of commodity k at node i, (2.9) limits the individual capacity by commodity for each arc, and (2.10) is the mutual capacity. With h commodities, n nodes, and m_0 out of m arcs having mutual capacities, the problem has $m \times h$ variables and $n \times h + m \times h + m_0$ constraints, of which m_0 of them are linking constraints crossing the h subproblems.

We obtain the data from the well-known Mnetgen generator [30][14]. The number of nodes $n \in \{64, 128, 256\}$, and the number of commodities $h \in \{4, 8, \cdots, n\}$ as Mnetgen can only generate problems with $h \leq n$. For each pair of (n, h), 12 problems are randomly generated. They can be classified into four groups. Six of them are of more arcs among the same number of nodes in the network (*dense*),with $m/n \approx 8$; the other six are of *sparse* network with $m/n \approx 3$. In terms of the number of arcs with mutual capacities, six of the problems are *hard* with 80% arcs having mutual capacity constraints (linking constraints), among which 30% are high cost; the other six are *easy* with 40% of arcs having mutual capacity and 10% are high cost. The combination of the two features make the four groups [14], as shown in Table 2.2. Within each group, the three problems have similar structure, and 30%, 60%, and 90% of arcs respectively have individual capacity constraints, which can be viewed as upper bounds for variables. Other arcs are treated as having infinite individual capacity (value Inf in MATLAB). Therefore, the percentage of arcs with individual capacity does not affect the problem size in this sense. However,

Table 2.2: The Four Types of Problems by Mnetgen

Type	Sparse	Dense
Easy	I	III
Hard	II	IV

[14] indicates that within each group of three, the difficulty of problems decreases as this percentage increases, because 'the number of *tight* mutual constraints tends to diminish'.

Table 2.3 summarizes the problem characteristics for this model. It only contains 24 out of the 216 problems: the first 12 ($n = 64, h = 4$) and last 12 ($n = 256, h = 256$) problems. For each pair of (n, h), problem 1-3 are type I, 4-6 are type II, 7-9 are type III, and 10-12 are type IV. In the table, row' is the original number of all the constraints, $rows$ is the number of constraints excluding the individual capacity ones in (2.9) because they are treated as upper bounds for the variables in our codes, $m_0\%$ is the percentage of linking constraints to all the constraints $rows$ (not including the individual capacity constraints), $cols$ is the number of columns, nz is the number of non-zeros in the coefficient matrix, and $nz\%$ is the percentage of non-zeros. The last two columns in Table 2.3 provide the average number of rows (not including the individual capacity constraints) and columns for the subproblems. Within each block, the number of rows equals the number of nodes (n); theoretically, the number of columns equal the number of arcs (m), but the actual number of the columns is quite different when Mnetgen generates problems. So, row_s and col_s show the average number of rows and columns of the blocks. For more information on all the 216 test problems, see Table A.2 in Appendix A.

2.3 Overview of Methodology

This section surveys methodological issues that arise in our research on evaluating optimization algorithms.

2.3.1 Theoretical vs. Empirical Analysis

To study the performance of algorithms, there are two possible approaches: *theoretical* and *empirical*. The former, also known as analytical or deductive analysis, has fully developed into a science [49]. Many brilliant results have been discovered by this approach in scholarly publications. The empirical analysis, on the other hand, has long been considered 'lowbrow or unsophisticated' [49] and has not gained as much attention as it deserves. However, the conclusions drawn from the widespread analytical method do not tell us how an algorithm really works in practice, both in the worst case and in the average case, because the proof is usually based on a simplified algorithm with strong assumptions.

It is easy to understand that worst-case complexity bounds are not much guide in practice. Moreover, theoretical average-case analysis also has plenty of limitations. In fact, average-case analysis is more difficult than worst-case [16]. Instead of giving a bound, average-case analysis needs to sample from the infinite population, and examines the average performance of every problem in the sample space [83]. It is virtually impossible to obtain a statistical average. Researchers have to assume a certain kind of distribution pattern, which may not be representative of real world applications.

Therefore, empirical analysis has been proposed as an alternative research method to address these problems [49]. As a method based on computational experiments, empirical analysis can sidestep the unrealistic assumptions and focus on typical problems. The advantages of an empirical science can be summarized as [49]:

1. It does not rely on proving hard worst-case and average-case theorems.
2. Unlike worst-case analysis, it can focus on typical problems.
3. Unlike average-case analysis, it need not restrict itself to a simple and unrealistic distribution of random problems.

In OR, a famous example of the unrepresentative worst-case analysis is for the simplex method [18]. Although it has an exponential complexity bound, its empirical performance is so efficient that it once discouraged the development of other methods in 1950-60s [81]. Even today, the simplex method is still a major solution approach for linear programming.

Due to the huge gap between its theoretical worst-case analysis and practice, people estimate simplex's performance by average-case analysis. Borgwardt [12] has studied the average behavior of simplex in a probabilistic manner. His analysis explains why simplex takes polynomial time to reach optimality in practice, yet some probabilistic assumptions are involved inevitably. Some other average-case observations are also available in the literature. These studies provide valuable information for the practical performance of simplex, and we will discuss them in detail Chapter 4.

IPM-based algorithms have polynomial worst-case complexity, which does not deviate from their practical performance as much as simplex-based algorithms. In Chapter 5, since most of the conclusions in the literature are worst-case complexity bounds, in the theoretical analysis part, we also compare ACCPM and IPM in the worst-case. The following empirical analysis then will address the observed average case.

2.3.2 Experimental Design

In fact, numerical results, which is part of empirical analysis, have already been involved in most research papers. However, there is still a long way to go before empirical analysis is improved to be a science: the computational testing is quite informal and needs more rigorously set principles of experimental design [42][66][67].

In this thesis, we try to understand how a problem's characteristics influence an algorithm's performance. More specifically, we describe the performance as a function of the dimensional parameters, *i.e.*,

$$performance = f(dimensional\ parameters), \tag{2.12}$$

where $performance$, usually known as *responses*, can be the number of iterations, the number of arithmetic operations, CPU time, and the relative efficiency of with and without decomposition; the dimensional parameters, which are the *factors*, include m_0, h, and $m_{sub} \times n_{sub}$. In addition, we also use $\overline{m} = m_0 + hm_{sub}$ and $\overline{n} = hn_{sub}$ as the total number of rows and columns for the entire problem when investigating the direct solution approaches.

On both the comparisons discussed in Chapter 4 and 5, basically, we first examine the performance of algorithms theoretically, and then test the conclusions by numerical data collected in our experiments.

2.3.3 Data Analysis Techniques

To address the performance functions (2.12), we conduct experiments on a large set of test problems. To analyze the testing results in hand, the next task is to build a regression model. Basically, there are two ways to develop a model upon which regression analysis can be facilitated. First, deduce a model from theoretical analysis. For instance, based on the simplex tableau process, given m inequalities and n variables, there are in total $m + n$ columns including slack variables. If you suppose that, on average, half of them have negative reduced cost in the initial tableau [83], then the total number of updates, *i.e.*, , the number of iterations, is related to $(m + n)$.

However, sometimes it is hard to get an explicit formula this way - too many uncertain factors involved. Then, there arises the second method, known as *heuristic use of experimentation* [49], which uses experiments to *suggest* hypotheses and then these hypotheses may someday be proven. In such situations, we observe the graphical relationships obtained by experiments, and try developing models on related parameters, based on curve shapes. A similar technique, *Exploratory Data Analysis*, is used in [47] to address the complex joint-effect by multiple factors in nonlinear optimization routines. This kind of *hypothetical* model, which is suggested by experimental analysis, might be proved by theory someday in the future, or it might still not be proved after a long time, as 'our questions about program performance and behavior far outstrip our ability to obtain answers by purely analytical means [68]'.

2.3.4 Multivariate Regression

Statistical methods are highly recommended in empirical science to evaluate the results [49]. After building tentative models, we validate (accept or reject) and revise them using regression analysis. In addition to the values of model coefficients, a regression also returns some statistical parameters providing further information. For example, r^2, the coefficient of determination, shows how good a fit is; meanwhile, a low r^2 can be attributed to big residuals, which suggests the possibility of uncovered parameters in the model; the p value from a two-sided t-test (H_0: the coefficient is zero) helps to exclude irrelevant parameters. We will use these statistical parameters in our analysis below.

We did the multivariate regression analysis in MATLAB by the method of ordinary least squares (function *regstats*). For the prediction tests in Chapter 4 and 5, we used the robust regression method (function $robustfit$) to limit the influence of extreme outliers by assigning reduced weights to them.

2.4 Measuring the Instance

We intend to examine the performance of algorithms for problems with a given size. Therefore, we first need to characterize a problem's size. A lot of analysis on theoretical complexity for linear programming attempts to obtain a suitable measure, and try to make it more relevant to the computational effort than the existing measures [78].

The most straightforward way is by $\overline{m}$ and $\overline{n}$, *i.e.*, , the number of rows (equalities) and columns[2]. Many studies are based on this measurement. However, as mentioned earlier, large-scale problems are usually sparse, and the huge volume of zeros in the coefficients greatly affects the complexity. In practice, LP models usually have fewer than 10 nonzeros per column, independent of the number of rows [62]. In such a case, $\overline{m}$ by $\overline{n}$ cannot appropriately reflect the problem size. The density of nonzeros in the coefficients is an important factor to consider as well.

There is another well-known measurement: the input length, which is the actual number of bits needed to store the data on a computer, and usually denoted by L. For example, the first polynomial time algorithm [53] for linear programming problems achieved a complexity bound of $O(\overline{n}^4 L)$, where $\overline{n}$ is the number of variables, and L is the input length, *i.e.*, the number of 0's and 1's needed to write the problem data in binary form. Later on, Karmarkar's algorithm [51] achieved a better complexity bound of $O(\overline{n}^{3.5} L)$. To some extent, this single parameter, input length, contains the information of the problem dimensions as well as the nonzero density. In addition, it is usually preferable to use only one parameter to specify a problem's size. Therefore, the input length is widely used in complexity analysis. However, sometimes L can be ambiguous too. For example, how to represent a problem greatly affects its input length [83].

Both ($\overline{m}$, $\overline{n}$) and L indicate the overall size of a problem. For linear programs with

[2]The number of columns includes slack and surplus variables when converting inequalities to equalities.

a block-angular structure, more parameters need to be employed to represent the inner structure. Therefore, we use four parameters to measure the test problems (see Section 2.1): m_0 represents the number of linking constraints; h stands for the number of blocks in the system; m_l and n_l are the number of rows and columns within the l^{th} block. In our experiments, both models have evenly sized blocks[3]. This is also true in some other applications. So we denote the subproblem dimensions by $m_{sub} \times n_{sub}$ in general. We also denote $m_0\%$ as the percentage of linking constraints to all the constraints. We will see in the later discussion that this parameter plays an important role for the complexity of these special structured problems.

[3]The stochastic financial model has strictly equal-sized subproblems, while the size of the blocks in the multicommodity network flow model are *approximately* the same ($< 5\%$ variance), and we take the average size of them.

Table 2.3: Multicommodity Network Flow Problem Characteristics (24 out of 216)

Prob.	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
1	196	84	1060	340	24.71%	720	1,754	0.7165%	64	180
2	200	79	1056	335	23.58%	721	1,721	0.7125%	64	180
3	189	85	1063	341	24.93%	722	1,773	0.7201%	64	181
4	194	146	1123	402	36.32%	721	1,982	0.6838%	64	180
5	201	167	1143	423	39.48%	720	2,039	0.6695%	64	180
6	203	157	1135	413	38.02%	722	1,997	0.6697%	64	181
7	520	195	1988	451	43.24%	1,537	3,653	0.5270%	64	384
8	528	207	2002	463	44.71%	1,539	3,684	0.5170%	64	385
9	532	227	2019	483	47.00%	1,536	3,726	0.5022%	64	384
10	537	433	2228	689	62.85%	1,539	4,335	0.4088%	64	385
11	522	410	2205	666	61.56%	1,539	4,288	0.4184%	64	385
12	524	407	2200	663	61.39%	1,537	4,270	0.4190%	64	384
205	822	315	250,851	65,851	0.48%	185,000	439,940	0.0036%	256	723
206	825	316	250,872	65,852	0.48%	185,020	441,570	0.0036%	256	723
207	824	314	250,870	65,850	0.48%	185,020	440,750	0.0036%	256	723
208	836	673	251,309	66,209	1.02%	185,100	519,760	0.0042%	256	723
209	827	685	251,421	66,221	1.03%	185,200	523,080	0.0043%	256	723
210	827	682	251,248	66,218	1.03%	185,030	522,400	0.0043%	256	723
211	2,186	831	426,567	66,367	1.25%	360,200	856,740	0.0036%	256	1,407
212	2,174	880	426,596	66,416	1.33%	360,180	864,590	0.0036%	256	1,407
213	2,143	825	426,411	66,361	1.24%	360,050	857,050	0.0036%	256	1,406
214	2,188	1,769	427,345	67,305	2.63%	360,040	1,010,500	0.0042%	256	1,406
215	2,178	1,772	427,358	67,308	2.63%	360,050	1,011,900	0.0042%	256	1,406
216	2,204	1,802	427,398	67,338	2.68%	360,060	1,015,400	0.0042%	256	1,406

Chapter 3

Algorithms and Notations

In this chapter, detailed algorithms and notations will be presented on both simplex-based and interior-point-based methods, including basic principles of the simplex method and interior point methods (IPM), as well as their counterparts in decomposition: the Dantzig-Wolfe decomposition method (DW) and the analytic center cutting plane method (ACCPM).

3.1 Simplex Based Algorithms

The simplex method was the first method developed to solve linear programs. When we analyze whether or not to use decomposition, the comparison between DW and simplex is one alternative, while the comparison between ACCPM and IPM is another alternative. Algorithms related to the former analysis are briefly introduced in this section, and algorithms related to the latter will be introduced in the next section.

3.1.1 Simplex and Revised Simplex Method

Ignoring its special structure, problem (2.2) to (2.5) can be viewed as a general linear program

$$\begin{array}{lll} \min & c^T x & \\ s.t. & \overline{A}x & = \overline{b} \\ & x & \leq 0, \end{array} \tag{3.1}$$

where $\overline{A} = \begin{bmatrix} A \\ B \end{bmatrix}$ and $\overline{b} = \begin{bmatrix} b \\ d \end{bmatrix}$. Consequently, the dimensions of (3.1) can be represented as $\overline{m} \times \overline{n}$, where $\overline{m} = m_0 + \sum_{l=1}^{h} m_l$, and $\overline{n} = \sum_{l=1}^{h} n_l$. Therefore, when considering the problem as a whole, there are $\overline{m}$ equalities and $\overline{n}$ variables altogether[1].

Linear programs can be converted to the standard form, and then solved by simplex. Working on the tableaus, the simplex method visits the extreme points one by one. At each iteration, one basic feasible solution is replaced by another, and hence a new tableau is formed. This is the so-called *standard simplex method*. For more details, see [84]. Notice that only a small part of the tableau has actually been changed from one iteration to the next, so the *revised simplex method* works directly from the original data, and finds new solutions without any reference to tableaus [16].

Problem (3.1) can be solved by the direct solution approach - simplex. However, the simplex method itself has a number of variants available. It is commonly accepted that the steepest edge is the most efficient pivoting rule to date [82]. Some leading solvers, such as CPLEX, adopt this pivoting rule. Incidentally, the simplex solver in MATLAB employs the smallest subscript rule, which is also known as the least index rule [10].

3.1.2 Dantzig-Wolfe Decomposition Method

The special structure of (2.1) can be exploited by decomposition algorithms, for example, the DW method. Geometrically, any point in the feasible region of a linear program can be represented as a convex combination of the extreme points and extreme rays. Thus, the

[1] Table 2.1 and 2.3 show the actual number columns according to the models. We convert all inequalities to equalities when doing regression analysis, *i.e.*, , $\overline{n}$ includes slack and surplus variables.

original problem can be converted into an equivalent *full* master problem as follows

$$\begin{array}{lrcll} \min\limits_{\alpha_l^i} & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (c_l^T x_l^i) & & & \\ s.t. & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (A_l x_l^i) & = & b & \\ & \sum\limits_{i \in I_l} \alpha_l^i & = & 1, \quad l = 1, 2, \cdots, h & \\ & \alpha_l^i \geq 0, & & i \in I_l \cup J_l, \ l = 1, 2, \cdots, h & \end{array} \tag{3.2}$$

where I_l represents the set of extreme points from subproblem l, and J_l is the set of extreme rays from subproblem l. The full master problem has fewer rows but many more columns than the original problem. In fact, the number of columns in (3.2) is usually astronomical. Fortunately, nobody really works on the full master problem. Instead, DW uses a technique called *column generation* [16], and starts with a subset of I_l and J_l.

Extreme points and extreme rays are generated by subproblems. The l^{th} subproblem can be described as

$$\begin{array}{lrll} \min & (c_l^T - \pi^T A_l) x_l - \mu_l & & \\ s.t. & B_l x_l & = d_l & \\ & x_l & \geq 0, \qquad l = 1, 2, \cdots, h & \end{array} \tag{3.3}$$

where π is the vector of dual variables corresponding to the linking constraints, and π can be obtained as a byproduct when solving the restricted master problem. It will then be passed to the subproblems (3.3). Next, subproblems return proposals (extreme points or extreme rays) to the restricted master problem. A new iteration begins again.

3.2 Interior Point Based Algorithms

In this section, we briefly introduce the principles of interior point methods (IPM) and the analytic center cutting plane method (ACCPM).

3.2.1 Introduction to Interior Point Method

In mid 1980s, IPM was proposed by Karmarkar [51] as a new solution method. It was considered a significant improvement in the optimization field - there was a front-page

New York Times article titled "Breakthrough in Problem Solving" on November 19, 1984. As implied by its name, IPM finds an optimal solution by moving through the interior of the feasible region. Even in the worst-case, IPM still has a polynomial time complexity.

In fact, the idea of moving inside the feasible region dates back to work in 1955 [31], and was also proposed by several other researchers simultaneously [82]. It was shown in [82] that Karmarkar's search direction is equivalent to projected Newton barrier methods with an appropriate choice of parameters. Nowadays IPM applications are quite different from Karmarkar's original projective method, but the key concept remains highly valuable [82].

There are three main types of IPM: the affine scaling algorithm, the potential reduction algorithm, and the path following algorithm. The last one, the path following algorithm, has the best known time complexity and good performance in practice [5]. This method has been widely employed in commercial IPM solvers, especially for large scale problems. Moreover, the concept of analytic center is also derived from this method.

There are three crucial building blocks for path following methods [62]: First, convert constrained problems to unconstrained ones by Lagrange's method; second, eliminate the nonnegativities ($x \geq 0$) by Fiacco and McCormick's barrier method; then, solve the unconstrained problem by Newton's method. Consider a problem with barrier terms

$$\begin{aligned} \min \quad & c^T x - u \sum_{j=1}^{\overline{n}} \log x_j \\ s.t. \quad & \overline{A}x = \overline{b} \end{aligned} \tag{3.4}$$

and its dual problem

$$\begin{aligned} \max \quad & \overline{b}^T y + u \sum_{j=1}^{\overline{n}} \log s_j \\ s.t. \quad & \overline{A}^T y + s = c \end{aligned} \tag{3.5}$$

where u is a positive barrier parameter. Given u, the solution $x(u)$ is a function of u. According to Karush-Kuhn-Tucker (KKT) conditions, the necessary and sufficient conditions for the point x(u) to be an optimal solution of problem (3.4) and (3.5) are the following

system of equations (due to duplication):

$$\begin{aligned} \overline{A}x(u) &= \overline{b} \\ \overline{A}^T y(u) + s(u) &= c \\ X(u)S(u)e &= eu, \end{aligned} \tag{3.6}$$

where $x(u)$, $y(u)$, $X(u)$, and $S(u)$ are functions of u, e is a vector with all ones, and X and S are square diagonal matrices with diagonal elements of x and s.

The path following algorithms solve the system of nonlinear equations (3.6) by Newton's method. At each iteration, we reduce the value of u. Fiacco and McCormick [27] have proved that $x(u) \to x^*$ as $u \to 0$. Incidentally, when $u \to \infty$, the minimizer (3.4) becomes

$$\begin{aligned} \min \quad & -\sum_{j=1}^{\overline{n}} \log x_j \\ s.t. \quad & \overline{A}x = \overline{b} \end{aligned} \tag{3.7}$$

The optimal solution to problem (3.7) corresponds to the analytic center of the feasible set [5]. The definition to the analytic center will be discussed with more details in the next section.

3.2.2 Non-differentiable Optimization

Although it can be applied to linear programming, as one of the cutting plane methods, ACCPM originated from solving non-differentiable optimization (NDO) problems. To fully understand ACCPM, we first recall a generic cutting plane algorithm. Consider the following non-differentiable problem [25]

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0, \end{aligned} \tag{3.8}$$

where f and g are real-valued, continuous, nondifferentiable, and convex functions. Convexity implies that there is at least one supporting hyperplane to f at every point x_0. The equation for the supporting hyperplane is given by

$$y = f(x_0) + \xi_0^f (x - x_0), \tag{3.9}$$

where $\xi_0^f(x - x_0)$ is one of the subdifferential set $\partial f(x)$ of f at x_0. For ease of notation, we assume (only in this section) that all subgradients are row vectors. Since a supporting hyperplane gives an underestimate of f, the subgradient inequality

$$f(x_0) + \xi_0^f(x - x_0) \leq f(x) \tag{3.10}$$

can be used to approximate f by the maximum of a set of piecewise linear functions. Therefore, given a set of points x_i, $i \in I_f$ and their corresponding subgradients ξ_i^f, f can be tangentially approximated by

$$\bar{f}(x) = \max_{i \in I_f} \{f(x_i) + \xi_i^f(x - x_i)\} \tag{3.11}$$

Equation (3.11) implies that $\bar{f}(x) \leq f(x)$ for any index set I_f. Larger sets will give better approximations. The same techniques can be applied to g, and hence g can be approximated by

$$\bar{g}(x) = \max_{j \in J_g} \{g(x_j) + \xi_j^g(x - x_j)\}, \tag{3.12}$$

where ξ_j^g is a subgradient of g at point x_j, $j \in J_g$. Thus, problem (3.8) can be approximated by

$$\begin{aligned} \min \quad & \max_{i \in I_f} \{f(x_i) + \xi_i^f(x - x_i)\} \\ \text{s.t.} \quad & \max_{j \in J_g} \{g(x_j) + \xi_j^g(x - x_j)\} \leq 0, \end{aligned} \tag{3.13}$$

which is equivalent to

$$\begin{aligned} \min \quad & \gamma \\ \text{s.t.} \quad & f(x_i) + \xi_i^f(x - x_i) \leq \gamma, \quad \forall i \in I_f \\ & g(x_j) + \xi_j^g(x - x_j) \leq 0, \quad \forall j \in J_g \end{aligned} \tag{3.14}$$

Problem (3.14) becomes a linear program that is easier to solve than the original one. However, this is only an approximation of the original problem (3.8), and will get better as more constraints are added. The nondifferentiability is eliminated at the cost of having a large number of constraints. To deal with this problem, cutting plane methods only use a subset of the constraints and generate the rest as needed. In fact, they solve series of *relaxed* master problems with the similar form of (3.14). The master problem at the k^{th} iteration is a relaxation of (3.8) because:

1. by convexity of g, $\{x : \max_{j \in J_g}\{g(x_j) + \xi_j^g(x - x_j)\} \leq 0\}$ contains $\{x : g(x) \leq 0\}$;

2. by convexity of f, $\max_{i \in I_f}\{f(x_i) + \xi_i^f(x - x_i)\} \leq f(x)$ for all $x \in \{x : g(x) \leq 0\}$.

This relaxation gets tighter when more points are added. In addition, for the cutting plane method, any feasible point gives an upper bound, and a lower bound can be given either by the optimal solution of the relaxed master problem, or by evaluating the dual problem. As for the stopping criterion, the algorithm will stop if the duality gap, which is the difference between the current best upper and lower bounds, drops below a certain pre-determined threshold. For more details, see [25] and [26].

3.2.3 ACCPM for Block-Angular Structured Problems

Cutting plane methods were proposed independently by Kelley [52], Cheney and Goldstein [15] as a solution approach for non-differentiable optimization. They solve constrained convex problems by approximating convex functions with the subgradients. They have stable numerical properties, and can be applied to linear programming as well as integer programming. There are several variants for cutting plane methods. For details, see [26].

The core difficulty with cutting plane methods is to calculate the centers of polyhedrons. For example, finding the gravity center itself can be as expensive as solving the original problem [25]. Therefore, an *analytic center* was defined [79][80], and the corresponding cutting plane method is called Analytic Center Cutting Plane Method (ACCPM).

When applied to the decomposition area [34], ACCPM can be viewed as the dual equivalent of the Dantzig-Wolfe decomposition (DW) method. Rather than passing *marginal prices* as DW method does, ACCPM passes *central prices* from the master problem to the oracles (subproblems). This property enables ACCPM to achieve a better convergence rate because a *center price* contains more information of all the cuts (proposals) accumulated so far [35].

Recall a problem with a block-angular structure discussed in section 2. After we intro-

duce artificial variables to the linking constraints, the restricted master problem becomes

$$\begin{array}{lll} \min\limits_{\alpha_l^i} & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (c_l^T x_l^i) + M_1^T x^- + M_2^T x^+ & \\ s.t. & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (A x_l^i) - x^- + x^+ = b & \\ & \sum\limits_{i \in I_l} \alpha_l^i = 1, & l = 1, 2, \cdots, h \\ & \alpha_l^i \geq 0, x^+ \geq 0, x^- \geq 0, & i \in I_l \cup J_l,\ l = 1, 2, \cdots, h \end{array} \tag{3.15}$$

where M_1 and M_2 are both $m_0 \times 1$ vectors.

Adding a proposal in the primal space is equivalent to adding a cut to the dual space. The dual problem of (3.15) is

$$\max \quad b^T \pi + \sum_{l=1}^{h} \mu_l \tag{3.16}$$

$$s.t. \quad (A_l x_l^i)^T \pi + \mu_l \leq c_l^T x_l^i, \qquad i \in I_l, l = 1, 2, \cdots, h \tag{3.17}$$

$$(A_l x_l^j)^T \pi \leq c_l^T x_l^j, \qquad j \in J_l, l = 1, 2, \cdots, h \tag{3.18}$$

$$-M_1 \leq \pi \leq M_2 \tag{3.19}$$

The penalty parameters M_1 and M_2 in the primal master problem (3.15) become box constraints (3.20) in the dual master problem. These box constraints, together with a lower bound (LB), make the ACCPM localization set (refer to (3.20)) bounded. An extreme point in the primal space corresponds to an optimality cut like (3.17) in the dual space, and an extreme ray to a feasibility cut like (3.18). Consider the following polyhedron

$$\begin{array}{ll} F = & \left\{(\pi, \mu) : b^T \pi + \mu \geq LB,\ \mu \leq c_l^T x_l^i - (A_l x_l^i)^T \pi,\ 0 \leq c_l^T x_l^j - (A_l x_l^j) \pi,\ -M_1 \leq \pi \leq M_2 \right\} \\ & i \in I_l,\ j \in J_l,\ l = 1, 2 \cdots h. \end{array} \tag{3.20}$$

Clearly, F contains the optimal solution of system (3.16) to (3.19). F is defined as the *localization set*. The pair (π, μ), *i.e.*, , the dual price vectors π and μ, will be sent to the subproblems.

For ease of notation, we introduce the following symbols

$$\begin{array}{lll} s_0 & = b^T\pi + \mu - LB & \\ s_l^i & = c_l^T x_l^i - (A_l x_l^i)^T \pi - \mu, & \text{if } i \in I_l,\ l = 1, 2 \cdots h \\ s_l^i & = c_l^T x_l^i - (A_l x_l^i)^T \pi, & \text{if } i \in J_l,\ l = 1, 2 \cdots h \\ \sigma_j^+ & = \pi_j + M_1^j, & j = 1, 2 \cdots m_0 \\ \sigma_j^- & = M_2^j - \pi_j, & j = 1, 2 \cdots m_0 \end{array} \tag{3.21}$$

If $I \neq \phi$, which, together with the box constraints $-M_1 \leq \pi \leq M_2$, ensures that F is bounded. We assume that the interior of F is not empty. The analytic center (π, μ) is then defined as the maximizer of

$$\varphi = \ln s_0 + \sum_{l=1}^{h} \sum_{i \in I_l \cup J_l} \ln s_l^i + \sum_{j=1}^{m_0} \ln \left(\sigma_j^+ \sigma_j^-\right) \tag{3.22}$$

3.3 Comparison of Simplex and IPM

Both simplex and IPM are still active in application, and competitive with each other. Currently they act as the two main solution methods for linear programming. Table 3.1 compares simplex and IPM[2] in terms of several criteria[3].

There has been an intense competition between simplex and interior point methods for a long time, but there is still no clear conclusion about which is the winner. Nemhauser argues that the best simplex algorithms are now competitive with the best IPM algorithms [74]. In Chapter 6, we will present a hybrid decomposition method that combines ACCPM and DW, which are the counterparts of IPM and simplex, respectively.

[2]Here the interior point method refers to the primal-dual path following method.

[3]This table is mostly from [50]. Some other papers are also referenced, including [7], [62], [32], [17], and [39].

Table 3.1: Comparison of Simplex and IPM

	Simplex	IPM
Geometrical Characteristics	Jumping from vertex to vertex on the boundary of the feasible set	In the interior of the feasible region
Degeneracy	The generated optimal basic solution is not strictly complementary	Converge to the analytic center and produces a strictly complementary pair of solutions
Computational Effort	No dominating step; Require many iterations, each of which is very fast	Cholesky factorization usually dominates; A small number of iterations, and the number of iterations grows slowly with problem size
Worst-case Complexity	Exponential	Polynomial
Initialization	Two phases; Take advantage if from an advanced starting point	Require a positive vector to start with
Warm Start	Easy to re-start	Not efficient in re-starting
Sensitivity Analysis	Strong	Weak

Chapter 4

With and Without Decomposition - DW vs. Simplex

The famous Klee-Minty [54] example has shown that in the worst case, the simplex method could visit every vertex before attaining optimality, which means an exponential complexity bound. In fact, it has been found that for any method that seems useful to some problems, other problems can be constructed to make that method very unsatisfactory [56]. However, the worst case of simplex almost never happens in solving real-world problems. In this chapter, we briefly review the studies on the efficiency of simplex-based algorithms, and compare them with the results of our experiments. This is far from a comprehensive literature review, instead, we just try to give the reader a picture about how simplex has been observed to behave in practice.

For each conclusion drawn from *general* problems, we provide some numerical results to test if it still holds for problems with a block-angular structure. We investigate the complexity of both simplex and DW, and then compare their relative efficiency, which is defined as the CPU time of DW over the CPU time of simplex.

4.1 Average Behavior of Simplex

Both the simplex method and the Dantzig-Wolfe decomposition method are iterative processes. Therefore, the complexity can be factored as the number of iterations times the

computational effort needed for each iteration.

4.1.1 Bounds for Number of Iterations

It is well known that although simplex has an exponential complexity bound, its practical behavior is much better. Reported in [21], with $\overline{m} < 50$ and $\overline{n} < 200$, the number of iterations is usually less than $3\overline{m}/2$ and only rarely going to $3\overline{m}$. These empirical findings were obtained very early, but later studies on larger problems are in striking agreement with them [16]. Bixby [6] tested CPLEX 1.0 on Netlib problems, 80% of which require less than $3\overline{m}$ iterations[1]. This behavior is summarized in [82] as 'the remarkable fact' that the primal simplex method typically requires a number of iterations between 2 and 3 times the number of constraints.

We now examine if this conclusion still holds for problems with a block-angular structure. Figure 4.1 shows the number of iterations[2] vs. the number of rows. The dotted lines in the figure are 'two times' and 'three times' the number of rows, respectively. Figure 4.1(a) reveals the number of iterations over the number of rows on the stochastic financial model, and Figure 4.1(b) is for the multicommodity network flow model[3].

More statistical data are summarized in Table 4.1, where f stands for the stochastic financial problems, $m(with)$ stands for the multicommodity network flow problems with individual capacity constraints counted as the number of rows, and $m(without)$ stands for multicommodity problems with individual capacity constraints treated as upper bounds. For the stochastic financial model (solved by MATLAB), since five problems did not converge due to degeneracy, we only collected results for the remaining 95 test problems. There are 79 out of 95 (83.16%) problems that required less than or equal to $3\overline{m}$ iterations, 61 out of 95 (64.21%) problems are less than or equal to $2\overline{m}$, with a mean 1.96, a maximum 5.90, and a minimum 0.39 times $\overline{m}$. For the multicommodity model (solved by CPLEX), if the individual capacity constraints are treated as upper bounds: 126 out of 216 (58.33%)

[1]The remaining problems are 'unbalanced' with $\overline{n} >> \overline{m}$, and the ratio of iterations to rows can be as big as 469.1. If taking the upper and lower bounds into consideration, the values down to 1.2 and under.

[2]In this thesis, the number of iterations refers to the *total* number of iterations including Phase I and Phase II for both simplex and DW.

[3]For the multicommodity model, the individual capacity for each arc is treated as upper bound in the code, but counted among the number of rows in this figure.

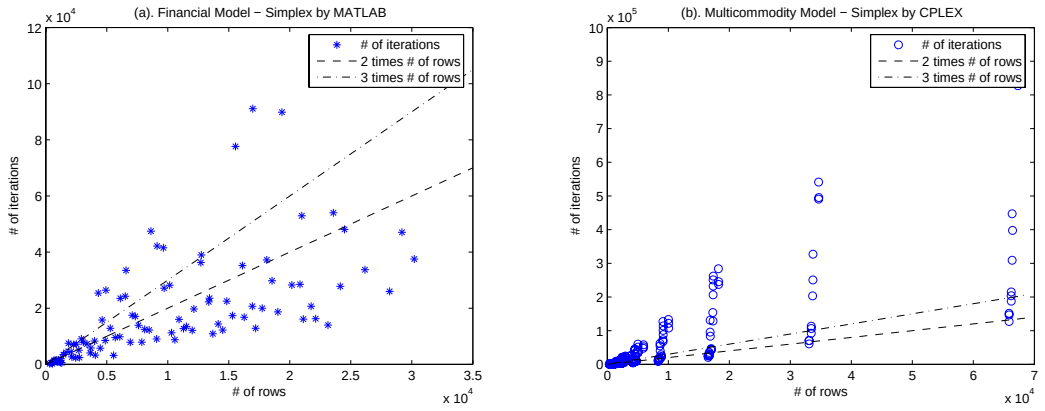


Figure 4.1: Simplex Number of Iterations vs. Number of Rows

problems converged less than $3\overline{m}$ iterations, 95 out of 216 (43.98%) problems are less than $2\overline{m}$, with a mean 4.17, a maximum 15.62, and a minimum 0.78 times $\overline{m}$. If the individual capacity constraints are counted as the number of rows: there are 216 out of 216 (100%) problems that required less than $3\overline{m}$ iterations, 201 out of 216 (93.06%) problems are less than $2\overline{m}$, with a mean 0.83, a maximum 2.48, and a minimum 0.24 times $\overline{m}$.

Table 4.1: Simplex Number of Iterations

Problem	Total # of problems	# of problems with iterations $\leq 3\overline{m}$	(%)	$\leq 2\overline{m}$	(%)	iterations / rows mean	max	min
f	95	79	83.16%	61	64.21%	1.96	5.90	0.39
$m(without)$	216	126	58.33%	95	43.98%	4.17	15.62	0.78
$m(with)$	216	216	100%	201	93.06%	0.83	2.48	0.24

Notice that Figure 4.1(b) shows a vertical distribution pattern, which implies that the number of iterations required to converge varies greatly even on problems with the same number of rows. This vertical pattern does not seem obvious in Figure 4.1(a) because for the multicommodity model, several same sized problems are generated with different density, while for the stochastic model, there are no two models with exactly the same overall dimensions but different inner structures.

We use regression analysis to further examine the dependence of the number of simplex

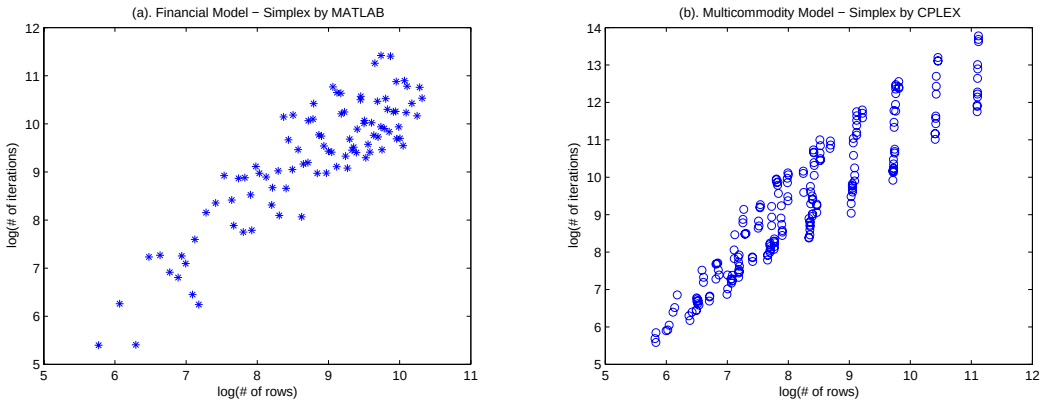


Figure 4.2: Simplex log(number of iterations) vs. log($\overline{m}$)

iterations on problem characteristics. Taking logarithms of both the horizontal and vertical axes in Figure 4.1 can help to disperse the points that are clustered near the origin - see Figure 4.2. The linear trends in Figure 4.2 suggest the regression model (4.1) for the relation between simplex iteration counts and the number of rows. The regression results are summarized in Table 4.2 on page 58 for both the financial and multicommodity[4] problems. The p values reveal the significance levels of the variables. A low value of p means that the corresponding independent variable has a low probability to be zero, and hence is statistically significant. We take 5% as the significance level in our analysis, i.e., if $p > 0.05$, the corresponding variable is considered insignificant. Notice that we keep a constant in a model as usual, even though the p value is big.

$$iter = e^{\alpha}\overline{m}^{\beta} \tag{4.1}$$

After taking logarithms, our experiments, as shown in Figure 4.2 (a) and (b), show linear trends for both test problems, but are somewhat scattered. Furthermore, the vertical cluster pattern appears again in Figure 4.2. All the phenomena imply that for problems with a block-angular structure, the number of rows and columns is not enough information to estimate the performance. This is easy to understand: with the same $\overline{m}$ and $\overline{n}$, the

[4]For all the regression results in this thesis, the multicommodity problems' individual capacity constraints are treated as upper bounds, and hence not counted among the number of rows.

inner structure can be different. Therefore, for special-structured problems, we introduce more parameters to examine their computational effort[5].

$$iter = e^{\alpha}\overline{m}^{\beta}(m_0\%)^{\gamma} \tag{4.2}$$

In model (4.2), the percentage of linking constraints ($m_0\%$) is employed so that the inner structure can be embodied to some extent. Regression results in Table 4.2 show that model (4.2) is a better fit than model (4.1) as the r^2 for both test problems are improved. The very low value of the p value also suggests that inner structure indicator, $m_0\%$, is indispensable. In the same vein, we can try another variable, $nz\%$, in the regression model (4.3).

$$iter = e^{\alpha}\overline{m}^{\beta}(nz\%)^{\gamma} \tag{4.3}$$

Regression results in Table 4.2 show that adding $nz\%$ can improve the fit on test sets f and m. Intuitively, both $m_0\%$ and $nz\%$ can indicate the density of nonzero elements in the problem matrix. For model (4.1), adding $m_0\%$ seems a better choice than adding $nz\%$. We checked the F-values for the entry of $m_0\%$ and $nz\%$, and all of them made statistically significant (95%) improvements on r^2.

The earlier studies that we have reviewed suggest that the number of iterations depends more on $\overline{m}$ than $\overline{n}$. In such cases, even if $\overline{n}$ appears, it is just a complementary condition, e.g., in the 'unbalanced' situation. To address the minor influence by $\overline{n}$, [16] indicates that for a fixed number of rows, the typical number of iterations increases with the logarithm of $\overline{n}$.

Besides the row-oriented observations, other researchers estimate the simplex iteration counts in relation to the number of columns. Vanderbei [83] used a heuristic statistical method to relate the number of simplex iterations to the number of columns, given that degeneracy does not arise. The reason is that from the perspective of simplex dynamics, if assuming half of the variables in the initial tableau are with negative reduced cost (for minimization problem), then half of the variables amount of tableau updates will be required. Therefore, the number of *columns* can be used to estimate the number of

[5]The correlation matrices among the dimensional parameters for both test models are provided in Appendix D.

iterations. By taking logarithms, a nice linear model in [83] was obtained based on 69 problems. Similar to the analysis in [83] (pp. 185-190), we build a regression model

$$iter = e^{\alpha}\overline{n}^{\beta} \tag{4.4}$$

Again, adding inner-structure variables $m_0\%$ and $nz\%$ respectively in model (4.4) leads to better fits (see Table 4.2).

$$iter = e^{\alpha}\overline{n}^{\beta}(m_0\%)^{\gamma} \tag{4.5}$$

$$iter = e^{\alpha}\overline{n}^{\beta}(nz\%)^{\gamma} \tag{4.6}$$

Table 4.2 summarizes the results for all the regression models in this section. The first three models, (4.1) to (4.3), estimate *iter* based on $\overline{m}$, and models (4.4) to (4.6) are based on $\overline{n}$. The two sets are analogous to each other: adding $m_0\%$ or $nz\%$ can evidently improve the r^2, which embodies the fitting level of a regression model, but the difference between adding $m_0\%$ and adding $nz\%$ is not big. Coefficient α corresponds to the constant in the regression models. Coefficient β takes positive values, implying that the number of iterations required by simplex increases with the total number of rows or columns, which is consistent with what we expected. Similarly, coefficient γ takes positive values, which means that *iter* increases with $m_0\%$ or $nz\%$. Our results also show that for the test problems, the column-oriented regression models are slightly better fits than the row-oriented ones.

Since for block-angular structured problems, the inner structure matters, we can further try using the four inner structure parameters, m_0, h, m_{sub}, and n_{sub}, to build a regression model

$$iter = e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon} \tag{4.7}$$

These four basic parameters can fully depict a block-angular structured problem. Other parameters such as $m_0\%$, $nz\%$, $\overline{m}$, and $\overline{n}$ can be derived from them. Model (4.7) has the highest r^2 value in all regression results in Table 4.2. For the financial problems, although the p value suggests that β (for m_0) is 22.52% likely to be redundant, we still keep the

variable in order to be consistent with discussions about DW decomposition. In fact, it is quite surprising to see that m_0 is insignificant in the regression model, as it seems an important parameter for block-angular structured problems and our above discussion has just shown that adding $m_0\%$ is a plus to improve fittings. This conflict can probably be explained by the actual problem characteristics. Recall Tables 2.1 and 2.3 in Section 2.2. The financial problems contain very low $m_0\%$ (up to 8.54%), so that we can say that all of them are sparse in terms of the percentage of linking constraints. In such low levels of $m_0\%$, the linking constraints did not make the problems more difficult to solve, and parameter β even takes a negative value. Conversely, the multicommodity problems have a larger range of $m_0\%$ (from 2.13% to 62.85%). As a result, variable m_0 is significant in the regression model (4.7).

For real-world problems, the number of rows and the number of columns are usually related to their economic interpretations, and so are the number of rows and columns in each block. For example, the block dimensions of the stochastic model (dual formulation; primal block-angular structure) are $(2h + n + 1 \times (h + 1))$, where h represents the number of scenarios and n is the number of assets. Due to this kind of relation, researchers usually use only one of them, rows or columns, to analyze algorithms' performance. However, variables m_{sub} and n_{sub} in the regression model (4.7) are both significant. Moreover, one of each even takes a negative value on test problems f and m, respectively. We have not been able to find a plausible explanation for this phenomenon yet.

Readers may have noticed that the regression analysis is done separately on the two sets of test problems (f and m in Table 4.2), and the results are quite *model dependent*. According to the coefficients of determination, r^2, equations (4.1) to (4.6) are better fits for the multicommodity problems than the financial problems. Coefficients α, β, γ, δ, and ϵ take different values on problem f and m, but suggest a similar tendency.

4.1.2 Number of Arithmetic Operations at Each Iteration

Generally speaking, the number of iterations as a criterion is not enough to assess the efficiency of algorithms. One example is the different pivoting rules of simplex. Although it usually requires fewer iterations by the largest increase rule than by the largest coefficient rule, the former takes more time to execute [16]. In this light, the computational effort at

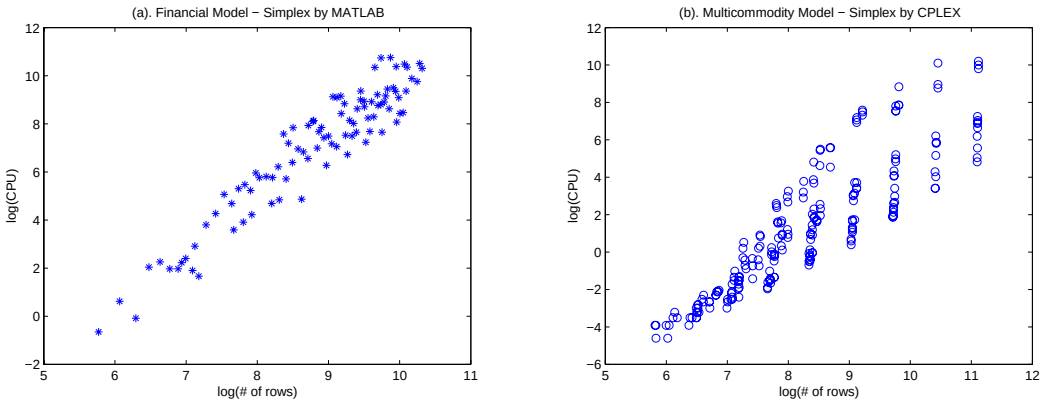


Figure 4.3: Simplex log(CPU time) vs. log($\overline{m}$)

each iteration also counts.

From an implementation point of view, Chvatal in [16] estimates that given $\overline{n} \geq \overline{m}$, the standard simplex method requires $\overline{mn}/4$ arithmetic operations per iteration on average, and for the revised simplex method, the figure is $32\overline{m} + 10\overline{n}$ per iteration. He further concludes that on large sparse problems, the revised simplex method usually takes less time than the standard simplex method for an iteration.

The per iteration computing time is influenced not only by the nonzero density of a problem, but also by computer construction, e.g., whether data are stored in memory or in peripheral devices matters greatly. Nevertheless, the total computing time is the most realistic criterion for choosing between algorithms. Therefore, we skip this part of per-iteration analysis, and move on to investigate the overall complexity of simplex.

4.1.3 Average Complexity of Simplex

Ideally, the overall complexity equals the number of iterations times the effort for each iteration. However, it is almost impossible to obtain a rigorous theoretical average estimation for simplex. In practice, CPU time is used to simulate the average complexity. We also conducted experiments over a large set of test problems to observe the average behavior of simplex.

Figure 4.3 shows the simplex CPU time vs. the number of rows on both test prob-

lem sets (taking logarithms on both axes). Similar to Section 4.1.1, we see the scattered linear distribution patterns again. We then build the following regression models to further address the simplex average complexity as a function of the problems' dimensional parameters.

$$CPU = e^{\alpha}\overline{m}^{\beta}(m_0\%)^{\gamma} \tag{4.8}$$

$$CPU = e^{\alpha}\overline{m}^{\beta}(nz\%)^{\gamma} \tag{4.9}$$

Models (4.8) and (4.9) are row-oriented simplex CPU estimations, with $m_0\%$ and $nz\%$ respectively to further address problems' inner structures. Accordingly, models (4.10) and (4.11) are column-oriented simplex CPU estimations. All the regression results in this section are summarized in Table 4.3. Comparing (4.8) with (4.10), and (4.9) with (4.11), we can conclude that the column-oriented models have better fits than the row-oriented models.

$$CPU = e^{\alpha}\overline{n}^{\beta}(m_0\%)^{\gamma} \tag{4.10}$$

$$CPU = e^{\alpha}\overline{n}^{\beta}(nz\%)^{\gamma} \tag{4.11}$$

Figure 4.4 illustrates how simplex CPU time varies by the number of columns and the percentage of linking constraints. The mesh plane in the figure is based on the regression results of (4.10), and the dots in the figures represent the test problems. Only a few test problems have high percentage of linking constraints, so the upper part of the plane is blank.

Model (4.12) is completely based on the four basic inner-structure variables. According to the results in Table 4.3, the r^2 wins over all other models. However, some high p values occur in this model. For test set f, besides the constant, m_0 becomes insignificant again. For test set m, dimensions of subproblems seem redundant here. When looking at the problem formulation, we see that the number of arcs for each commodity is almost fixed, which means that the size of each block is the about same. In this sense, the overall size of the problems as well as the simplex CPU time probably mainly depends on h, the number

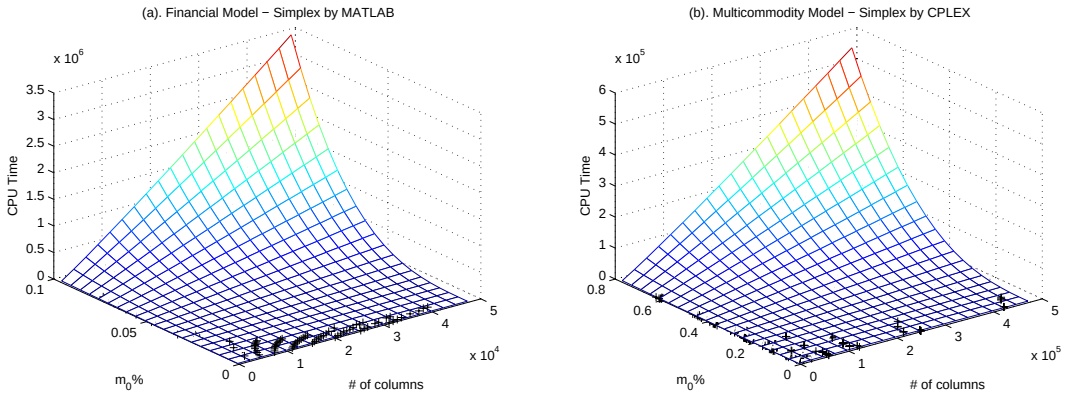


Figure 4.4: Simplex CPU Time vs. $\overline{n}$ and $m_0\%$

of blocks. But, it is still unclear to us why m_{sub} and n_{sub} appear significant in model (4.7) in Table 4.2.

$$CPU = e^{\alpha} m_0^{\beta} h^{\gamma} m_{sub}^{\delta} n_{sub}^{\epsilon}, \tag{4.12}$$

More than half of the models in Table 4.3 achieve r^2s with 0.9 and higher values, which implies that on problems with a block-angular structure, it is a good way to measure simplex CPU time by the overall size ($\overline{m}$ or $\overline{n}$ together with a density indicator ($m_0\%$ or $nz\%$). More intuitively in 3D plots, Figure 4.4 shows the relation of simplex CPU time to the number of columns and the percentage of linking constraints. Notice that only a small portion of the test problems have a high percentage of linking constraints, which is a nice feature for decomposition algorithms that we will discuss next.

4.2 Average Behavior of DW

The task to estimate the average complexity of DW in a explicit form is more difficult than simplex. Although it is guaranteed to take a finite number of iterations to converge, few theoretical bounds have been derived [1].

4.2.1 Number of Outer Iterations

Following the thinking thread of simplex, we can analyze the average complexity of DW from a tableau's perspective. Assume there are h subproblems, and each has $C_{n_{sub}}^{m_{sub}}$ basic solutions at most. Thus, the full master problem has up to $hC_{n_{sub}}^{m_{sub}}$ possible columns, $\frac{1}{n_{sub}}2^{n_{sub}} \leq C_{n_{sub}}^{m_{sub}} \leq 2^{n_{sub}}$, and this number reaches a maximum when $m_{sub} = n_{sub}/2$ [83]. Taking an average of the upper and lower bounds, and assuming there are half of the columns with negative reduced cost [83] and h proposals are returned from subproblems, DW requires

$$\frac{2^{n_{sub}}/n_{sub} + 2^{n_{sub}}}{4} \tag{4.13}$$

outer iterations on average. Apparently, this deductive value is exponential, but is most likely too big according to experience. Indeed, although DW can be understood as a large simplex tableau, the calculation of reduced cost is done separately by subproblems, rather than by considering all the columns in the full master problem. The technique, column generation, is the key of the DW method. Therefore, the estimation (4.13) is not realistic at all.

Similar to the aforementioned average behavior of simplex, we can assume that the outer iterations required in DW to reach an optimal solution is $const * (m_0 + h)$, i.e., a constant value times the number of rows in the master problem. In the simplex counterpart, this constant has been observed to be $2 \sim 3$, consequently, it should be $2/h \sim 3/h$ for DW given h proposals are returned at each iteration. However, from a tableau perspective, the *full* master problem is extremely 'unbalanced', in terms of the number of rows versus the number of columns. In this sense, the value of *const* should be bigger than $2 \sim 3$. We then examine this constant in a form of $const * (m_0 + h)/h$, from numerical experiments.

Figure 4.5 shows the number of DW outer iterations vs. the number of constraints $(m_0 + h)$ in the master problem. For the financial problems in Figure 4.5 (a), the '$2 \sim 3$ times number of rows' rule seems to hold. The multicommodity problems in Figure 4.5 (b) appear to need more iterations to converge as quite a few points exceed the $3(m_0 + h)/h$ line.

Next, we try to build regression models to address the DW convergence properties. Usually, we test a theoretical conclusion by numerical results. In a complex case such as the DW decomposition algorithm, there is no explicit form to describe its complexity, so

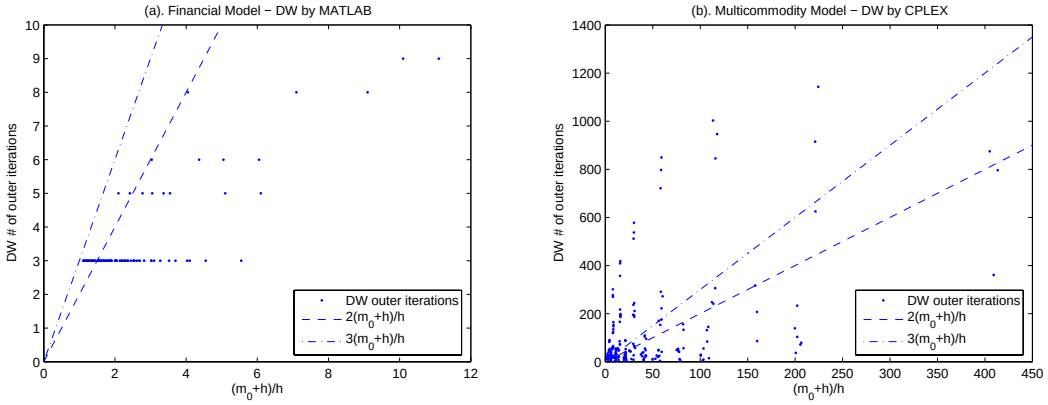


Figure 4.5: DW Number of Outer Iterations vs. Number of Linking Constraints

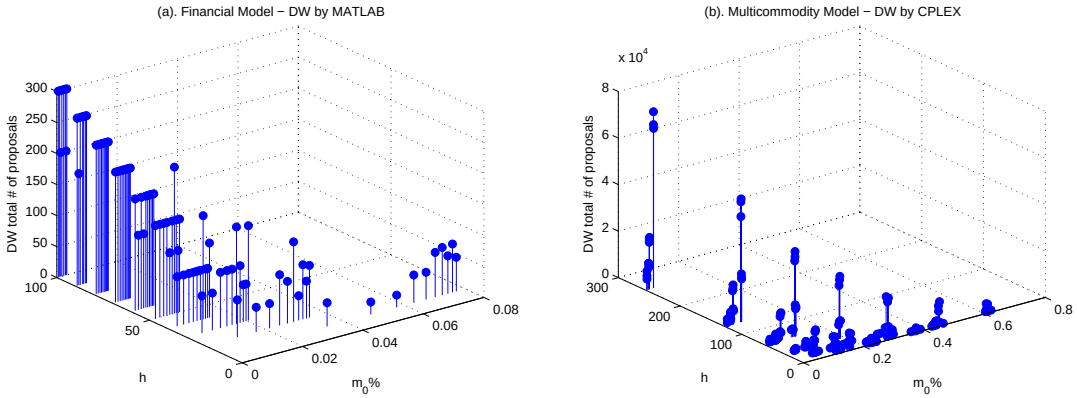


Figure 4.6: DW Number of Proposals vs. $m_0\%$, h

we use the heuristic method [49] to suggest a model and examine the joint effect of multiple parameters [47]. For example, Figure 4.6 gives us helpful hints on how the number of DW outer iterations increases with the percentage of linking constraints and the number of blocks.

If DW is viewed as a big simplex tableau, the total number of proposals in DW corresponds to the number of iterations in simplex. Consequently, the number of DW outer iterations equals the number of proposals divided by the number of blocks, given that each subproblem returns one proposal per iteration. However, this assumption is not true because after a few iterations, some of the subproblems will have nonnegative objective values and stop returning proposals to the master problem. We actually built some regression models to predict the number of DW outer iterations, but obtained poor results. Therefore, the total number of proposals is used in our regression analysis as the indicator of DW convergence properties.

If the number of proposals required in DW is to some extent related to the number of rows in the master problem, we suggest that $m_0\%$ and $nz\%$ may provide valuable complementary information for the test problems with a block-angular structure. Equations (4.14) and (4.15) are built for DW as row-oriented regression models, where $(m_0 + h)$ is the number of rows in the master problem, and $\#prop$ stands for the total number of proposals required to achieve optimality.

$$\#prop = e^{\alpha}(m_0 + h)^{\beta}(m_0\%)^{\gamma} \tag{4.14}$$

$$\#prop = e^{\alpha}(m_0 + h)^{\beta}(nz\%)^{\gamma} \tag{4.15}$$

Equations (4.16) and (4.17) are column-oriented regression models. Notice that $\overline{n}$ is the number of columns in the original problem, rather than the columns in the DW master problem, which is $\#prop$ itself. When DW is understood as a big simplex tableau, $\overline{n}$ seems reasonable to use to estimate the number of tableau updates, which is also the number of proposals required in DW.

$$\#prop = e^{\alpha}\overline{n}^{\beta}(m_0\%)^{\gamma} \tag{4.16}$$

$$\#prop = e^{\alpha}\overline{n}^{\beta}(nz\%)^{\gamma} \tag{4.17}$$

Model (4.18) is developed based on the four inner structure variables. Regression results in this section are all summarized in Table 4.4. For the row-oriented models (4.14) and (4.15), the r^2s are less than 0.9, but variables are all significant. For the column-oriented models, ($4.16m$) reaches a good r^2, ($4.17m$) is slightly worse than ($4.15m$), and the variable $m_0\%$ and $nz\%$ appear redundant in both ($4.16f$) and ($4.17f$). Models ($4.14f$) to ($4.17f$) suggest consistent conclusions: the number of DW outer iterations increases with problems' size but decreases with density.

Among all the models in Table 4.4, model (4.18) has the highest r^2 value, but also with severely high p values. Parameter β takes positive values on both f and m, which suggests that the more linking constraints, the more outer iterations required in DW. Parameter γ is positive in ($4.18m$), which implies that $\#prop$ increases with h. However, the negative γ in ($4.18f$) is hard to explain: a higher h value usually leads to a bigger problem, but why is $\#prop$ less? These kinds of unexpected numerical results can probably be explained by the complex joint-effect of multiple variables. As discussed earlier, variables in real-world applications sometimes relate to each other. Therefore, for the overlapping part between variables, a regression analysis takes the one that results in higher r^2. Notice that n_{sub} is significant in ($4.18f$) but not in ($4.18m$), while β and γ are significant in ($4.18m$) but not in ($4.18f$).

$$\#prop = e^{\alpha} m_0^{\beta} h^{\gamma} m_{sub}^{\delta} n_{sub}^{\epsilon} \tag{4.18}$$

4.2.2 Complexity at Each Iteration

As a decomposition algorithm, DW involves more factors that influence its complexity properties than simplex, so the analysis becomes more difficult. In a parallel computing environment, the per iteration complexity of DW is measured by the sum of the effort of solving the master problem and the maximum effort of subproblems. The work of solving the master problem and subproblems is alternate, and seldom overlaps [43]. Notice that we simulate an ideal parallel computing, i.e., factors such as overhead incurred each time, and the inter-processor communication are left out. The relative effort between the master and subproblems depends on their sizes. For most problems in our experiments, the dominating work in DW is to solve the restricted master problem. On the 216 test problems from the multicommodity model, 99.82% of the CPU time was consumed on

solving master problems. On the 100 test problems from the financial model, this figure is 35.22% because the average problem size is smaller than the multicommodity problems.

Unlike simplex, the DW restricted master problem's size varies during a solution process. Using the column generation technique, proposals are added gradually to the restricted master problem. Therefore, the number of columns in the master problem keeps increasing, and so does the effort needed to solve the master problem at each iteration. At the k^{th} iteration, the dimensions of the restricted master problem is $(m_0 + h)$ by kh, given that all the h subproblems are of negative objective values. However, this assumption is not true especially near the end of the procedure, as some of the subproblems already have nonnegative objective values and stop returning proposals. It is difficult to know in advance the actual number of proposals added per iteration.

At each iteration, solving the restricted master problem is a simplex process. As is known, simplex-type algorithms find an optimal solution by moving along the outside edges of the feasible region. Different starting points lead to big variances in the number of iterations. Although the warm start technique contributes largely to the success of simplex [11], it also makes it difficult to estimate the per iteration complexity for DW.

In our experiments, we collected separately the CPU time for solving the master problem and subproblems, as well as the number of *inner* iterations. However, due to the aforementioned difficulties, we do not conduct detailed empirical analysis on the DW per iteration complexity. Rather, we will look at the overall CPU time in the next section.

4.2.3 Average Complexity of DW

We measure the average DW behavior by its CPU time from our experiments. Figure 4.7 provides an overall picture of DW CPU time vs. the number of rows in the master problem. The scattered linear distribution of the points appears here again, which tells us that besides $(m_0 + h)$, some other variables play a role as well.

As a decomposition algorithm, DW is assumed to be implemented in a cluster with $h+1$ processors. One of the processors solves the master problem, and the rest h processors are assigned to the h subproblems. Therefore, when we consider DW CPU time, h seems an important variable in a regression model because theoretically, the more subproblems in a problem, the more processors to share the workload of solving the problem. Similar to the

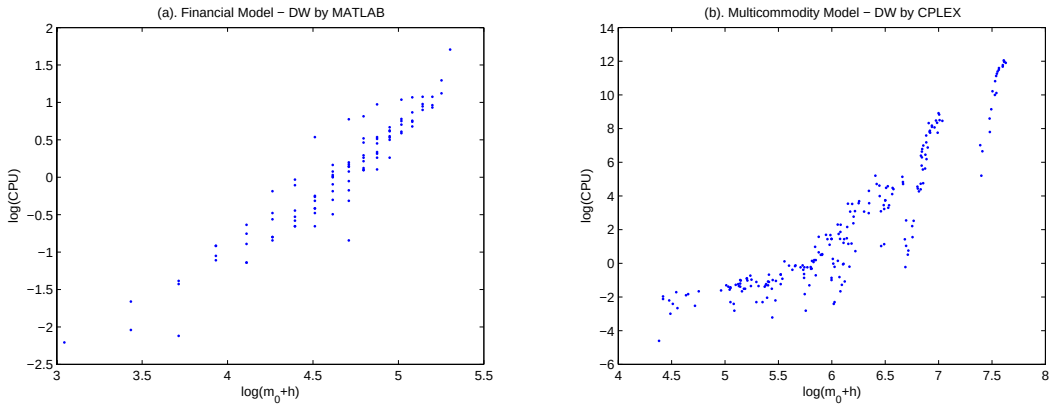


Figure 4.7: DW CPU Time vs. $(m_0 + h)$

previous discussions, it is reasonable to assume that the overall size parameters, such as $(m_0 + h)$ and $\overline{n}$, as well as the density indicators, such as $m_0\%$ and $nz\%$, have influences on DW CPU time. We then build regression models as follows

$$CPU = e^{\alpha}(m_0 + h)^{\beta}h^{\gamma} \tag{4.19}$$

Equation (4.19) is a simple model that examines the influence of $(m_0 + h)$ and h, i.e., the number of rows in the DW restricted master problem and the number of blocks. Notice that h appears twice in the model, but the regression results in Table 4.5 show that there is no redundancy for test set m, and according to the p value, h is slightly redundant for test set f.

Since the analysis is mainly on large sparse linear programs, it is natural to add one more density indicator, and the above model becomes

$$CPU = e^{\alpha}(m_0 + h)^{\beta}h^{\gamma}(nz\%)^{\delta} \tag{4.20}$$

Model (4.20) improves r^2 on the financial problems, but does not improve r^2 on the multicommodity problems. Moreover, variable $nz\%$ seems redundant for the latter.

Next, we try a few column-oriented models. Equation (4.21) examines DW CPU time over the total number of columns of a problem and the number of blocks. Particularly for block-angular structured problems, we add variable $m_0\%$ into the model, and this yields

Equation (4.22).

$$CPU = e^{\alpha}\overline{n}^{\beta}h^{\gamma} \tag{4.21}$$

$$CPU = e^{\alpha}\overline{n}^{\beta}h^{\gamma}(m_0\%)^{\delta} \tag{4.22}$$

Surprisingly, although the r^2 values are quite improved on test sets f and m, h seems redundant in model (4.22), probably because $m_0\%$ contains part of the information that h provides.

Lastly, we build a regression model based on the four inner structure variables.

$$CPU = e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon} \tag{4.23}$$

Regression results are summarized in Table 4.5. For models (4.19) and (4.20), DW CPU time increases with the number of rows in the master problem as well as to h. Adding $nz\%$ makes a considerable improvement on the fitting of $(4.19f)$, but not much on $(4.19m)$. For models (4.21) and (4.22), DW CPU time increases with the total number of columns but decreases with h. Adding $m_0\%$ improves the fit on both test sets.

Readers may have noticed that there are two different conclusions on h. For (4.19) and (4.20), little information of the overall problem is given, and h to some extent indicates the problem size. On the contrary, (4.21) and (4.22) contain the overall number of columns, h then divides the problem into several processors. Therefore, the 'increasing' and 'decreasing' with h do not really conflict. It all depends on the way the regression models are built.

The inner-structure based regression model (4.23) achieves the highest r^2 in Table 4.5, although with a few redundant variables. According to the results, DW takes more CPU time if a problem has more linking constraints or more blocks. The subproblem dimensions, m_{sub} and n_{sub}, take opposite signs for both test sets. In the solution process of a decomposition algorithm, the dimensions of blocks affect the load balancing between the master processor and subproblem processors. Intuitively, if we assume that m_{sub} and n_{sub} increase proportionally, which is not unusual in application, then any one of m_{sub} or n_{sub} is capable to represent the effort required for a subproblem.

4.3 Relative Efficiency: DW vs. Simplex

Decomposition algorithms have been developed for several decades, but they are not widely adopted in practice. Ho and Loute [45] concluded about 20 years ago that it was unlikely that decomposition could generally be significantly more efficient than simplex, based on their experiments where DW only outperformed simplex on 2 out of the 30 test problems. We will see whether this conclusion still holds in today's computing environments.

It is generally believed that if a problem has a special structure, it should be exploited. Yet, with the rapid development of computer technology, the answer is not so definite. In the 1950's, a problem with the order of 50 by 100 was considered large-scale [48], whereas modern computers can handle problems with millions of constraints or variables [8]. Today's capacity of problem solving was unthinkable in the past. Both memory and CPU have been greatly advanced, and consequently, decomposition seems unnecessary in some cases.

However, besides reducing dimension, decomposition has other motivations such as partitioning heterogeneous problems, decentralizing complex multilevel models, and parallelizing computations [61]. On very large-scale problems, it is a wise solution to implement decomposition algorithms over a cluster of *affordable* processors [29][4] compared to a super computer, which is usually prohibitively expensive. The key is to determine whether a problem is worth calling decomposition. Conventionally, decomposition algorithms are considered advantageous for solving *large* problems, but this argument itself is quite vague: how big is truly big?

Due to the difficulty of giving a rigorous complexity formula for either simplex or DW practically, people use CPU time to measure the complexity of them. By comparing the relative efficiency of DW and simplex over a set of different sized problems, we try to find out when the direct solution approach is more efficient, and when decomposition algorithms would outperform. A ratio is defined as

$$R = \frac{CPU_{DW}}{CPU_{simplex}} \tag{4.24}$$

According to our experiments, $R \in (5.09 \times 10^{-5}, 0.21)$ for the financial problems, and $R \in (0.1026, 298.37)$ for the multicommodity problems.

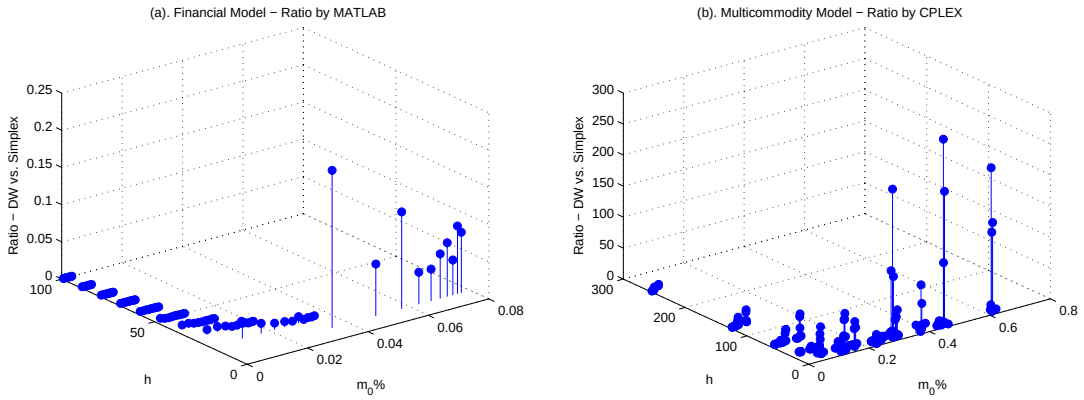


Figure 4.8: Ratio (DW/Simplex) vs. $m_0\%$ and h

4.3.1 Overview of Data

We first take an overview of the numerical results. Figure 4.8 reveals the ratio vs. $m_0\%$ and h on both test models. Since some of the stems are too short to observe, we take logarithms on the ratios[6], as shown in Figure 4.9. It is commonly believed that higher $m_0\%$ leads to more computational effort for decomposition algorithms. In a distributed computing environment, more blocks (h) means there would be more processors to solve a problem, i.e., the workload is shared. Figure 4.8 (a) and (b) confirm these intuitive conclusions: the ratio, R, increases with $m_0\%$ and decreases with h. More specifically, for problems with lower $m_0\%$ and higher h, decomposition algorithms (DW in this case) are more advantageous than direct solution methods (simplex). Most of the test problems have a low percentage of linking constraints, but for those with higher $m_0\%$, the ratios increase suddenly. Results from both test models suggest a similar tendency, but the ranges of R on each model vary greatly - see the scales of the vertical axes in Figure 4.8 (a) and (b). The difference can be attributed to the two different solvers used on each model, and this issue will be discussed below.

We can also examine R in the financial model over its economic interpretations. Basically, more assets lead to more linking constraints, and the number of scenarios also means the number of blocks. Figure 4.10 reveals the ratio vs. the number of assets and scenarios,

[6]To avoid negative values for the ratios, we take $\log(R+1)$.

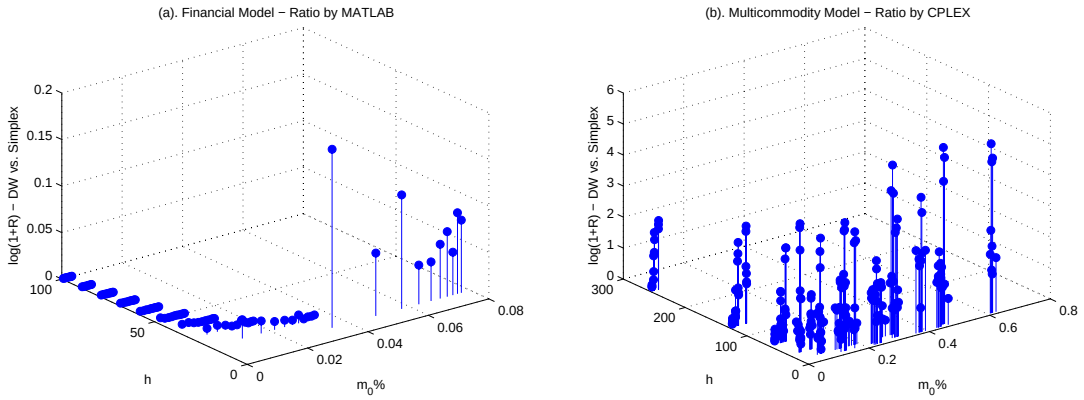


Figure 4.9: Ratio (DW/Simplex) vs. $m_0\%$ and h

which are the only two variables in the financial model. For problems with fewer scenarios, the ratios are high; for the majority of other problems, the ratios are quite low.

The multicommodity model is difficult to address visually as it contains more variables, which have joint effects on the solution time. We then examine the data by groups. Figures 4.11 (a) and (b) provide pictures of the relative efficiency on problems with 128 and 256 nodes, respectively. The smaller problems require little solving time, and hence a few upper lines in the figures fluctuate a lot. Ignore them when looking at the figure.

As mentioned in Section 2.2.2, for each pair of (n, h), 12 problems are generated by MNETGEN. Recall the way that the 12 problems are generated, and recall the four-type category. Put simply, problems from type I to type IV gradually become more and more complicated to solve, i.e., denser[7]. From the behavior of the 12 problems on each pair of (n, h), together with the variance in their structures, we try to find a clue about how the size parameters influence the ratio R.

Looking at Figure 4.11 horizontally, we see that the curves rise a bit (except the upper ones that fluctuate too much), which implies that DW takes longer than simplex when a problem is more dense. Looking vertically, we get 12 groups of points identified by their problem number. The problems with the same number (e.g., there are six and seven

[7]Here, we loosely use the word 'denser' to indicate the situation where the network has more arcs, and more arcs have mutual capacity constraints.

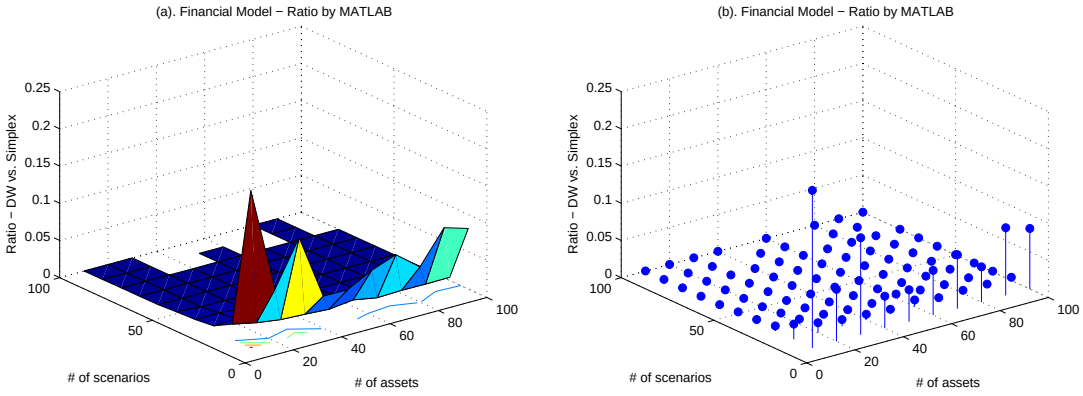


Figure 4.10: Ratio (DW/Simplex) vs. Number of Assets and Number of Scenarios

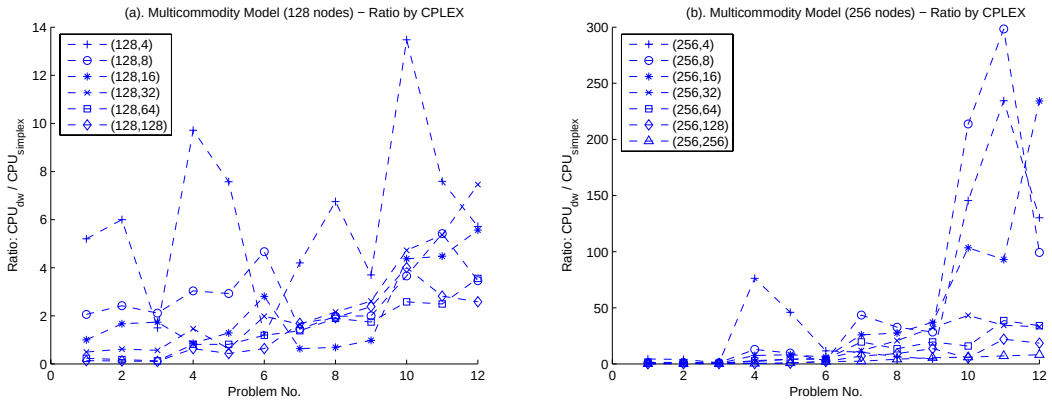


Figure 4.11: Ratio (DW/Simplex) of Multicommodity Problems with 128 and 256 Nodes

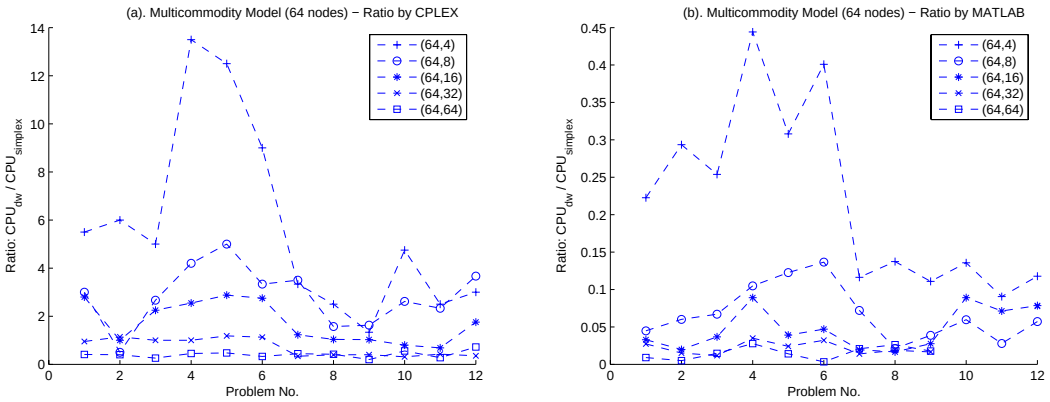


Figure 4.12: Ratio (DW/Simplex) of Multicommodity Problems with 64 Nodes

problems No.1 in (a) and (b), respectively) have similar structures but various number of commodities. Those with higher h reside in lower positions. It is easy to understand that the more subproblems, the more processors to divide the workload, which is also a major advantage of decomposition in parallel computing. It is worth noting that dimensions are not the only factors that determine CPU time. The three problems generated for each (n, h) pair are supposed to have similar structures, yet the number of iterations and the CPU time vary a lot sometimes. This is probably due to the real shape of the feasible polyhedron - the facets, edges, and vertices can be different [82]. At present, this issue is still an open topic to address.

Inevitably, examining an algorithm becomes examining its implementation. To address the influence of different solvers, we did another set of experiments. As mentioned earlier, all the test problems in the multicommodity model are solved by CPLEX. We now solve some of them (smaller problems with 64 nodes) with another solver. A MATLAB built-in revised simplex code, with the smallest-subscript rule, is adopted and modified for warm start purposes. In this case, both the direct solution approach (simplex) and the decomposition algorithm (DW) call this code, so that their relative efficiency is comparable.

Figure 4.12 reveals the ratios on test problems for both solvers. The overall tendencies of (a) and (b) are consistent. When the transportation network is denser and there are more linking constraints, DW is less advantageous. This agrees with what we have expected.

Comparing Figure 4.12 (a) with (b), we see the difference between the two groups of trend lines. Most of the curves in (a) are quite flat, while those in (b) show a clearer rising tendency. As a leading software package, CPLEX already takes advantages of the sparse matrices when doing Cholesky factorizations. In this sense, the simple revised simplex code at MATLAB seems more suitable for comparing with and without decomposition. However, the MATLAB solver is unbearably slow for larger problems such as the problems with 128 and 256 nodes in this model. So CPLEX is still used for all of the multicommodity problems.

4.3.2 Regression Models

Next, we discuss the ratios by regression analysis. When we compare simplex and DW, since the former solves the problem as a whole unit, and the latter explores the inner structures, our first model comes naturally

$$R = e^{\alpha}(m_0\%)^{\beta}h^{\gamma}, \tag{4.25}$$

where $m_0\%$ to some extent represents the problem's decomposability, and h means the number of processors involved. Regression results are shown in Table 4.6. We expect the regression analysis on the ratios to be difficult. Ratio R involves information of the DW and simplex CPU time, as well as the distributed computing issues. The r^2 for test set f is surprisingly good, yet for test set m, r^2 is very low. For reference purposes, we also provide the regression results on multicommodity problems with 64 nodes ($m*$), which were solved by the simplex solver in MATLAB. Evidently, model (4.25) achieves a much better fit on $m*$ than m, but $m_0\%$ is insignificant here. According to the results of (4.25), we can see that the regression analysis is model dependant: a parameter's value and significance level are different on the three test sets. For example, parameter β takes positive values on test set m and $m*$, which implies that larger $m_0\%$ leads to relatively worse performance of DW. However, β is negative on test set f, probably because all the problems in f have very low $m_0\%$, and hence within this range ($< 8\%$), a few more linking constraints do not harm DW's efficiency.

Our next regression model (4.26) takes into account the number of rows in the master problem (from the DW CPU time side), the total number of columns (from simplex side),

and the nonzero density of the problem. Regression fitting is still good for test set f, and no variable is redundant in this model. Although some variables have high p value for test set m and $m*$, fitting is better for both of them than model (4.25).

$$R = e^{\alpha}(m_0 + h)^{\beta}\overline{n}^{\gamma}(nz\%)^{\delta} \tag{4.26}$$

In the factor $(m_0 + h)$, h corresponds to the number of convexity constraints in the DW master problem. However, parameter h seemingly plays a bigger role especially in decomposition. So, the next model tries using m_0 and h as separate variables

$$R = e^{\alpha}m_0^{\beta}\overline{n}^{\gamma}h^{\delta} \tag{4.27}$$

The r^2s for the three test sets are all improved.

Lastly, we build a model purely based on the four basic inner structure variables

$$R = e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon} \tag{4.28}$$

There are redundancies on all the test sets, but the r^2s for (4.28) are the highest in Table 4.6.

4.3.3 Prediction

In the previous section, we have built several models for the ratio R by using regression analysis. These models show the relative efficiency of DW and simplex, i.e., with and without decomposition. The results can hopefully provide helpful information for users to estimate, for a given problem, which solution approach would be faster, and hence determine what method to use. Next, we will investigate whether these results are reliable to be used as guidance in practice. In this section, we use *robust regression* to re-assess the parameters in the aforementioned models. The results will be slightly different from those by using the ordinary least squares regression because the effect of outliers are reduced to minimum by assigning small weights to them. It is commonly believed that robust regression can lead to a better fit especially for prediction purposes [64].

Random Prediction Due to space limitations, we only test the prediction ability on one regression model, with some of the test problems. We first randomly select ten test

problems from the stochastic financial model and from the multicommodity network flow model. Then, the rest of the test problems are used as data for the robust regression. We choose regression model (4.28) to test. Since m_{sub} and n_{sub} seem redundant with each other sometimes in our previous discussion, we further modify the model as

$$R = e^{\alpha} m_0^{\beta} h^{\gamma} n_{sub}^{\delta} \tag{4.29}$$

The $robustfit$ function in MATLAB uses an iteratively reweighted least squares algorithm, with the weights at each iteration calculated by applying the *bisquare* function to the residuals from the previous iteration. This algorithm gives lower weight to points that do not fit well. The results are less sensitive to outliers in the data as compared with ordinary least squares regression [64]. The robust regression results are given in Table 4.7. The weights assigned to the variables on both test sets are provided in Tables C.1 and C.2 on page 177.

Table 4.8 shows the validation of model (4.29) on financial problems. Based on the observed and predicted data, we evaluate the errors in two ways: one, if the observed and predicted results suggest the same pick (simplex or DW), the *pick*% takes value 1, otherwise, 0; two, we examine the percentage of errors defined as $\frac{|\hat{R}-R|}{R}$%. For each test problem, a 95% confidence interval for $\hat{R}$ is also provided in the table. Experiments show that *pick*% is 100% for the 10 randomly chosen financial problems. Similarly, Table 4.9 shows the results on 10 multicommodity problems, for which *pick*% is 90% valid. On the other hand, the errors measured as $\frac{|\hat{R}-R|}{R}$% are very large - averaging 43% and 97% for the two types of model. Evidently, a correct choice (in the *pick*% sense) can usually be made, even with a model that has huge prediction errors (in the $\frac{|\hat{R}-R|}{R}$% sense).

Extrapolation Our analysis aims at *large-scale* problems, so the ability of model extrapolation is important. Meanwhile, we want the test problems to be solved in a reasonable period of time. Therefore, rather than generating new larger problems, we pick the largest 10 problems from the current sets f and m as the ones to be tested, and use the rest to do the robust regression. Again, we test the inner-structure regression model (4.29). The regression results are given in Table 4.10. The weights assigned to the variables on both test sets are provided in Tables C.3 and C.4 in Appendix C.

The extrapolation tests are summarized in Tables 4.11 and 4.12. In both tables, the problems are arranged in ascending order in terms of $\overline{n}$. We obtained 100% accuracy of $pick\%$ on the 20 problems from both test sets.

4.4 Preliminary Conclusions

In our experiments, we actually tried many more regression models, but only reported some of the representative ones in this thesis. We obtained satisfactory fits over the current regression models, but there still could be better ones that we did not build. To sum up, we give the following preliminary conclusions drawn from our experiments:

1. Sparse problems are not significantly easier for the simplex method to solve in terms of the $3\overline{m}$ bound.

2. On problems with a block-angular structure, $\overline{n}$ is as important as $\overline{m}$ for simplex.

3. For examining a block-angular structured problem's complexity, its inner structure matters. Parameters such as $m_0\%$ and $nz\%$ are beneficial complements in a regression model to predict the number of iterations or CPU time of simplex or DW.

4. Regression results tend to be model dependent and implementation (solver) dependent. Putting aside the economic interpretations, both the test-problem models have block-angular structures. However, the regression results from the two models are not completely consistent, which means that the conclusions are difficult to generalize for all block-angular structured linear programs.

5. The number of blocks and the percentage of linking constraints are significant when considering the relative efficiency of with and without decomposition.

6. The four 'inner structure' parameters, m_0, h, m_{sub} and n_{sub} are easy to use, and have good regression results on all the models. Although some of them seem redundant (with high p values), the r^2 of these models still outperform almost all other models.

7. Some unexpected signs in the regression analysis are due to interactions and collinearity among variables.

8. Several variants of simplex have already involved advances in sparsity techniques [75]. It is more difficult for decomposition algorithms to significantly outperform simplex even for problems with a special structure.

9. Prediction and extrapolation tests on selected regression models reveal high accuracy, which suggests that empirical analysis is a viable way in practice to predict the relative efficiency of DW and simplex based on problem characteristics.

Table 4.2: Regression Results for Simplex Iteration Counts

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)*					
$e^{\alpha}\overline{m}^{\beta}$	f	0.1580 (0.7576)	1.0369				0.7765
(4.1)	m	-1.6470	1.3268				0.8618
$e^{\alpha}\overline{m}^{\beta}(m_0\%)^{\gamma}$	f	-1.0569	1.4112	0.4181			0.8240
(4.2)	m	-3.9876	1.7970	0.6811			0.9315
$e^{\alpha}\overline{m}^{\beta}(nz\%)^{\gamma}$	f	-1.2282	1.4366	0.4355			0.8227
(4.3)	m	-3.2429	3.0283	1.7051			0.8666
$e^{\alpha}\overline{n}^{\beta}$	f	0.3652 (0.4893)	0.9825				0.7576
(4.4)	m	-2.5652	1.2511				0.9473
$e^{\alpha}\overline{n}^{\beta}(m_0\%)^{\gamma}$	f	-1.3290	1.4704	0.5495			0.8278
(4.5)	m	-3.8540	1.4870	0.4190			0.9809
$e^{\alpha}\overline{n}^{\beta}(nz\%)^{\gamma}$	f	-1.5701 (0.0054)	1.5078	0.5785			0.8264
(4.6)	m	-3.7319	2.2267	1.1075			0.9712
$e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon}$	f	16.4160	-0.3691 (0.2251)	6.7092	29.9476	-34.2685	0.8803
(4.7)	m	-3.1616	0.6325	1.0922	-0.3038	1.0721	0.9828

*Most variables have very low p values, and here we only indicate those with $p \geq 0.05$.

Table 4.3: Regression Results for Simplex CPU Time

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)					
$e^{\alpha}\overline{m}^{\beta}(m_0\%)^{\gamma}$	f	-15.6688	3.0130	0.7808			0.9441
(4.8)	m	-23.2181	3.3986	1.6058			0.8947
$e^{\alpha}\overline{m}^{\beta}(nz\%)^{\gamma}$	f	-15.9991	3.0634	0.8165			0.9433
(4.9)	m	-25.5331	10.6416	8.3689			0.8123
$e^{\alpha}\overline{n}^{\beta}(m_0\%)^{\gamma}$	f	-16.2372	3.1367	1.0593			0.9473
(4.10)	m	-22.5771	2.7605	1.0640			0.9219
$e^{\alpha}\overline{n}^{\beta}(nz\%)^{\gamma}$	f	-16.7167	3.2128	1.1196			0.9465
(4.11)	m	-21.8575	4.2963	2.4236			0.8909
$e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon}$	f	4.7207 (0.1750)	0.1259 (0.7025)	8.8941	33.4388	-37.7261	0.9668
(4.12)	m	-20.1835	2.3226	1.7855	0.0521 (0.8038)	0.2610 (0.3658)	0.9396

Table 4.4: Regression Results for DW Number of Proposals

Regression Model	Test Set	α	β	γ	δ	ϵ (p value for each coefficient)	r^2
$e^{\alpha}(m_0+h)^{\beta}(m_0\%)^{\gamma}$	f	-0.2920	0.7664	-0.3288			0.8579
		(0.2713)					
(4.14)	m	-7.2457	2.0255	-0.7729			0.8817
$e^{\alpha}(m_0+h)^{\beta}(nz\%)^{\gamma}$	f	-0.1354	0.7338	-0.3328			0.8576
		(0.6094)					
(4.15)	m	-6.4911	1.1902	-0.8345			0.8801
$e^{\alpha}\overline{n}^{\beta}(m_0\%)^{\gamma}$	f	0.2101	0.5054	-0.0157			0.8448
		(0.3906)		(0.7131)			
(4.16)	m	-6.5444	1.5729	0.6670			0.9029
$e^{\alpha}\overline{n}^{\beta}(nz\%)^{\gamma}$	f	0.2089	0.5066	-0.0140			0.8448
		(0.4144)		(0.7573)			
(4.17)	m	-6.0039	2.4610	1.4346			0.8603
$e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon}$	f	-3.7150	0.0045	-0.8269	-4.9796	6.9350	0.8647
			(0.9776)	(0.1286)	(0.1410)		
(4.18)	m	-5.3933	1.3552	0.9538	0.0656	0.1337	0.9198
					(0.6307)	(0.4765)	

Table 4.5: Regression Results for DW CPU Time

Regression Model	Test Set	α β γ δ ϵ (p value for each coefficient)	r^2
$e^{\alpha}(m_0+h)^{\beta}h^{\gamma}$ (4.19)	f	-7.7068 1.5926 0.0906 (0.0754)	0.8953
	m	-26.0247 4.2161 0.7879	0.8745
$e^{\alpha}(m_0+h)^{\beta}h^{\gamma}(nz\%)^{\delta}$ (4.20)	f	-6.6605 1.0687 1.0777 0.4844	0.9223
	m	-25.6836 4.3762 1.0293 0.2836 (0.2674)	0.8752
$e^{\alpha}\overline{n}^{\beta}h^{\gamma}$ (4.21)	f	-9.9164 2.4003 -3.1405	0.8263
	m	-30.1159 4.4103 -3.0658	0.7758
$e^{\alpha}\overline{n}^{\beta}h^{\gamma}(m_0\%)^{\delta}$ (4.22)	f	-7.4563 1.3533 -0.4246 0.6522 (0.2025)	0.9164
	m	-25.6363 3.5398 -0.0180 2.5267 (0.9548)	0.8637
$e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon}$ (4.23)	f	-2.9415 0.2278 1.7106 8.9980 -9.3636 (0.0522)(0.1147)	0.9251
	m	-25.2577 3.8158 1.1263 0.2554 -0.0326 (0.4491)(0.9440)	0.8859

Table 4.6: Regression Results for The Ratio of DW vs. Simplex

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)					
$e^{\alpha}(m_0\%)^{\beta}h^{\gamma}$	f	4.3059	-1.0181	-4.3211			0.9431
(4.25)	m	1.9448	1.5817	0.8382			0.4030
	$m*$	-0.6577	0.1113 (0.5572)	-0.8252			0.6869
$e^{\alpha}(m_0+h)^{\beta}\overline{n}^{\gamma}(nz\%)^{\delta}$	f	8.3541	2.4397	-3.6970	-1.4222		0.9353
(4.26)	m	-4.2694	2.1065	-0.7645	0.0508 (0.8651)		0.6077
	$m*$	2.6075	0.4976	-0.6785 (0.1767)	0.4411 (0.4201)		0.7362
$e^{\alpha}m_0^{\beta}\overline{n}^{\gamma}h^{\delta}$	f	3.9978	-0.9636	1.0628	-4.4761		0.9438
(4.27)	m	-5.0498	1.4364	-0.0858 (0.7218)	-0.5699		0.6299
	$m*$	3.1098	0.7216	-1.2917	0.2867 (0.4451)		0.7412
$e^{\alpha}m_0^{\beta}h^{\gamma}m_{sub}^{\delta}n_{sub}^{\epsilon}$	f	-7.2443	0.0642 (0.8321)	-7.0808	-23.7059	27.5393	0.9510
(4.28)	m	-5.0741	1.4932	-0.6592	0.2034 (0.4074)	-0.2935 (0.3848)	0.6312
	$m*$	-7.70E13 (0.4227)	0.6860	-0.9202	1.85E13 (0.4227)	-1.2597	0.7353

$*$These problems were solved by the simplex solver in MATLAB.

Table 4.7: Robust Regression Results for Random Prediction Tests

Problem	Regression Coefficients				p values for				$dof*$
	α	β	γ	δ	α	β	γ	δ	
f	4.8389	-0.8554	-3.1161	0.6154	< 0.05	< 0.05	< 0.05	0.2966	81
m	-5.3063	1.4127	-0.6683	-0.0166	< 0.05	< 0.05	< 0.05	0.9478	202

*Degrees of freedom for error

Table 4.8: Prediction Experiments on Randomly Chosen Financial Problems

Problem		Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
70	10	7.24E-02	DW	4.34E-02	DW	2.81E-02	6.72E-02	40.02%	1
80	90	7.44E-05	DW	8.84E-05	DW	7.31E-05	1.07E-04	18.80%	1
100	20	4.35E-03	DW	4.93E-03	DW	3.59E-03	6.76E-03	13.28%	1
80	30	5.95E-04	DW	1.75E-03	DW	1.47E-03	2.07E-03	193.22%	1
20	70	4.05E-04	DW	4.75E-04	DW	3.93E-04	5.74E-04	17.20%	1
40	100	1.61E-04	DW	1.12E-04	DW	9.33E-05	1.34E-04	30.49%	1
100	30	1.04E-03	DW	1.55E-03	DW	1.25E-03	1.92E-03	47.99%	1
100	10	8.13E-02	DW	3.77E-02	DW	2.03E-02	6.99E-02	53.67%	1
50	10	4.29E-02	DW	5.04E-02	DW	3.65E-02	6.96E-02	17.40%	1
90	90	8.36E-05	DW	8.14E-05	DW	6.67E-05	9.94E-05	2.63%	1
Average								43.47%	100%

Table 4.9: Prediction Experiments on Randomly Chosen Multicommodity Problems

Problem			Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	$No.$	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
64	32	5	1.18	simplex	0.57	DW	0.41	0.79	51.94%	0
256	8	3	1.04	simplex	3.82	simplex	2.83	5.15	267.45%	1
128	4	5	7.57	simplex	5.81	simplex	4.40	7.67	23.25%	1
128	64	9	1.74	simplex	1.57	simplex	1.27	1.94	9.91%	1
64	16	8	1.04	simplex	1.27	simplex	1.04	1.54	21.90%	1
256	8	6	2.78	simplex	10.45	simplex	8.36	13.06	276.42%	1
128	8	9	2.00	simplex	6.55	simplex	5.22	8.23	227.68%	1
256	64	5	4.15	simplex	2.63	simplex	2.14	3.22	36.68%	1
256	64	4	2.90	simplex	2.59	simplex	2.12	3.18	10.67%	1
128	128	8	1.92	simplex	1.08	simplex	0.83	1.41	43.65%	1
Average									96.95%	90%

Table 4.10: Robust Regression Results for Extrapolation Tests

Problem	Regression Coefficients				p values for				$dof*$
	α	β	γ	δ	α	β	γ	δ	
f	4.7313	-0.9813	-3.4524	0.9576	< 0.05	< 0.05	< 0.05	0.0673	81
m	-4.4299	1.4402	-0.7050	-0.1721	< 0.05	< 0.05	< 0.05	0.4957	202

*Degrees of freedom for error

Table 4.11: Extrapolation Experiments on the Financial Problems

Problem		Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
10	100	2.63E-04	DW	3.28E-04	DW	2.14E-04	5.03E-04	25.08%	1
80	90	7.44E-05	DW	7.48E-05	DW	6.14E-05	9.11E-05	0.48%	1
20	100	2.39E-04	DW	1.79E-04	DW	1.35E-04	2.38E-04	24.99%	1
90	90	8.36E-05	DW	6.85E-05	DW	5.57E-05	8.43E-05	18.06%	1
30	100	2.88E-04	DW	1.26E-04	DW	1.01E-04	1.58E-04	56.26%	1
40	100	1.61E-04	DW	9.86E-05	DW	8.08E-05	1.20E-04	38.79%	1
60	100	1.07E-04	DW	7.05E-05	DW	5.80E-05	8.57E-05	34.26%	1
80	100	1.47E-04	DW	5.62E-05	DW	4.56E-05	6.93E-05	61.85%	1
90	100	8.40E-05	DW	5.14E-05	DW	4.13E-05	6.40E-05	38.80%	1
100	100	1.84E-04	DW	4.75E-05	DW	3.78E-05	5.97E-05	74.23%	1
Average								37.28%	100%

Table 4.12: Extrapolation Experiments on the Multicommodity Problems

Problem			Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	$No.$	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
256	128	8	9.21	simplex	2.08	simplex	1.45	2.98	77.43%	1
256	128	10	5.32	simplex	5.71	simplex	3.89	8.39	7.46%	1
256	128	12	18.48	simplex	5.70	simplex	3.88	8.37	69.16%	1
256	128	11	22.06	simplex	5.80	simplex	3.94	8.53	73.73%	1
256	256	9	5.51	simplex	1.09	simplex	0.72	1.64	80.26%	1
256	256	7	2.43	simplex	1.10	simplex	0.73	1.66	54.71%	1
256	256	8	4.16	simplex	1.19	simplex	0.80	1.79	71.29%	1
256	256	10	6.05	simplex	3.26	simplex	2.12	5.03	46.05%	1
256	256	11	7.13	simplex	3.27	simplex	2.12	5.04	54.14%	1
256	256	12	8.21	simplex	3.35	simplex	2.17	5.18	59.18%	1
Average									59.34%	100%

Chapter 5

With and Without Decomposition - ACCPM vs. IPM

In the previous chapter, we have compared the relative efficiency between the Dantzig-Wolfe decomposition method (DW) [20] and the simplex method. In this chapter, we do the comparison through another approach: we compare ACCPM (Analytic Center Cutting Plane Method) versus IPM (Interior Point Method). ACCPM [34] is a relatively new method based on IPM. ACCPM with multiple cuts can be viewed as the dual problem of DW. Therefore, comparing the complexity of ACCPM and IPM means comparing with and without decomposition in the IPM-based algorithms.

Based on the conclusions from the literature, we first deduce the complexity of ACCPM and IPM, respectively, over problems with a block-angular structure. The relative efficiency obtained by this kind of theoretical analysis has many limitations. For example, most of the conclusions provide complexity bounds, which implies a worst-case analysis. Users actually care more about the real behavior of an algorithm on various problems. However, average-case analysis is even more difficult than worst-case [16], as it is virtually impossible to get a statistical average from sample problems. In addition, a theoretical proof often makes strong assumptions, which are not true in practice. Therefore, empirical analysis has gained more attention in recent years (see [65] [49], and [66]).

Different from some papers' 'last-page' numerical results, we use a large set of test problems with both primal and dual block-angular structures to test our conclusions, *i.e.*,

the theoretical relative efficiency of ACCPM vs. IPM. By this way, the gap between theory and practice can be clearly identified. We try to provide helpful information for decision making such as which solver (algorithm) to choose.

5.1 Complexity Analysis for ACCPM vs. IPM

In this section, we analyze the theoretical complexity of ACCPM and IPM. Our study is based on some conclusions by other researchers, and we focus mainly on the worst-case complexity bounds. Compared to the simplex-based algorithms, IPM as well as its counterpart in decomposition framework, ACCPM, have polynomial complexity for both the average and the worst case.

Both IPM and ACCPM are iterative process. Therefore, the complexity can be factored as the number of iterations times the computational effort needed for each iteration.

5.1.1 Complexity of IPM

As discussed earlier, the path following method is the most efficient IPM variant, based on which ACCPM is developed. Given a problem with m equalities and n variables, the path following method needs at most

$$O\left(\sqrt{n}\log(\varepsilon_0/\varepsilon)\right) \tag{5.1}$$

iterations to reduce the duality gap from ε_0 to ε [5]. In an observed average case, the primal-dual path following algorithm needs

$$O\left(\log n \log(\varepsilon_0/\varepsilon)\right) \tag{5.2}$$

iterations [5]. However, no satisfactory explanation has been achieved for this behavior yet [5].

For IPMs, Newton's method is used to solve an unconstrained optimization problem. To get a Newton's direction, we need to solve a linear system of equations. At each iteration, this work involves

$$O\left(m^2n + m^3\right) \tag{5.3}$$

arithmetic operations [62].

The total complexity for solving a problem equals the number of iterations times the complexity per iteration. Therefore, for a block-angular structured problem with m_0 linking constraints and h blocks of $m_s \times n_s$, the complexity for solving it as a whole requires

$$F_{ipm} = O\left((hm_s + m_0)^2 \cdot hn_s + (hm_s + m_0)^3\right) \cdot O\left(\sqrt{hn_s}\log(\varepsilon_0/\varepsilon)\right) \tag{5.4}$$

arithmetic operations in total.

5.1.2 Complexity of ACCPM with Multiple Cuts

The studies on the complexity of ACCPM with multiple cuts were initiated by Ye [85]. Goffin and Vial [38] confirmed and further expanded Ye's analysis. We first review some important conclusions in the literature, and then apply these complexity conclusions to the special case - problems with a block-angular structure.

Important Conclusions

Ye [85] first analyzed the complexity of ACCPM with multiple cuts, and proved that ACCPM with multiple cuts added at each iteration is still 'a fully polynomial approximation algorithm'. Following Ye's study, Goffin and Vial [38] further proved the complexity for the recovery of a new analytic center, and proposed a feasibility restoration direction. For a problem with m inequalities and n variables ($m < n$), their conclusions are as follows:

1. Recovery of a new analytic center [38]
   The number of Newton steps to compute an updated analytic center is bounded by

   $$O\left(p\log(p+1)\right), \tag{5.5}$$

   where p is the number of new cuts added by the oracle(s). In fact, the value of p may vary over iterations, and it is more precise to denote it as p_k. Notice that this complexity result is only dependent on the number of new cuts per iteration, which means that the dimension parameters m and n are irrelevant here.

2. Convergence results [85][38]
   The number of cuts generated is at most

$$O\left(\frac{\overline{p}^2 n^2}{\varepsilon^2}\right), \tag{5.6}$$

   where $\overline{p}$ is the maximum number of cuts generated at any given iteration, and ε is the duality gap, also known as the final precision.

In these studies, the multiple cuts generated by the subproblem(s) are used as given, *i.e.*, how the cuts are generated is out of consideration. Actually, for a generic convex optimization problem, the query points can be generated randomly, and more query points lead to more accurate approximations of the original functions. However, when ACCPM is used in decomposition, these query points are passed from the subproblems, *i.e.*, the optimal solutions of the subproblems. This means that the effort required to solve the subproblems do impact the overall algorithm. Therefore, we need to consider the complexity for both the master problem and the subproblem in a decomposition context.

Complexity of Master Problem

For a problem with a block-angular structure, assuming that each subproblem returns one proposal at each iteration, the number of Newton steps to recover a new analytic center is bounded by

$$O\left(h \log(h+1)\right), \tag{5.7}$$

where h is the number of blocks in the problem. The value v corresponds to the number of *inner* iterations for solving the master problem per *outer* iteration.

Since usually we have $m \leq n$, according to (5.3), the total number of arithmetic operations per iteration is $O\left(n^3\right)$. However, in ACCPM, there are many more constraints (cuts) than variables in the master problem, and the complexity becomes $O\left(m^3\right)$ in this case. Therefore, the computational cost of one Newton step at the k^{th} iteration is bounded by

$$O\left((2m_0 + kh)^3\right), \tag{5.8}$$

where $2m_0 + kh$ is the total number of rows in the master problem at the k^{th} iteration (see (3.17) to (3.19) on page 28), and $2m_0$ corresponds to the initial box constraints.

According to (5.7) and (5.8), the number of arithmetic operations of the master problem at the k^{th} iteration can be calculated as

$$F^k_{accpm,M} = O\left((2m_0 + kh)^3 \cdot h\log(h+1)\right). \tag{5.9}$$

According to (5.6), the number of cutting planes generated throughout the algorithm is at most

$$v = O\left(\frac{h^2(m_0+h)^2}{\varepsilon^2}\right). \tag{5.10}$$

Since we assume that each subproblem generates one proposal per iteration, there are totally h cuts added per iteration. Then, the number of outer iterations is at most

$$v' = v/h = O\left(\frac{h(m_0+h)^2}{\varepsilon^2}\right). \tag{5.11}$$

Summing up $F^k_{accpm,M}$ from all the v' iterations, the overall complexity of the master problem is bounded by

$$F_{accpm,M} = O\left(\sum_{k=1}^{\lceil v' \rceil}(2m_0 + kh)^3 \cdot h\log(h+1)\right). \tag{5.12}$$

For simplicity of notations, we use v' as an integer, and ignore its magnitude symbol. So, the ACCPM master problem complexity becomes

$$F_{accpm,M} = O\left(h\log(h+1)\sum_{k=1}^{\frac{h(m_0+h)^2}{\varepsilon^2}}(8m_0^3 + 12m_0^2kh + 6m_0k^2h^2 + k^3h^3)\right). \tag{5.13}$$

In an O-notation [77], only the term with the highest power is kept. Ignoring all the constant coefficients, we obtain

$$O\left(h\log(h+1)\left[m_0^3 \cdot \frac{h(m_0+h)^2}{\varepsilon^2} + m_0^2h \cdot \frac{h^2(m_0+h)^4}{\varepsilon^4} + \sum_{k=1}^{\frac{h(m_0+h)^2}{\varepsilon^2}}(m_0k^2h^2 + k^3h^3)\right]\right) \tag{5.14}$$

According to the results of sum of squares and sum of cubes

$$\begin{aligned} 1 + 2^2 + \cdots + n^2 &= n(n+1)(2n+1)/6, \\ 1 + 2^3 + \cdots + n^3 &= n^2(n+1)^2/4, \end{aligned} \tag{5.15}$$

we obtain

$$O\left(h\log(h+1)\left[m_0^3 \cdot \frac{h(m_0+h)^2}{\varepsilon^2} + m_0^2 \cdot \frac{h^3(m_0+h)^4}{\varepsilon^4} + m_0 \cdot \frac{h^5(m_0+h)^6}{\varepsilon^6} + \frac{h^7(m_0+h)^8}{\varepsilon^8}\right]\right) \tag{5.16}$$

as the total complexity of the master problem for the entire ACCPM solving process. Usually, in the O-notation, only the term with the highest power is kept, but we now keep the full equation of (5.16) for further discussion.

Complexity of Subproblems

In a distributed computing environment with one subproblem assigned to each node, the CPU time for solving all the subproblems equals the time for solving the biggest one, *i.e.*, $CPU_{sub} = max(CPU_{sub(l)}),\ l = 1, 2 \cdots h$. Therefore, supposing the biggest subproblem has dimensions $m_s \times n_s$, it needs, from (5.1) and (5.3),

$$F^k_{accpm,sub} = O\left((m_s^2 n_s + m_s^3)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right) \tag{5.17}$$

arithmetic operations to solve the subproblems per outer iteration.

The Overall Complexity of ACCPM

Summing up the complexity for both the master problem and the subproblems, we obtain

$$\begin{aligned} F_{accpm} = \ & F_{accpm,M} + F_{accpm,sub} = F_{accpm,M} + v' F^k_{accpm,sub} \\ = \ & O\left(h\log(h+1)\left[m_0^3 \cdot \tfrac{h(m_0+h)^2}{\varepsilon^2} + m_0^2 \cdot \tfrac{h^3(m_0+h)^4}{\varepsilon^4} + m_0 \cdot \tfrac{h^5(m_0+h)^6}{\varepsilon^6} + \tfrac{h^7(m_0+h)^8}{\varepsilon^8}\right]\right) \\ & + O\left(\tfrac{h(m_0+h)^2}{\varepsilon^2}\right) \cdot O\left((m_s^2 n_s + m_s^3)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right) \end{aligned} \tag{5.18}$$

5.1.3 Relative Complexity of ACCPM vs. IPM

We can use (5.4) and (5.18) to define

$$\begin{aligned} R &= F_{accpm}/F_{ipm} \\ &= \frac{O\left(h\log(h+1)\left[m_0^3\cdot\frac{h(m_0+h)^2}{\varepsilon^2}+m_0^2\cdot\frac{h^3(m_0+h)^4}{\varepsilon^4}+m_0\cdot\frac{h^5(m_0+h)^6}{\varepsilon^6}+\frac{h^7(m_0+h)^8}{\varepsilon^8}\right]\right)}{+O\left(\frac{h(m_0+h)^2}{\varepsilon^2}\right)\cdot O\left((m_s^2n_s+m_s^3)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right)} \\ &\qquad O((hm_s+m_0)^2\cdot hn_s+(hm_s+m_0)^3)\cdot O\left(\sqrt{hn_s}\log(\varepsilon_0/\varepsilon)\right) \end{aligned} \tag{5.19}$$

as the relative efficiency of ACCPM vs. IPM. Clearly, R is a complicated expression. In the following sections, we will discuss this ratio under different circumstances.

Notice that the ratio of worst-case complexities for ACCPM and IPM does not lead to a worst-case R. Instead, in order to understand the behavior of R, F_{accpm} and F_{ipm} should be estimated based on accurate computational models. Unfortunately, such models do not exist. However, it is widely believed that the worst-case results for ACCPM and IPM are much closer to experience than for the simplex method. Therefore, we conduct the following ratio analysis only for the purpose of generating plausible hypotheses about the behavior of the true ratio. We will then test the hypotheses on various numerical examples.

Influence of the Number of Linking Constraints - m_0

On block-angular structured problems, it is widely believed that the more linking constraints, the more computational effort needed for decomposition algorithms to solve them. In this section, we examine the influence on R of the number of linking constraints. Dividing both the numerator and denominator by $h^8m_s^8$, (5.19) becomes

$$\frac{\begin{array}{c}O\left(h\log(h+1)\left[\frac{1}{\varepsilon^2h^2m_s^2}\left(\frac{m_0}{hm_s}\right)^3\left(\frac{m_0+h}{hm_s}\right)^2+\frac{h}{\varepsilon^4m_s^2}\left(\frac{m_0}{hm_s}\right)^2\left(\frac{m_0+h}{hm_s}\right)^4+\frac{h^4}{\varepsilon^6m_s}\frac{m_0}{hm_s}\left(\frac{m_0+h}{hm_s}\right)^6+\frac{h^7}{\varepsilon^8}\left(\frac{m_0+h}{hm_s}\right)^8\right]\right) \\ +O\left(\frac{1}{\varepsilon^2h^5m_s^6}\left(\frac{m_0+h}{hm_s}\right)^2\right)\cdot O\left((m_s^2n_s+m_s^3)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right)\end{array}}{O\left(\frac{n_s}{h^5m_s^6}\left(\frac{m_0}{hm_s}+1\right)^2+\frac{1}{h^5m_s^5}\left(\frac{m_0}{hm_s}+1\right)^3\right)\cdot O\left(\sqrt{hn_s}\log(\varepsilon_0/\varepsilon)\right)} \tag{5.20}$$

Let $k_0=\frac{m_0}{hm_s}$ represent the percentage of linking constraints to nonlinking constraints.

Then the ratio R becomes

$$\frac{O\left(h\log(h+1)\left[\frac{k_0^3}{\varepsilon^2h^2m_s^2}\left(k_0+\frac{1}{m_s}\right)^2+\frac{hk_0^2}{\varepsilon^4m_s^2}\left(k_0+\frac{1}{m_s}\right)^4+\frac{h^4k_0}{\varepsilon^6m_s}\left(k_0+\frac{1}{m_s}\right)^6+\frac{h^7}{\varepsilon^8}\left(k_0+\frac{1}{m_s}\right)^8\right]\right)}{+O\left(\frac{1}{\varepsilon^2h^5m_s^6}\left(k_0+\frac{1}{m_s}\right)^2\right)\cdot O\left((m_s^2n_s+m_s^3)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right)}$$
$$\overline{O\left(\frac{n_s}{h^5m_s^6}\left(k_0+1\right)^2+\frac{1}{h^5m_s^5}\left(k_0+1\right)^3\right)\cdot O\left(\sqrt{hn_s}\log(\varepsilon_0/\varepsilon)\right)} \tag{5.21}$$

If we keep increasing the number of linking constraints (m_0), k_0 will become infinitely large. Therefore, the complexity of ACCPM will increase by a rate of $O(k_0^8)$, while the complexity of IPM will only increase by $O(k_0^3)$. That is, when the ratio of linking constraints to nonlinking (k_0) is high, ACCPM (the decomposition approach) will be slower than IPM (the direct approach - solving the problem as a whole). This is compatible with what people have expected.

Conversely, we consider a problem with very few linking constraints, *i.e.*, $k_0 \to 0$. According to (5.21), we have

$$R_1 = \frac{F_{accpm}}{F_{ipm}} = \frac{F_{accpm,M}+F_{accpm,sub}}{F_{ipm}} = \frac{O\left(\frac{\log(h+1)h^{13}}{\varepsilon^8m_s^2}\right)+O\left(\frac{1}{\varepsilon^2}(n_s+m_s)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right)}{O\left((n_s+m_s)\sqrt{hn_s}\log(\varepsilon_0/\varepsilon)\right)} \tag{5.22}$$

The equation (5.22) looks too complicated to provide any hints. Notice that the second term of the numerator has a similar form with the denominator. Surprisingly, even $k_0 \to 0$, ACCPM can still be quite costly due to the large power over h (the first term in the numerator) compared with IPM. However, if the number of blocks (h) is small enough with respect to m_s and n_s, which is not unusual in practice, ACCPM can hopefully be faster. Therefore, we can assume that $F_{accpm,M} \le F_{accpm,sub}$, *i.e.*,

$$O\left(\frac{\log(h+1)h^{13}}{\varepsilon^8m_s^2}\right) \le O\left(\frac{1}{\varepsilon^2}(n_s+m_s)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right), \tag{5.23}$$

and hence

$$O\left(h^{13}\log(h+1)\right) \le O\left(\varepsilon^6\left(m_s^2n_s+m_s^3\right)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right), \tag{5.24}$$

Without loss of generality, we assume that $m_s \le n_s$. Ignoring the predetermined parameters ε and ε_0, inequality (5.24) becomes

$$h^{13}\log(h+1) \le C_1 n_s^{3.5}, \tag{5.25}$$

where C_1 is a constant to replace the O-notation. This inequality approximately suggests the relationship between h and n_s. Given h satisfies inequality (5.24), together with the assumption $k_0 \to 0$, the ratio R_1 tends to $O\left(\frac{1}{\sqrt{h}}\right)$. This indicates that when the percentage of linking constraints is low, ACCPM (decomposition) will be more advantageous than IPM (direct solution approach) with an increasing h, as long as h is in some *moderate interval*, *i.e.*, satisfying (5.25).

It is worth noting that (5.25) actually means at each outer iteration, the computational effort is dominated by subproblems. At a given iteration, the master problem's number of columns is determined by h, and the number of rows is $m_0 + h$. As a result, when h is smaller, the size of the master problem is smaller. This explains why we have $F_{accpm,M} \leq F_{accpm,sub}$. If considering $F_{accpm,M} \geq F_{accpm,sub}$, which is common in practice, it is difficult to deduce ratio R from (5.22) to a succinct formula.

If we implement ACCPM on a single computer, the ratio becomes

$$R_1 = \frac{O\left(\frac{\log(h+1)h^{13}}{\varepsilon^8 m_s^2}\right) + O\left(\frac{h}{\varepsilon^2}(n_s + m_s)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right)}{O\left((n_s + m_s)\sqrt{hn_s}\log(\varepsilon_0/\varepsilon)\right)} \tag{5.26}$$

In equation (5.26), the second term of the numerator itself (the sum of the subproblems' complexity in ACCPM) is greater than the denominator (the complexity of IPM), which means that *theoretically*, the complexity bound of ACCPM will never beat the complexity bound of IPM in a serial computing environment.

Generally speaking, decomposition methods are computationally complicated. In theory, only in a parallel computing environment, when the linking constraints are very few, and the number of blocks is not too big, can decomposition be faster than solving the problem as a whole. Of course, a main advantage of decomposition lies in dealing with the insufficient memory problem, but this is another issue.

This conclusion also indicates that for a block-angular structured problem, if there are many blocks, *i.e.*, exceeding the inequality ((5.25)), combining some of them may lead to better performance of decomposition.

Influence of the Size of Subproblems - $m_s\&n_s$

Recall (5.18). If m_s keeps increasing, it will be overwhelmingly bigger than all other parameters. Then the complexity of ACCPM becomes

$$F_{accpm} = O\left(h^8 \log(h+1) \cdot \frac{(m_0+h)^8}{\varepsilon^2}\right) + O\left(\frac{h(m_0+h)^2}{\varepsilon^2}\right) \cdot O\left(m_s^{3.5} \log(\varepsilon_0/\varepsilon)\right) \quad (5.27)$$

Notice that only the second term of the above expression contains m_s. Therefore, if m_s is big enough, the second term will be more costly than the first one, *i.e.*,

$$O\left(h^8 \log(h+1) \cdot \frac{(m_0+h)^8}{\varepsilon^2}\right) \leq O\left(\frac{h(m_0+h)^2}{\varepsilon^2} \cdot m_s^{3.5} \log(\varepsilon_0/\varepsilon)\right) \quad (5.28)$$

Assuming $m_0 < h$ and ignoring ε_0 and ε, inequality (5.28) can be further simplified as

$$h^{13} \log(h+1) \leq C_2 m_s^{3.5}, \quad (5.29)$$

where C_2 is a constant to remove the O-notation.

Therefore, when m_s keeps increasing, the computational effort of ACCPM is dominated by the subproblems. According to (5.18), the complexity of ACCPM will eventually become

$$F_{accpm} = O\left(\frac{h(m_0+h)^2}{\varepsilon^2}\right) \cdot O\left(m_s^{3.5} \log(\varepsilon_0/\varepsilon)\right) \quad (5.30)$$

Furthermore, the assumption, $m_0 \leq h$, mentioned in (5.29), is important for decomposition algorithms to be advantageous. With this assumption, the complexity of ACCPM will be

$$F_{accpm} = O\left(\frac{h^3 m_s^{3.5}}{\varepsilon^2} \log(\varepsilon_0/\varepsilon)\right) \quad (5.31)$$

Meanwhile, as m_s increases, the complexity of IPM will become

$$F_{ipm} = O\left(h^3 m_s^3 \sqrt{h m_s} \log(\varepsilon_0/\varepsilon)\right) \quad (5.32)$$

As a result, the ratio becomes

$$R_2 = (5.31)/(5.32) = O\left(\frac{1}{\sqrt{h}}\right) \quad (5.33)$$

Similarly, when increasing n_s, we get the same result on the ratio

$$R_2 = O\left(\frac{1}{\sqrt{h}}\right), \tag{5.34}$$

and the corresponding limitation for h is

$$h^{13}\log(h+1) \leq C_3 n_s^{3.5}, \tag{5.35}$$

where C_3 is a constant to remove the O-notation.

These results tell us that when one of the subproblems is big (m_s or n_s increases), the relative efficiency of ACCPM vs. IPM is inversely proportional to $\sqrt{h}$, given h satisfies inequality (5.29) or (5.35).

Influence of the Number of Blocks - h

Consider a problem with a large number of blocks, *i.e.*, h is very big. Then, according to (5.18), the complexity of ACCPM will become

$$F_{accpm} = O\left(h^8\log(h+1)\cdot\frac{(m_0+h)^8}{\varepsilon^8}\right)+O\left(\frac{h(m_0+h)^2}{\varepsilon^2}\right)\cdot O\left((m_s^2 n_s + m_s^3)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right) \tag{5.36}$$

In the above equation, the first term represents the complexity of the master problem during all the iterations, and the second term is the complexity of the subproblems. If h keeps increasing, the first term will be more costly as it has a higher order over h, *i.e.*,

$$O\left(h^8\log(h+1)\cdot\frac{(m_0+h)^8}{\varepsilon^8}\right) \geq O\left(\frac{h(m_0+h)^2}{\varepsilon^2}\right)\cdot O\left((m_s^2 n_s + m_s^3)\sqrt{n_s}\log(\varepsilon_0/\varepsilon)\right), \tag{5.37}$$

which means that the computational effort is dominated by the master problem now. Since $h \geq m_0$, with a further assumption $n_s \geq m_s$, this expression becomes

$$h^{13}\log(h+1) \geq C_3 n_s^{3.5}, \tag{5.38}$$

where C_3 is a constant to remove the O-notation. Given (5.38) holds, the ACCPM complexity can be denoted as

$$F_{accpm} = O\left(h^8\log(h+1)\cdot\frac{(m_0+h)^8}{\varepsilon^8}\right) \tag{5.39}$$

According to (5.4), as h increases, the complexity of IPM will become

$$F_{ipm} = O\left(h^7 \log(\varepsilon_0/\varepsilon)\right) \tag{5.40}$$

Therefore, the relative efficiency ratio can be calculated as

$$R_3 = (5.39)/(5.40) = O\left(h^9 \log(h+1)\right) \tag{5.41}$$

This result indicates that when h keeps increasing and eventually becomes big enough to satisfy (5.38), ACCPM results in huge computational effort, and it is a better choice to solve the problem as a whole by IPM.

5.2 Empirical Analysis for ACCPM vs. IPM

5.2.1 Limitations of Theoretical Analysis

In the previous section, we deduced the complexity of ACCPM and IPM on problems with a block-angular structure. However, there are a few inevitable drawbacks in theoretical analysis:

1. Worst case vs. average case. All the complexity conclusions are from the worst-case, which may not be able to represent an algorithm's real performance. In fact, even the average-case analysis cannot tell us how an algorithm really works in practice. There is a gap between theory and practice.

2. A number of assumptions are made to build the ideal mathematical models. For example, the conclusions in Section 5.1.2 assume that all the cuts are 'central', which is not true in real implementation.

3. Decomposition algorithms are usually implemented in distributed systems. In our discussion, we assume that the master problem is assigned to one processor, and each subproblem is assigned to one processor as well. In fact, it is very difficult to have plenty of processors to satisfy such an assumption. For example, the Flexor system in the University of Waterloo only has 52 processors. In our analysis, we also ignore the communication overhead, while in practice, that is an important factor to consider when evaluating algorithms' performance.

4. The models are analyzed in extreme cases. For example, when we consider a problem with fewer linking constraints, we assume that the ratio $k_0 = m_0/hm_s$ tends to zero; when we consider a problem with many blocks, we let h be infinity. Then, some conclusions are drawn based on these unreal extreme assumptions.

5. All the constants C_1 through C_5 in the theoretical O-notation are neglected. However, having a parameter tend to infinity is only a mathematical concept, which never happens in real life. Therefore, if some parameter is quite big but not enough to 'tend to infinity', those constants will count.

6. Some other factors are omitted in our theoretical analysis, such as the density of the coefficient matrix. There are also some other factors which affect the overall performance but are too hard to be addressed. For example, some problems do not have a huge size, but result in poor convergence of an algorithm [82]. Another example is that we often estimate the effort to solve a problem based on their dimensions, but with a warm start, the complexity can be greatly reduced. The mystery of optimization has not been completely figured out yet.

5.2.2 Empirical Analysis on IPM

In order to better investigate algorithms' performance, we do empirical analysis over a large set of test problems.

IPM Number of Iterations

We first test the theoretical conclusions of IPM using empirical methods. Figure 5.1 shows the number of iterations in response to the number of columns for both models. The dots in the figure represent observed results, and the curve is based on the regression model (5.42) below. As discussed before, IPM has an observed average behavior of $\log(\overline{n})$, where

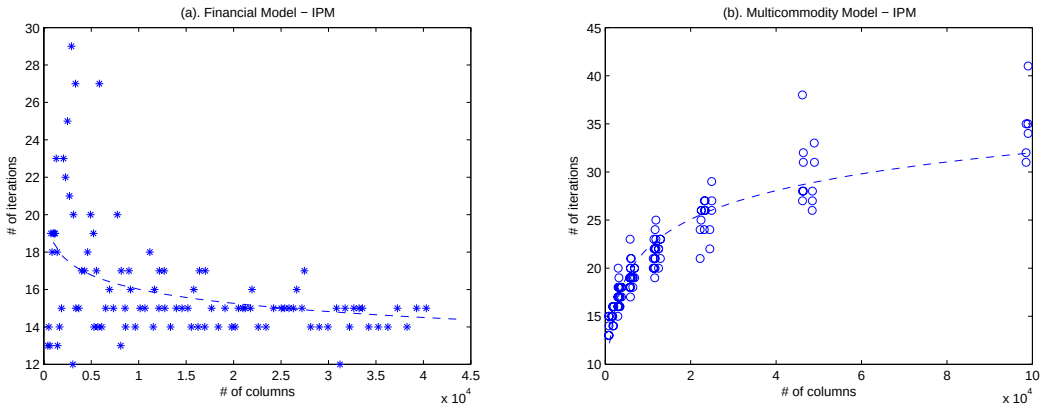


Figure 5.1: IPM number of iterations vs. number of columns for Both Models

$\overline{n}$ is the total number of columns[1]. Therefore, we build the following regression model

$$Iter = \alpha + \beta \log(\overline{n}) \tag{5.42}$$

Regression results are summarized in Table 5.1 on page 92. The regression analysis is done separately on three test sets: f represents the financial problems, m stands for the multicommodity problems, and b means both. Under each coefficient in the table, there is a corresponding p value, which is obtained from a two-sided t-test with the null assumption H_0: the coefficient is zero. We use 5% as the significance level, *i.e.*, if $p < 5\%$, the null assumption is rejected; otherwise, a p value more than 5% means an insignificant parameter in the regression model. The coefficient of determination, r^2, which reveals the goodness of fitting, is also provided in the table. Big residuals will lead to a poor r^2, which suggests uncovered parameters in the model.

The results in Table 5.1 are quite *model dependent*. Firstly, according to r^2s, (5.42) is a bad fit on test set f, but is very good on test set m. No parameters seem redundant on both of them. If we arbitrarily put the two test sets together (denoted as b), the regression results are still poor. Notice that although α has a high p value on set b, we still keep it in the model because it is a constant. Secondly, a negative β appears on set f, which is quite

[1]See equation (5.2). The parameters ε_0 and ε are ignored here for two reasons: one, there is little we can do about the initial duality gap as we use the artificial variables as the first starting point; two, the final precision is a constant in our experiments.

unexpected. People commonly believe that the larger the problem is, the more iterations it needs. Since the r^2 on set f is as low as 0.16, the regression model seems inappropriate and does not tell us much. In addition, recall the problem characteristics in Table 2.1. Notice that all the financial problems have very low $m_0\%$ (up to 8%). Therefore, the negative β may imply that when the problem is sparse, the number of iterations is very insensitive to the scale.

Due to the poor fit of (5.42) on f, we try a model with $m_0\%$

$$Iter = e^{\alpha}\overline{n}^{\beta}m_0\%^{\gamma} \tag{5.43}$$

The r^2 of test set f is considerably improved, although still not ideal.

Similarly, we try another model with $nz\%$

$$Iter = e^{\alpha}\overline{n}^{\beta}nz\%^{\gamma} \tag{5.44}$$

The r^2 of test set f is further improved to 0.5575, while the r^2 for test set m is as high as 0.9250.

Next, we use the four independent inner-structure parameters, m_0, h, m_s and n_s, to do the regression. The r^2s reach the highest values for both f and m in Table 5.1. For (5.45m), only the parameter n_s seems redundant.

$$Iter = e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon} \tag{5.45}$$

IPM CPU Time

Next, we examine how IPM's CPU time varies with a problem's dimensions. Figure 5.2 shows the CPU time of IPM vs. the number of columns in a problem. Since the points seem scattered, Figure 5.3 takes logarithms on both axes.

The first regression model that we test is the same as the one in the previous analysis of iteration counts

$$CPU = \alpha + \beta\log(\overline{n}) \tag{5.46}$$

Regression results in this section are provided in Table 5.2 on page 93. Model (5.46) has very low r^2s on both sets f and m, as well as the set with putting the two together. Accordingly, we try another model with $m_0\%$

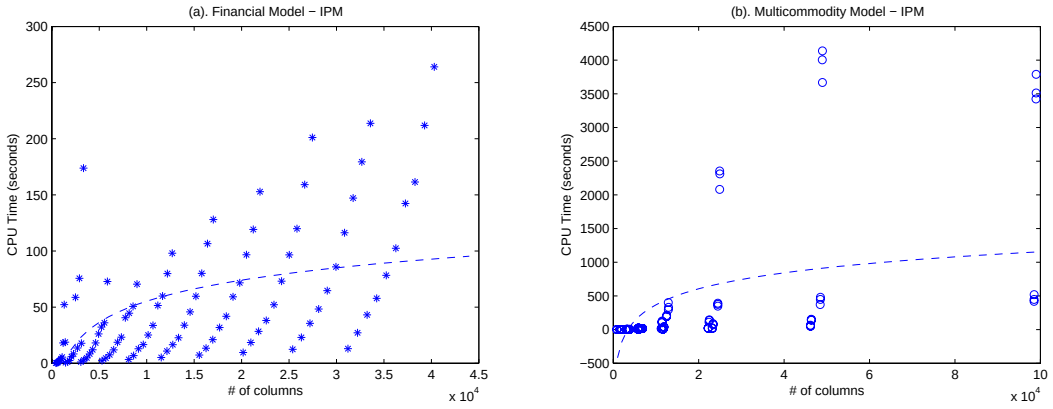


Figure 5.2: IPM CPU vs. number of columns for Financial and Multicommodity Models

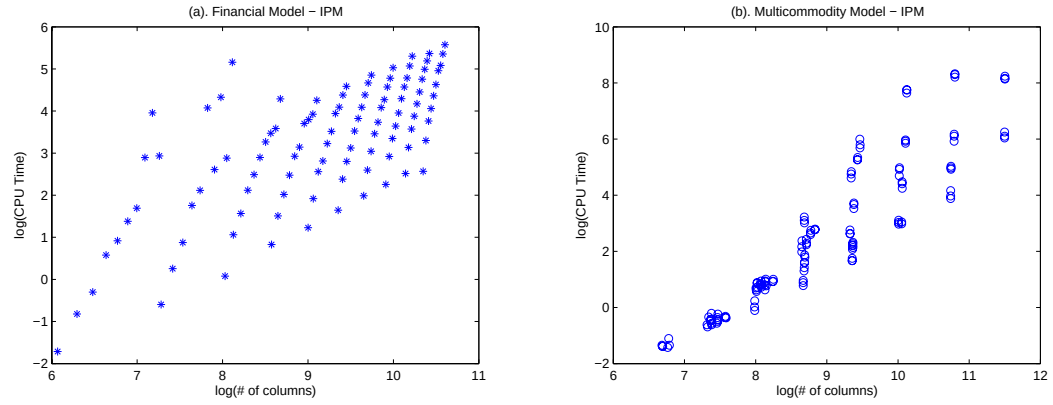


Figure 5.3: IPM log(CPU) vs. log(n) for Financial and Multicommodity Model

$$CPU = e^{\alpha}\overline{n}^{\beta}m_0\%^{\gamma} \tag{5.47}$$

In the regression results for both test sets f and m, r^2s are significantly improved. Similarly, we look at another following model with $nz\%$, and the regression results show good fits for both sets.

$$CPU = e^{\alpha}\overline{n}^{\beta}nz\%^{\gamma} \tag{5.48}$$

Lastly, we try the regression model with the inner-structure parameters

$$CPU = e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon} \tag{5.49}$$

The r^2s for both f and m are the highest in Table 5.2.

5.2.3 Empirical Analysis on ACCPM

ACCPM Convergence

Recall the theoretical bound (5.6), which suggests that the number of cuts that ACCPM requires is determined by the number of cuts generated per iteration (h in our case), the number of columns ($\overline{n}$), and the tolerance (ε). Ignoring the constant ε, our first regression model is then built based on this conclusion

$$\#prop = e^{\alpha}h^{\beta}\overline{n}^{\gamma} \tag{5.50}$$

Regression results are summarized in Table 5.3 on page 94, where r^2s show that model (5.50) fits well on both test sets f and m, but the variable h seems redundant. Both β and γ take positive values, but are much less than '2' of (5.6), which is the worst-case complexity.

Taking into consideration $m_0\%$ and $nz\%$, the following two regression models further improved r^2 on both sets f and m than (5.50).

$$\#prop = e^{\alpha}h^{\beta}\overline{n}^{\gamma}(m_0\%)^{\delta} \tag{5.51}$$

$$\#prop = e^{\alpha}h^{\beta}\overline{n}^{\gamma}(nz\%)^{\delta} \tag{5.52}$$

Taking the block-angular structure into mind, we can try another with the four inner-structure parameters

$$\#prop = e^{\alpha} m_0^{\beta} h^{\gamma} m_s^{\delta} n_s^{\epsilon} \tag{5.53}$$

The r^2s of models $(5.53f)$ and $(5.53m)$ are the highest in Table tab:accpm.prop.

ACCPM CPU Time

Similarly, we introduce the following regression model to estimate the ACCPM CPU time in response to the number of blocks and the number of columns

$$CPU = e^{\alpha} h^{\beta} \overline{n}^{\gamma} \tag{5.54}$$

The regression results in this section are summarized in Table 5.4. With $m_0\%$ or $nz\%$, the fit can be somewhat improved by models (5.55) and (5.55)

$$CPU = e^{\alpha} h^{\beta} \overline{n}^{\gamma} (m_0\%)^{\delta} \tag{5.55}$$

$$CPU = e^{\alpha} h^{\beta} \overline{n}^{\gamma} (nz\%)^{\delta} \tag{5.56}$$

Next, we try a regression model based on the four inner-structure parameters.

$$CPU = e^{\alpha} m_0^{\beta} h^{\gamma} m_s^{\delta} n_s^{\epsilon} \tag{5.57}$$

The r^2 of $(5.57f)$ and $(5.57m)$ are good, but not the best in the table. Moreover, a few variables are redundant according to the p values.

5.2.4 Empirical Analysis on the Ratio - Relative Efficiency

Overview of Data

As mentioned earlier, there are only two parameters in the stochastic financial model: the number of assets and the number of scenarios. This enables us to observe its performance by 3D figures.

On the stochastic financial problems, Figure 5.4 reveals the CPU time of ACCPM and IPM, respectively. Both the surfaces in the figures increase along the number of assets as

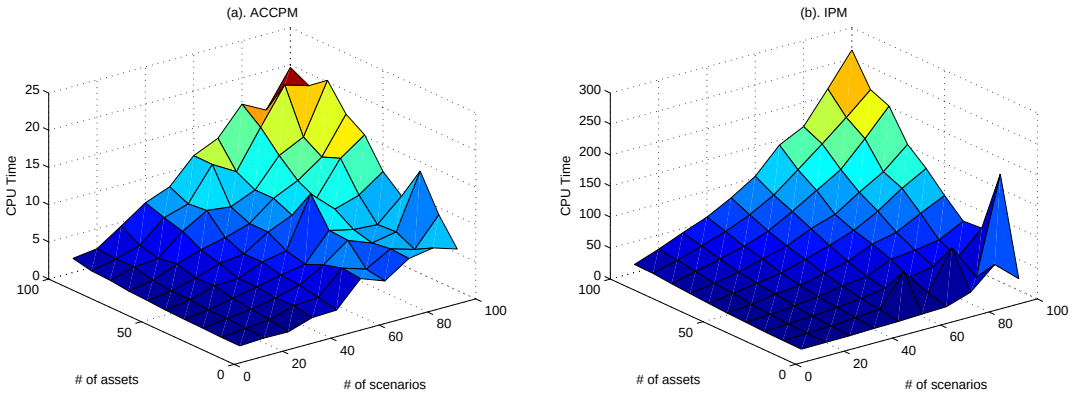


Figure 5.4: CPU Time vs. Number of Assets and Number of Scenarios (Financial Model)

well as the number of scenarios. If we look at the scale of the vertical axes, we see that the CPU time of IPM, the direct solution approach, increases more than ACCPM, the decomposition approach. Similarly, Figure 5.5 shows how the ratio of ACCPM vs. IPM varies along the two parameters using a mesh plot as well as a contour plot. Figure 5.6 is the cutaway views of Figure 5.5, so that we can see how the ratio varies along one parameter with another fixed.

The multicommodity model has more than two parameters, and hence is more difficult to visualize. We choose two parameters, $m_0\%$ and h, to do the 3D plottings. The reason that we choose these two parameters is that for problems with a block-angular structure, the percentage of linking constraints has long been considered important; when we evaluate the relative efficiency of ACCPM and IPM, the number of h means the number of processors, which affects the decomposition solution time.

Figure 5.7 shows the CPU time of ACCPM and IPM, respectively. Generally speaking, the higher h, the bigger the problem; the higher $m_{\%}$, the denser the problem. Therefore, the two planes indicating ACCPM and IPM solution time increase with both h and $m_0\%$. Similarly, Figure 5.8 reveals the tendency of the ratio (ACCPM/IPM) along $m_0\%$ and h. Most of the stems in the figure are short, but we still can tell that with higher $m_0\%$, ACCPM takes longer than IPM.

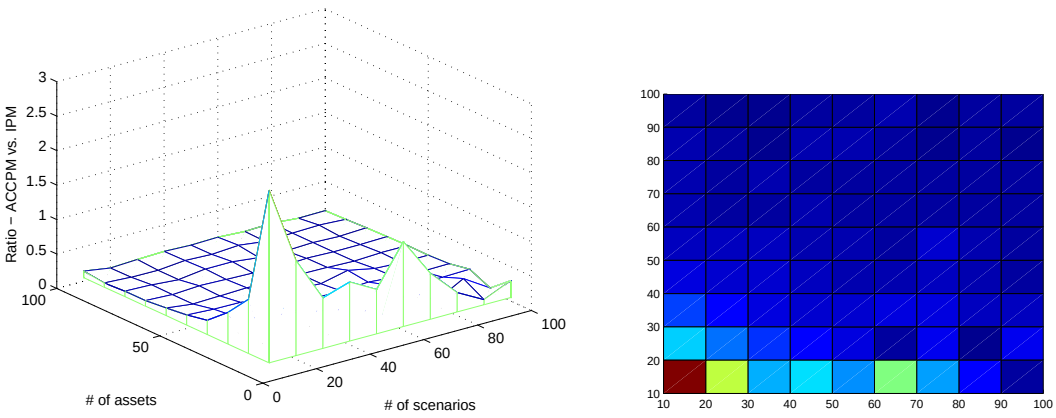


Figure 5.5: Ratio (ACCPM vs. IPM) and Surface Map for Financial Model

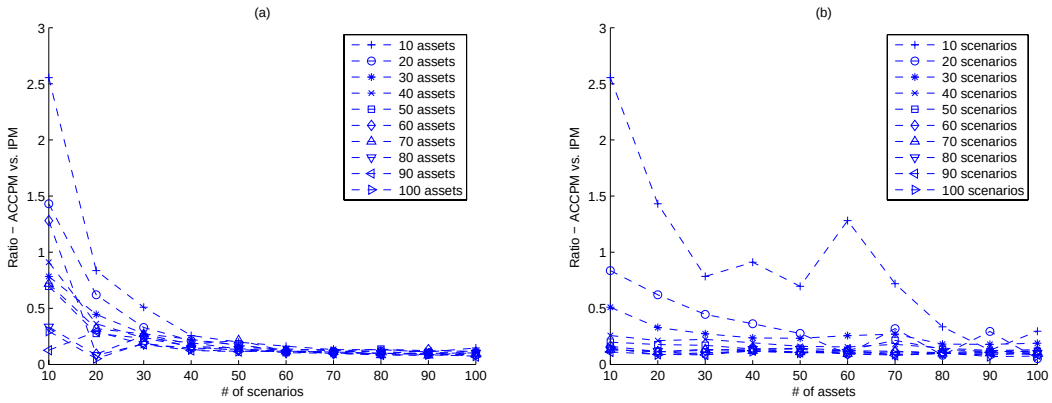


Figure 5.6: Cutaway view for the Ratio (Financial Model)

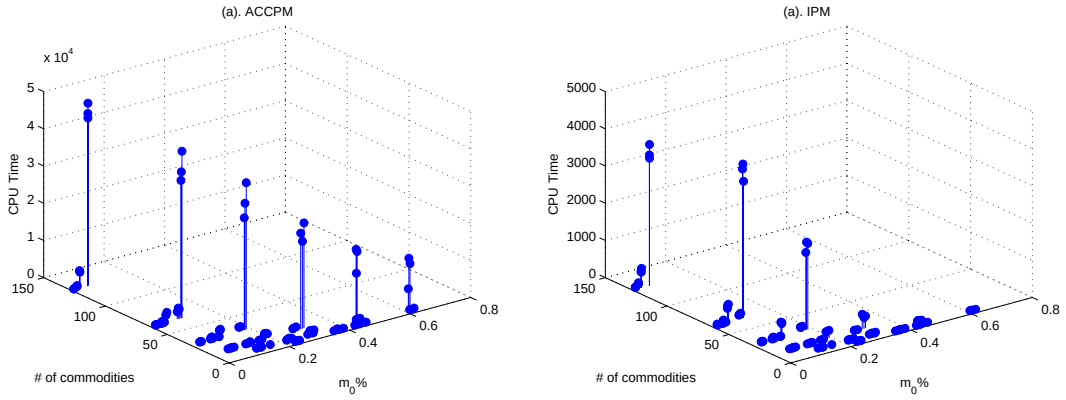


Figure 5.7: CPU Time vs. $m_0\%$ and h (Multicommodity Model)

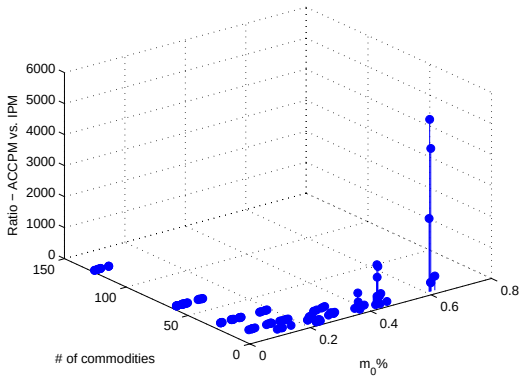


Figure 5.8: Ratio (ACCPM vs. IPM) for Multicommodity Model

Regression Analysis on R

It is difficult to build regression models on the ratio of ACCPM and IPM as there are more factors involved. After observing the tendencies shown in the figures in the previous section, we first test the following regression model

$$R = e^{\alpha}(m_0\%)^{\beta}h^{\gamma} \tag{5.58}$$

Regression results are summarized in Table 5.5 on page 96. For the test set f, both β and γ take negative values, while for the test set m, they both take positive values. We have expected positive $m_0\%$ and negative h, implying that the ratio is proportional to $m_0\%$ but inversely proportional to h. That is, the higher $m_0\%$, the less advantageous decomposition algorithms; the bigger h, the faster decomposition algorithms.

When examining the ratio R, we recall what factors would influence the CPU time of ACCPM and IPM, respectively, so that we can find a clue to build regression models. In addition to $m_0\%$ and h, $\overline{n}$ seems important to the complexity of IPM. We then introduce the following model

$$R = e^{\alpha}h^{\beta}\overline{n}^{\gamma}(m_0\%)^{\delta} \tag{5.59}$$

The r^2s for both f and m are somewhat improved, but still not ideal.

Our next regression model contains $(m_0{+}h)$, $\overline{n}$, and $m_0\%$, which represent the number of rows of the master problem, the total number of columns, and one of the density indicators.

$$R = e^{\alpha}(m_0 + h)^{\beta}\overline{n}^{\gamma}(m_0\%)^{\delta} \tag{5.60}$$

This model achieves higher r^2s, but also results in more redundancy.

Similarly, we can use $nz\%$ to replace $m_0\%$

$$R = e^{\alpha}(m_0 + h)^{\beta}\overline{n}^{\gamma}(nz\%)^{\delta}, \tag{5.61}$$

and the fitting results are almost the same as (5.60).

Lastly, we try a regression model with the inner-structure parameters

$$R = e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon} \tag{5.62}$$

The r^2s are comparable with the results of other models in the table, but the redundancy is severe, especially for test set f.

Prediction

In this section, we investigate whether these regression results are reliable to be used as guidance in practice. We use *robust regression* to re-assess the parameters in model (5.63). It is commonly believed that robust regression can lead to a better fit especially for prediction purposes.

$$R = e^{\alpha} m_0^{\beta} h^{\gamma} n_s^{\delta} \tag{5.63}$$

Random Prediction We randomly select ten test problems from the stochastic financial model and from the multicommodity network flow model. Then, the rest of the test problems are used as data for the robust regression. The robust regression results are given in Table 5.6. The weights assigned to the variables on both test sets are provided in Tables C.5 and C.6 on page 182.

In Section 5.1.3, we have obtained the ratio of ACCPM and IPM: theoretically, with a certain dimensional parameter tending to infinity, R tends to $\frac{1}{\sqrt{h}}$. As mentioned before, in such cases, one parameter becomes overwhelmingly big so that other parameters can be neglected, which is unrealistic. An algorithm's performance on real-world problems is often difficult to analyze as there are complex joint-effect involved. Comparing regression results in Table 5.6, we see that γ takes negative values on both test sets, which is consistent with the theoretical conclusion, $\frac{1}{\sqrt{h}}$. However, other parameters are significant, too.

Table 5.7 shows the validation of model (5.63) on financial problems. Based on the observed and predicted data, we evaluate the errors in two ways: one, if the observed and predicted results suggest the same pick (simplex or DW), the *pick*% takes value 1, otherwise, 0; two, we examine the percentage of errors defined as $\frac{|\hat{R}-R|}{R}$%. Experiments show that *pick*% is 100% for the 10 randomly chosen financial problems. Similarly, Table 5.8 shows the results on 10 multicommodity problems, for which *pick*% is 90% valid.

Extrapolation Our analysis aims at *large-scale* problems, so the ability of model extrapolation is important. Meanwhile, we want the test problems to be solved in a reasonable period of time. Therefore, rather than generating new larger problems, we pick the largest 10 problems from the current sets f and m respectively as the ones to be tested, and use the

rest to do the robust regression. Again, we test the inner-structure regression model (5.63). The regression results are given in Table 5.9. The weights assigned to the variables on both test sets are provided in Tables C.7 and C.8 on page 184.

The extrapolation tests are summarized in Tables 5.10 and 5.11. In both tables, the problems are arranged in ascending order in terms of $\overline{n}$. We obtained 100% accuracy of *pick*% on the 20 problems from both test sets.

5.3 Preliminary Conclusions

We tried many regression models in our experiments, and only reported some of the representative ones in this thesis. In order to analyze numerical results, building appropriate regression models is crucial but difficult. For most of the models, we obtained satisfactory fits, but there still could be better ones that we did not try. We summarize this chapter with the following points:

1. On sparse problems with a block-angular structure, the traditional overall dimension, $\overline{n}$, together with the density indictors, $m_0\%$ or $nz\%$, generate satisfactory models.

2. Regression results tend to be model dependent. This is probably due to the difference of actual characteristics between the two test sets. For example, $m_0\%$ of the financial problems are all low, in which case the coefficient of $m_0\%$ in a regression model may take a value with an opposite sign as we expected.

3. The number of blocks h frequently appear redundant when considering the relative efficiency of with and without decomposition (ACCPM vs. IPM), which confirms that IPM is insensitive to the problem size. That is, although many (h) processors are used by ACCPM, the speedup factor to IPM is not significant.

4. The four 'inner-structure' parameters, m_0, h, m_s and n_s are easy to use, and have good regression results on all the models. However, on some models, the redundancy is severe. Mathematically, the four inners-structure parameters are independent, but in practice, they usually have economic implications, and this kind of correlation most likely occurs between the block dimensions m_s and n_s. In this case, we can

keep one of them in the regression model. For example, in the prediction analysis, there was no redundancy in the models.

5. Prediction and extrapolation tests on selected regression models reveal high accuracy, which proves that empirical analysis is a viable way in practice to predict the relative efficiency of ACCPM and IPM based on problem characteristics.

6. To some extent, our empirical results agree with the theoretical conclusions. But evidently, the empirical results can provide more realistic and reliable information for evaluating an algorithm's performance.

Table 5.1: Regression Results for IPM Iteration Counts

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)*					
$\alpha + \beta \log(\overline{n})$	f	26.0409	-1.0885				0.1615
	m	1.2525	0.1965				0.9193
(5.42)	b	1.0519 (0.6590)	1.9756				0.1988
$e^{\alpha}\overline{n}^{\beta}m_0\%^{\gamma}$	f	2.8318	0.0779	0.1555			0.4757
(5.43)	m	1.2718	0.1928	-0.0065			0.9198
$e^{\alpha}\overline{n}^{\beta}nz\%^{\gamma}$	f	1.9576	0.4875	-0.9479			0.5575
(5.44)	m	1.3101	0.1314	-0.0774 (0.4214)			0.9250
$e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon}$	f	5.5154	-0.2631	0.7715	7.2347	-7.7373	0.6496
(5.45)	m	1.2223	0.0721	0.1843	0.1304	0.0418 (0.1351)	0.9360

*Most variables have very low p values, and here we only indicate those with $p \geq 0.05$.

Table 5.2: Regression Results for IPM CPU Time

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)*					
$\alpha + \beta \log(\overline{n})$	f	-194.1474	27.0763				0.2999
	m	-2783	342.1				0.2488
(5.46)	b	-1690.6	207.5				0.1487
$e^{\alpha}\overline{n}^{\beta}m_0\%^{\gamma}$	f	-10.3231	2.3436	1.5716			0.9351
(5.47)	m	-17.4491	2.4638	0.9664			0.9195
$e^{\alpha}\overline{n}^{\beta}nz\%^{\gamma}$	f	-11.0865	2.4711	1.6775			0.9381
(5.48)	m	-16.5897	4.2050	2.7294			0.9002
$e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon}$	f	-10.0684	0.6609	0.6976 (0.3127)	7.6868 (0.0753)	-5.7601 (0.2313)	0.9598
(5.49)	m	-17.0444	1.3548	1.5600	-0.1525 (0.5813)	1.3620	0.9198

*Most variables have very low p values, and here we only indicate those with $p \geq 0.05$.

Table 5.3: Regression Results for ACCPM Total Number of Cuts

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)*					
$e^{\alpha}h^{\beta}\overline{n}^{\gamma}$	f	0.2248 (0.2522)	0.1049 (0.3149)	0.4332			0.9569
(5.50)	m	-3.0701	0.1296 (0.1405)	0.9257			0.8805
$e^{\alpha}h^{\beta}\overline{n}^{\gamma}(m_0\%)^{\delta}$	f	-0.4280	-0.6157	0.7110	-0.1730		0.9684
(5.51)	m	-0.9645	1.0434	0.5422	0.6486		0.9206
$e^{\alpha}h^{\beta}\overline{n}^{\gamma}(nz\%)^{\delta}$	f	-0.3713 (0.0612)	-0.6391	0.7079	-0.1879		0.9684
(5.52)	m	-2.7367	0.3240	1.3608	0.7088		0.8924
$e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon}$	f	-3.7413	0.1205 (0.0539)	-0.8499	-6.7832	8.5199	0.9748
(5.53)	m	-1.8732	0.8956	0.9930	0.2296	-0.2581	0.9320

*Most variables have very low p values, and here we only indicate those with $p \geq 0.05$.

Table 5.4: Regression Results for ACCPM CPU Time

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)*					
$e^{\alpha}h^{\beta}\overline{n}^{\gamma}$	f	-12.5381	-6.1016	4.0774			0.8022
(5.54)	m	-21.6393	-3.5039	4.1874			0.7694
$e^{\alpha}h^{\beta}\overline{n}^{\gamma}(m_0\%)^{\delta}$	f	-8.5867	-1.7393	2.3957	1.0475		0.9494
(5.55)	m	-13.9862	-0.1828 (0.5064)	2.7933	2.3575		0.9113
$e^{\alpha}h^{\beta}\overline{n}^{\gamma}(nz\%)^{\delta}$	f	-8.8961	-1.5557	2.3985	1.1485		0.9522
(5.56)	m	-20.5623	-2.8759	5.5927	2.2893		0.8026
$e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon}$	f	-7.6286	0.9437	-0.1386 (0.7819)	1.0981 (0.7245)	0.0784 (0.9820)	0.9495
(5.57)	m	-15.3815	0.4369	3.4591	0.1303 (0.4640)	-0.0722 (0.7246)	0.9678

*Most variables have very low p values, and here we only indicate those with $p \geq 0.05$.

Table 5.5: Regression Results for the Ratio of ACCPM vs. IPM

Regression Model	Test Set	α	β	γ	δ	ϵ	r^2
		(p value for each coefficient)*					
$e^{\alpha}(m_0\%)^{\beta}h^{\gamma}$	f	1.6593	-0.4053	-1.4385			0.7001
(5.58)	m	5.5405	1.4388	0.1459 (0.3707)			0.6473
$e^{\alpha}h^{\beta}\overline{n}^{\gamma}(m_0\%)^{\delta}$	f	5.4004	1.4288	-1.4019	-0.0288 (0.7591)		0.7843
(5.59)	m	1.2400 (0.3101)	-0.9476	0.7385	1.0021		0.6799
$e^{\alpha}(m_0+h)^{\beta}\overline{n}^{\gamma}(m_0\%)^{\delta}$	f	4.4798	-1.1001	-0.0230 (0.9178)	0.1953 (0.1781)		0.7894
(5.60)	m	0.2712 (0.6218)	4.0777	-2.5847	-1.1350		0.8578
$e^{\alpha}(m_0+h)^{\beta}\overline{n}^{\gamma}(nz\%)^{\delta}$	f	4.4088	-1.1703	0.0438 (0.8567)	0.2418 (0.1308)		0.7905
(5.61)	m	1.2384	2.8876	-2.6677	-1.3291		0.8484
$e^{\alpha}m_0^{\beta}h^{\gamma}m_s^{\delta}n_s^{\epsilon}$	f	2.4398 (0.2833)	0.2829 (0.1941)	-0.8362 (0.2561)	-6.5887 (0.1511)	5.8385 (0.2543)	0.7930
(5.62)	m	1.6629 (0.0894)	2.1042	-1.1231	0.2828 (0.3480)	-1.4342	0.7845

*Most variables have very low p values, and here we only indicate those with $p \geq 0.05$.

Table 5.6: Robust Regression Results for Random Prediction Tests

Problem	Regression Coefficients				p values for				$dof*$
	α	β	γ	δ	α	β	γ	δ	
f	3.8045	-0.1060	-0.4444	-0.6564	< 0.05	0.1091	< 0.05	< 0.05	86
m	2.8707	2.0201	-1.0399	-1.4009	< 0.05	< 0.05	< 0.05	< 0.05	118

*Degrees of freedom for error

Table 5.7: Prediction Experiments on Random Chosen Financial Problems

Problem		Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
80	90	9.85E-02	ACCPM	8.13E-02	ACCPM	7.53E-02	8.77E-02	17.45%	1
100	20	5.19E-02	ACCPM	2.58E-01	ACCPM	2.27E-01	2.93E-01	396.41%	1
80	30	1.81E-01	ACCPM	2.12E-01	ACCPM	1.98E-01	2.27E-01	17.02%	1
20	70	1.13E-01	ACCPM	1.38E-01	ACCPM	1.27E-01	1.50E-01	22.47%	1
40	100	1.16E-01	ACCPM	8.50E-02	ACCPM	7.88E-02	9.16E-02	26.94%	1
100	30	1.89E-01	ACCPM	1.93E-01	ACCPM	1.77E-01	2.10E-01	1.98%	1
100	10	2.96E-01	ACCPM	4.01E-01	ACCPM	3.14E-01	5.13E-01	35.69%	1
50	10	6.96E-01	ACCPM	5.90E-01	ACCPM	5.18E-01	6.71E-01	15.26%	1
90	90	1.08E-01	ACCPM	7.88E-02	ACCPM	7.27E-02	8.55E-02	26.84%	1
10	50	1.99E-01	ACCPM	2.17E-01	ACCPM	1.93E-01	2.44E-01	9.19%	1
Average								56.92%	100%

Table 5.8: Prediction Experiments on Random Chosen Multicommodity Problems

Problem			Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	$No.$	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
128	32	2	3.73	IPM	3.39	IPM	2.59	4.45	8.93%	1
64	64	10	17.67	IPM	14.03	IPM	10.32	19.08	20.58%	1
128	128	2	2.67	IPM	0.82	ACCPM	0.57	1.19	69.16%	0
128	4	1	69.35	IPM	33.03	IPM	24.34	44.83	52.37%	1
64	64	7	7.05	IPM	3.08	IPM	2.44	3.90	56.24%	1
128	64	3	1.17	IPM	1.55	IPM	1.12	2.13	32.61%	1
128	4	9	162.57	IPM	69.96	IPM	48.12	101.72	56.97%	1
64	16	2	3.61	IPM	4.70	IPM	3.52	6.27	30.07%	1
128	16	4	21.50	IPM	29.03	IPM	24.69	34.14	35.05%	1
128	64	2	2.42	IPM	1.74	IPM	1.28	2.35	28.21%	1
Average									39.02%	90%

Table 5.9: Robust Regression Results for Extrapolation Tests

Problem	Regression Coefficients				p values for				$dof*$
	α	β	γ	δ	α	β	γ	δ	
f	4.5602	-0.0539	-0.2661	-0.9703	< 0.05	0.3861	0.0700	< 0.05	86
m	2.9678	2.0586	-1.0669	-1.4370	< 0.05	< 0.05	< 0.05	< 0.05	118

*Degrees of freedom for error

Table 5.10: Extrapolation Experiments on Financial Problems

Problem		Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
90	90	1.08E-01	ACCPM	7.45E-02	ACCPM	6.82E-02	8.14E-02	30.83%	1
30	100	1.04E-01	ACCPM	8.35E-02	ACCPM	7.64E-02	9.12E-02	20.03%	1
100	90	8.58E-02	ACCPM	7.21E-02	ACCPM	6.58E-02	7.91E-02	16.00%	1
40	100	1.16E-01	ACCPM	7.99E-02	ACCPM	7.37E-02	8.65E-02	31.32%	1
50	100	1.02E-01	ACCPM	7.68E-02	ACCPM	7.10E-02	8.31E-02	24.79%	1
60	100	1.10E-01	ACCPM	7.40E-02	ACCPM	6.83E-02	8.02E-02	32.97%	1
70	100	8.97E-02	ACCPM	7.15E-02	ACCPM	6.57E-02	7.78E-02	20.30%	1
80	100	1.02E-01	ACCPM	6.92E-02	ACCPM	6.33E-02	7.56E-02	32.16%	1
90	100	6.99E-02	ACCPM	6.70E-02	ACCPM	6.11E-02	7.36E-02	4.09%	1
100	100	7.45E-02	ACCPM	6.51E-02	ACCPM	5.90E-02	7.17E-02	12.58%	1
Average								22.51%	100%

Table 5.11: Extrapolation Experiments on Multicommodity Problems

Problem			Observed		Predicted		95% CI for $\hat{R}$		errors	
n	h	$No.$	R	$pick$	$\hat{R}$	$\widehat{pick}$	lb	ub	$\frac{\lvert\hat{R}-R\rvert}{R}\%$	$pick\%$
128	64	7	5.41	IPM	5.67	IPM	4.06	7.93	4.83%	1
128	64	12	9.24	IPM	22.25	IPM	15.46	32.02	140.84%	1
128	64	10	9.50	IPM	22.79	IPM	15.80	32.87	139.90%	1
128	64	11	12.21	IPM	23.79	IPM	16.43	34.45	94.84%	1
128	128	9	10.33	IPM	2.59	IPM	1.73	3.88	74.89%	1
128	128	8	10.79	IPM	2.67	IPM	1.79	3.99	75.22%	1
128	128	7	8.04	IPM	2.86	IPM	1.92	4.26	64.47%	1
128	128	11	14.31	IPM	11.21	IPM	7.31	17.18	21.71%	1
128	128	12	11.89	IPM	11.30	IPM	7.36	17.34	4.94%	1
128	128	10	13.17	IPM	11.04	IPM	7.21	16.91	16.17%	1
Average									63.78%	100%

Chapter 6

A Hybrid Solution Approach Combining ACCPM and DW

6.1 Introduction

Convergence speed and accuracy are important goals for the design of an algorithm. Yet, in linear programming, these two goals create a dilemma for anyone making a choice between the two main solution approaches: the simplex method and interior point method (IPM). Proposed by Karmarkar [51], IPMs take polynomial time to find a near-optimal solution. This is undoubtedly an advantage compared to the simplex method with exponential worst-case complexity. IPMs move inside the feasible region, and repel the current point away from the boundaries of the feasible region. This enables IPMs to make considerable progress in the next iteration [5], but also limits the accuracy of the final solution. The simplex method, in contrast, can reach an optimal corner point and hence achieve better accuracy. For years, researchers have been intrigued by the possibility of combining the two methods (e.g., [8][58][3][59]). The attempts usually run IPM first, and then simplex, in the hopes of making significant progress at the beginning as well as being able to reach a precise solution. This is also the motivation of our research.

Large-scale real-world problems often have special structures that can be well exploited by decomposition algorithms. Sometimes, decomposition can be the only way to deal with huge problems in terms of memory limit and solution time. Dantzig-Wolfe (DW)

decomposition [20][22] is a method that has been widely used to solve problems with a block-angular structure, which is also the focus of this chapter. The DW method can be understood as an extension of the simplex method, and therefore it inherits both the benefits and drawbacks of simplex.

When looking at the DW master problem from the dual space, we observe that it has an analogous form with cutting plane methods. Adding a proposal to the primal master problem is equivalent to adding a cut to the dual space. By choosing different points to generate cuts, different variants of cutting plane methods have been proposed. The earliest one, Kelley's cutting plane method [52], corresponds exactly to the dual DW master problem. The main difficulty with cutting plane methods lies in the calculation of the centers of polyhedrons. For example, calculating the gravity center can be more expensive than the optimization problem itself [24]. To overcome this difficulty, an *analytic center* was defined [79][80] and was proved to be highly efficient. The analytic center cutting plane method (ACCPM) uses concepts from interior point literature, and its convergence rate is very insensitive to the problem scale [41]. ACCPM can also be applied to the decomposition area [34]. It has good performance in practice due to the fact that, compared to the *marginal* price in DW, a *central* price embodies the information of all the cuts (proposals) accumulated up to that point [35].

While ACCPM has good convergence properties, its accuracy is still not ideal. On some test problems in [57], ACCPM could barely reach a 10^{-3} relative precision. Researchers have been trying to improve the algorithm, by using methods such as the partial Lagrangian relaxation and column elimination [4]. These techniques can help control the size of the master problem, and make the algorithm more efficient. However, in some situations, the convergence rate is not satisfactory: in [4], the ACCPM variant takes hundreds, even thousands, of iterations to reach a final solution with a 10^{-5} relative precision. In contrast, on the same problems, it only takes about 1/10 of those iterations to reach a moderate 10^{-3} precision.

In this chapter, we try to improve the accuracy of ACCPM by combining IPM and simplex in the decomposition context. That is, we propose a hybrid of ACCPM and DW, which act as the decomposition counterparts of IPM and simplex, respectively. We use a weighted primal Newton method to calculate the analytic center in a carefully constructed

ACCPM master problem [24], and then adopt a weighted version of DW after effecting the switch. With this combination of methods, the coefficient matrix for the DW restricted master problem is readily available during the whole process. Thus, relative to other ACCPM variants, our proposed hybrid approach is simple in idea, and can be implemented without much additional effort. Furthermore, we present, for the first time, a warm start procedure for the weighted DW algorithm.

Our contributions are twofold. First, while IPM has been involved in variants of the DW algorithm (see [81], [63], and [87]), a hybrid method combining ACCPM and DW is a new attempt, and has not been proposed in the literature. Several techniques, such as the weighted versions of ACCPM [24] and DW, the constructed master problem [24], and a variant of warm-start recovery, are incorporated, in order to make the hybrid approach competitive. Second, in order to provide full-scale numerical results, our computational tests entail different sized problems with both primal and dual block angular structures. Our results throw light into how the hybrid method compares with ACCPM with respect to solution accuracy, and with DW in terms of handling degeneracy. We also discuss some factors that appear to influence the hybrid algorithm's performance.

6.2 Preliminaries

In order to illustrate the hybrid algorithm, we use a slightly different system of notations in this chapter.

6.2.1 Dantzig-Wolfe Decomposition Method

Consider a primal block-angular linear program

$$\begin{array}{lllll} \min & c_1^T z_1 + & c_2^T z_2 + & \cdots + & c_h^T z_h & \\ s.t. & A_1 z_1 + & A_2 z_2 + & \cdots + & A_h z_h & = b \\ & B_1 z_1 & & & & = d_1 \\ & & B_2 z_2 & & & = d_2 \\ & & & \ddots & & \vdots \\ & & & & B_h z_h & = d_h \\ & z_l \geq 0, & & & l & = 1, 2, \cdots, h. \end{array} \tag{6.1}$$

We denote the dimension of vector b by m_0, which is also the number of linking constraints. We use h for the number of blocks in the system. The Dantzig-Wolfe decomposition method removes those complicated linking constraints so that the rest can be solved as separate, smaller subproblems. Geometrically, any point in the feasible region of a linear program can be represented as a convex combination of the extreme points and extreme rays. Applying this fact to the subproblems, the original problem can be converted into an equivalent *full* master problem, which has fewer rows than (6.1) but usually many more columns. In practice, DW uses *column generation* [16], and works on a *restricted* master problem

$$\begin{array}{lllll} \min\limits_{\alpha_l^i} & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (c_l^T z_l^i) & & & \\ s.t. & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (A_l z_l^i) & = & b & \\ & \sum\limits_{i \in I_l} \alpha_l^i & = & 1, & l = 1, 2, \cdots, h \\ & \alpha_l^i \geq 0, & & & i \in I_l \cup J_l,\ l = 1, 2, \cdots, h \end{array} \tag{6.2}$$

where I_l represents the set of current extreme points from the l^{th} subproblem, and J_l is the set of extreme rays. The l^{th} subproblem (oracle), $l = 1, 2, \cdots, h$, can be described as

$$\begin{array}{ll} \min & (c_l^T - \pi^T A_l) z_l - \mu_l \\ s.t. & B_l z_l = d_l \\ & z_l \geq 0, \end{array} \tag{6.3}$$

where π and μ ($\mu = [\mu_1, \mu_2, \cdots, \mu_h]^T$) are the dual vectors of (6.2) corresponding to the linking and convexity constraints, respectively . Dual variables π and μ obtained as byproducts when solving the restricted master problem (6.2) are passed to the subproblems (6.3), which in turn return proposals to the restricted master problem. Then, a new iteration begins again.

6.2.2 Analytic Center Cutting Plane Method

In order to implement ACCPM, we need to add some artificial variables, z^- and z^+, to the canonical restricted master problem (6.2), and this yields:

$$\begin{array}{lll} \min\limits_{\alpha_l^i} & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (c_l^T z_l^i) + M_1^T z^- + M_2^T z^+ & \\ s.t. & \sum\limits_{l=1}^{h} \sum\limits_{i \in I_l \cup J_l} \alpha_l^i (A_l z_l^i) - z^- + z^+ = b & \\ & \sum\limits_{i \in I_l} \alpha_l^i = 1, & l = 1, 2, \cdots, h \\ & \alpha_l^i \geq 0, z^+ \geq 0, z^- \geq 0, & i \in I_l \cup J_l,\ l = 1, 2, \cdots, h \end{array} \tag{6.4}$$

where M_1 and M_2 are both $m_0 \times 1$ vectors.

Adding proposals in the primal space is equivalent to adding cuts in the dual space. The dual problem of (6.4) is

$$\max \quad b^T \pi + \sum_{l=1}^{h} \mu_l \tag{6.5}$$

$$s.t. \quad (A_l z_l^i)^T \pi + \mu_l \leq c_l^T z_l^i, \qquad i \in I_l, l = 1, 2, \cdots, h \tag{6.6}$$

$$(A_l z_l^j)^T \pi \leq c_l^T z_l^j, \qquad j \in J_l, l = 1, 2, \cdots, h \tag{6.7}$$

$$-M_1 \leq \pi \leq M_2 \tag{6.8}$$

The penalty coefficients M_1 and M_2 in the primal master problem (6.4) become the bounds for box constraints (6.8) in the dual master problem. These box constraints, together with a lower bound (LB), make the ACCPM localization set (a polyhedron defined to contain the optimal solution of system (6.5)$\sim$(6.8); see [34] for details) bounded. An extreme point in the primal space corresponds to an optimality cut given by (6.6) in the dual space, and an

extreme ray corresponds to a feasibility cut given by (6.7). To denote the above problems in a simple form, we define

$$A_m = \begin{bmatrix} \cdots A_l z_l^i \cdots & \cdots A_l z_l^j \cdots & -E & E \\ E_{conv} & 0 & 0 & 0 \end{bmatrix}, \tag{6.9}$$

where E is an $m_0 \times m_0$ identity matrix corresponding to the artificial variables, and E_{conv} corresponds to the coefficients of the convex combination constraints. Let $|I| = \sum_{l=1}^{h} |I_l|$ and $|J| = \sum_{l=1}^{h} |J_l|$ denote the total number of extreme points and extreme rays so far generated from the subproblems. A_m then has m_0+h rows and $|I|+|J|+2m_0$ columns altogether. We further define $x_m = \left[\cdots \alpha_l^i \cdots \alpha_l^j \cdots, z^{-T}, z^{+T}\right]^T$, $c_m = \left[\cdots c_l^T z_l^i \cdots c_l^T z_l^j \cdots,\ M_1^T, M_2^T\right]^T$, $y_m = \begin{bmatrix} \pi \\ \mu \end{bmatrix}$, and $b_m = \begin{bmatrix} b \\ e \end{bmatrix}$, where e is an $h \times 1$ vector with all ones, $i \in I_l$, $j \in J_l$, and $l = 1, 2, \cdots, h$. Now the primal restricted master problem with artificial variables (6.4) can be restated as

$$\begin{aligned} \min \quad & c_m^T x_m \\ s.t. \quad & A_m x_m = b_m \\ & x_m \geq 0 \end{aligned} \tag{6.10}$$

Similarly, the dual master problem becomes

$$\begin{aligned} \max \quad & b_m^T y_m \\ s.t. \quad & A_m^T y_m \leq c_m \end{aligned} \tag{6.11}$$

Our discussion above points to how DW and ACCPM are similar to each other in respect of receiving proposals from the subproblems and passing price information from the master. However, they differ in the *type* of price that is passed. DW sends *marginal* prices corresponding to the optimal solution of the restricted master problem, while ACCPM passes *central* prices by solving for analytic centers from the following problem, which is derived from (6.11) (for details, see [24]).

$$\max \left\{ \varphi_D(s) :\ A_m^T y_m \leq c_m \right\}, \tag{6.12}$$

where φ_D, the potential function in the dual space, is given by

$$\varphi_D(s) = \ln s_0 + \sum_{i=1}^{|I|+|J|+2m_0} \ln s_i, \tag{6.13}$$

where $s_0 = b_m^T y_m - LB$, $s = c_m - A_m^T y_m$ (s has elements $\{s_i, i = 1, 2, \cdots, |I| + |J| + 2m_0\}$). Sometimes, a proposal is repeatedly generated from a subproblem. It can be proved [36] that adding such a repeated proposal is equivalent to assigning a weight υ_i to the corresponding ln term in the potential function (6.13). This gives rise to the following weighted dual potential function

$$\varphi_D(s) = \upsilon_0 \ln s_0 + \sum_{i=1}^{|I|+|J|+2m_0} \upsilon_i \ln s_i \tag{6.14}$$

We use the weighted version of ACCPM in our experiments, and define $\upsilon_0 = \sum_{i=1}^{|I|+|J|+2m_0} \upsilon_i$ to balance the first term with the proposals added in the following iterations [36]. The weighted Lagrangian function of (6.12) and (6.14) is

$$L(x_m, y_m, s) = \left\{ \upsilon_0 \ln s_0 + \sum_{i=1}^{|I|+|J|+2m_0} \upsilon_i \ln s_i + x_m^T (c_m - A_m^T y_m - s) \right\}. \tag{6.15}$$

The first order optimality conditions of (6.15) are

$$\begin{array}{lll} \frac{\partial L}{\partial s}: & \frac{\upsilon_i}{s_i} - x_m^i = 0, & i = 1, 2, \cdots, |I| + |J| + 2m_0, \\ \frac{\partial L}{\partial y_m}: & \frac{\upsilon_0}{s_0} b_m - A_m x_m = 0, & x_m > 0, \\ \frac{\partial L}{\partial x_m}: & A_m^T y_m + s = c_m, & s > 0. \end{array} \tag{6.16}$$

Similar to [24], we define

$$\begin{array}{l} x_0 = \upsilon_0 / s_0, \ \widetilde{x} = \left[\begin{array}{c} x_0 \\ x_m \end{array} \right], \ \widetilde{c} = \left[\begin{array}{c} -LB \\ c_m \end{array} \right], \ \widetilde{s} = \left[\begin{array}{c} s_0 \\ s \end{array} \right], \\ \widetilde{\upsilon} = [\cdots, \upsilon_i, \cdots]^T, \ i = 0, 1, 2, \cdots, |I| + |J| + 2m_0, \\ \widetilde{y} = y_m, \ \widetilde{A} = [-b_m, \ A_m], \end{array} \tag{6.17}$$

and let $\widetilde{S}$ be the diagonal matrix of $\widetilde{s}$. Then, the optimality conditions can be re-written as

$$\begin{array}{lll} \widetilde{S}\widetilde{x} = \widetilde{\upsilon}, & & \\ \widetilde{A}\widetilde{x} = 0, & & \widetilde{x} > 0 \\ \widetilde{A}^T \widetilde{y} + \widetilde{s} = \widetilde{c}, & & \widetilde{s} > 0 \end{array} \tag{6.18}$$

The system of equations (6.16) is now converted to a standard form in (6.18). This problem has a similar form with the 'alternative master problem' in [63], the solution of which is a point located between the analytic center and the optimum. Next, we will use the primal Newton method to solve (6.18).

6.2.3 Weighted Primal Newton Method

An analytic center can be calculated by primal, dual, or primal-dual Newton methods. Among these, the primal Newton method is necessary for the version of ACCPM we used because dual feasibility is not guaranteed to be recovered after cutting a huge portion in the dual space. That is, ACCPM needs a feasible point to start with at every iteration, and only the primal feasibility, $\widetilde{A}\widetilde{x} = 0$, can be perfectly recovered after adding cuts [24]. Next, we continue in the same vein as Section 6.2.2 and introduce the *weighted* (to deal with proposal-repetition) version of the primal Newton method.

Upon definition, an analytic center can be obtained [24] by maximizing the weighted primal potential function

$$\max \left\{ \varphi_P(\widetilde{x}) : \widetilde{A}\widetilde{x} = 0,\ \widetilde{x} > 0 \right\}, \tag{6.19}$$

where $\varphi_P(\widetilde{x}) = -\widetilde{c}^T\widetilde{x} + \sum_{i=0}^{|I|+|J|+2m_0} \widetilde{v}_i \ln \widetilde{x}_i$. If we write out the Lagrangian function of (6.19), and take the first order derivatives, we obtain exactly the same optimality conditions as (6.18).

Denote $\widetilde{N}$ as the diagonal matrix of $\widetilde{v}$ and $\widetilde{X}$ as the diagonal matrix of $\widetilde{x}$. The weighted damped primal Newton steps [24] can be described as follows:

Initialize:

$\widetilde{A}\widetilde{x} = 0$, $\widetilde{x} > 0$

A centering parameter $0 < \theta < 1$

An initial value of $\|q(\widetilde{x})\| > \theta$

While $\|q(\widetilde{x})\| > \theta$,

1. $\widetilde{y}(\widetilde{x}) = (\widetilde{A}\widetilde{N}^{-1}\widetilde{X}^2\widetilde{A}^T)^{-1}\widetilde{A}\widetilde{N}^{-1}\widetilde{X}^2\widetilde{c}$

2. $\widetilde{s}(\widetilde{x}) = \widetilde{c} - \widetilde{A}^T\widetilde{y}(\widetilde{x})$
3. $q(\widetilde{x}) = \widetilde{v}^{\frac{1}{2}} - \widetilde{N}^{-\frac{1}{2}}\widetilde{X}\widetilde{s}(\widetilde{x})$
4. $d\widetilde{x} = \widetilde{N}^{-\frac{1}{2}}\widetilde{X}q(\widetilde{x})$
5. If $\|q(\widetilde{x})\| > 1$, then
   stepsize $\beta \in argmax\{\varphi_P(\widetilde{x} + \beta d\widetilde{x}) : \widetilde{x} + \beta d\widetilde{x} > 0\}$
6. Else
   $\beta = 1$
7. $\widetilde{x} = \widetilde{x} + \beta d\widetilde{x}$

End While

6.3 The Hybrid Approach

To motivate the hybrid approach, we observe that the matrix needed in the DW restricted master problem (6.10) is always readily available when updating the ACCPM master problem with more cuts (proposals) - A_m is a submatrix of $\widetilde{A}$. This is the key for easy switching.

6.3.1 Solving the Constructed Master Problem

The construction of the tilde denoted system (6.17) allows us to provide a compact representation of Newton's method. Nevertheless, at the time of switch, we need to recover a good feasible solution to problem (6.10) from $\widetilde{x}$. This feasible solution is the starting point in a basis recovery scheme described in section 6.3.4. Our first proposition shows how a feasible solution to the DW restricted master problem can be recovered from a solution to (6.18). The proposition also gives current upper and lower bounds (CUB and CLB), which will be used to update the current best upper and lower bounds (UB and LB) on the optimal value of the original problem.

Proposition 1. *At iteration k, if $\widetilde{x}^k = \begin{bmatrix} \widetilde{x}_0^k \\ \widetilde{x}_m^k \end{bmatrix}$ denotes a feasible solution to problem (6.18), and we assume that all the artificial variables are zero, then a feasible solution to the DW restricted master problem (6.10) is $\widetilde{x}_m^k/\widetilde{x}_0^k$, CUB to problem (6.1) is $(c_m^k)^T\widetilde{x}_m^k/\widetilde{x}_0^k$, and CLB is UB+$\sum_{l=1}^{h}\left[(c_l^T - (\pi^k)^T A_l)z_l^k - \mu_l^k\right]$, where π^k and μ_l^k are dual variables from the most recent master problem and z_l^k is the current proposal returned from the l^{th} subproblem.*

Proof. Since $\widetilde{x}^k$ is a feasible solution to (6.18) at the current iteration, it satisfies $\widetilde{A}^k\widetilde{x}^k = 0$. According to the definitions of the tilde variables (6.17), we have
$\left[-b_m, A_m^k\right]\begin{bmatrix} \widetilde{x}_0^k \\ \widetilde{x}_m^k \end{bmatrix} = 0$, i.e., $A_m^k\widetilde{x}_m^k = b_m\widetilde{x}_0^k$, i.e., $A_m^k\frac{\widetilde{x}_m^k}{\widetilde{x}_0^k} = b_m$,
implying that $\widetilde{x}_m^k/\widetilde{x}_0^k$ is feasible for the restricted master problem (6.10), which is a restatement of (6.4). In turn, since all the artificial variables are zero[1], $(c_m^k)^T\widetilde{x}_m^k/\widetilde{x}_0^k$, the objective value of the restricted master problem (6.10), can act as an upper bound of the original minimization problem (6.1). A lower bound, which comes from the subproblems, is the same as in the classical DW method: *CLB* is the current best upper bound (*UB*) plus the sum of subproblems' objectives. □

It is worth noting that in the DW method, the series of CUB is monotonically non-increasing, because we keep adding new proposals to the restricted master problem from one iteration to the next, and solve it to *optimality* for an upper bound. In ACCPM, a CUB comes from a feasible *center* point, so it is not optimal, and not monotonic anymore.

6.3.2 The Weighted Dantzig-Wolfe Decomposition Method

As mentioned earlier, we use the weighted primal Newton method to solve the ACCPM master problem. This means that a repeated proposal will not be added to the master problem matrix $\widetilde{A}$. Instead, its weight in $\widetilde{N}$ will be increased by one. Therefore, the weights

[1]Compared to the classical DW master problem (6.2), problem (6.10) contains some artificial variables. However, given the original problem is feasible, after a few iterations, these artificial variables in a solution to (6.10) will be zero valued (more precisely, very close to zero, in an IPM context), and hence the solution can be used to update CUB to the original problem.

(υ_i) only appear in the potential function given in (6.19), while $\widetilde{A}$, as well as A_m, contain unique proposals.

The weighted technique can help reduce the size of the master problem, and hence improve the computational efficiency. The effort for checking repetition can be ignored compared to other dominating calculations. Since we already have a mechanism to recognize repetition of proposals in ACCPM, and it takes little CPU time, after the switch, it seems natural to adopt the weighted technique in DW as well. We have devised a weighted DW that uses only unique proposals. Denoting N as the diagonal matrix of $\upsilon = \{\upsilon_i, i = 1, 2, \cdots, |I| + |J| + 2m_0\}$, a weighted DW restricted master problem can be defined as

$$\begin{aligned} \min \quad & (N * c_m)^T x_m \\ & (A_m * N) x_m = b_m \\ & x_m \geq 0 \end{aligned} \tag{6.20}$$

Usually, standard DW algorithms do not deal with the repeated proposal issue. They rather add every proposal satisfying a certain rule[2], and then *purge* (drop) a portion of the proposals after a few iterations [46].

Denote a_m^i as the i^{th} column of matrix A_m, c_m^i as the i^{th} element of vector c_m, and α^i as the i^{th} element of variable x_m. At the k^{th} iteration, assume that a same proposal is returned from the same subproblem. When this repeated proposal is added into the restricted master problem, we denote the new column in A_m as $a_m^{i'}$, the new element in c_m as $c_m^{i'}$, and the new element in variable x_m as $\alpha^{i'}$. A standard DW master problem, with one repeated proposal, can be described as

$$\begin{aligned} \min \quad & \left[\cdots c_m^i \cdots c_m^{i'} \cdots\right] \left[\cdots \alpha^i \cdots \alpha^{i'} \cdots\right]^T \\ s.t. \quad & \left[\cdots a_m^i \cdots a_m^{i'} \cdots\right] \left[\cdots \alpha^i \cdots \alpha^{i'} \cdots\right]^T = b_m \\ & \left[\cdots \alpha^i \cdots \alpha^{i'} \cdots\right] \geq 0 \end{aligned} \tag{6.21}$$

While we also employed the standard DW method for our DW code, for our hybrid implementation, we only store unique proposals. The next proposition shows that the weighted DW method is equivalent to the standard one.

[2]For example, the most common rule for a minimization problem: proposals from subproblems with negative reduced cost.

Proposition 2. *After the switch, the weighted DW restricted master problem (6.20) is equivalent to that of the standard DW method (6.21), and CUB in the successive iterations can be updated by* $(N * c_m)^T x_m$.

Proof. To show the equivalence for $v_i = 2$, since $a^i_m = a^{i'}_m$ and $c^i_m = c^{i'}_m$, the standard DW restricted master problem (6.21) becomes

$$\begin{array}{lll} \min & \left[\cdots + c^i_m(\alpha^i + \alpha^{i'}) + \cdots\right] & \\ s.t. & \left[\cdots + a^i_m(\alpha^i + \alpha^{i'}) + \cdots\right] & = b_m \\ & \left[\cdots \alpha^i \cdots \alpha^{i'} \cdots\right] & \geq 0 \end{array} \tag{6.22}$$

Defining $\alpha^{i''} = \frac{\alpha^i + \alpha^{i'}}{2}$, we have

$$\begin{array}{lll} \min & \left[\cdots + 2c^i_m\alpha^{i''} + \cdots\right] & \\ s.t. & \left[\cdots + 2a^i_m\alpha^{i''} + \cdots\right] & = b_m \\ & \left[\cdots \alpha^{i''} \cdots\right] & \geq 0 \end{array} \tag{6.23}$$

The left multiplication of c_m by N and the right multiplication of A_m by N as shown in (6.20) give the proper weights to the master problem, as deduced in (6.22) and (6.23). Any feasible solution to (6.21) corresponds to a feasible solution to (6.20) of equal objective value. A similar argument works in the other direction. Therefore, the standard and weighted DW methods are equivalent to each other. Then, a feasible solution of (6.20) can be used to update CUB as $(N * c_m)^T x_m$. The proof easily extends to the case where $v_i > 2$. □

The success of simplex in practice is mainly attributed to the warm start strategy [11]. Evidently, warm start also plays an important role in DW decomposition. After adding new proposals, the restricted master problem starts with the same basis as that of the previous iteration, treating all the new columns as nonbasic variables equal to zero, and then searches for entering pivots. For a weighted version, the difficulty lies in restoring the basic variables because if there is any column in the basis being repeated at the current iteration, its weight will be changed, and therefore the basic variables from the previous iteration are not feasible any more. The next proposition shows a warm start technique for the weighted DW method.

Proposition 3. *If the superscript k denotes the iteration index and subscript B stands for basic, then in the weighted DW method, a starting basis at iteration $k+1$ can be obtained by $(W * x^k_{mB})$, where W is a diagonal matrix whose diagonal elements are given as follows*

$$w_i = \left\{ \begin{array}{l} \frac{v_i}{v_i+1}, \text{ if the } i^{th} \text{ basic variable is repeated} \\ 1, \text{ otherwise} \end{array} \right\}, i = 1, 2, \cdots, m_0 + h, \tag{6.24}$$

and its objective function value equals the objective value at iteration k.

Proof. At iteration k, if none of the new proposals returned from subproblems are repeated, or if the proposal repetitions are only in the nonbasic variables, we can proceed to iteration $k+1$ and do a warm start with x^k_{mB} as in the classical DW method (all the elements in W are ones). Otherwise, assume that the i^{th} basic variable has just been repeated, and the corresponding diagonal elements of N^k_B and W are v_i and w_i, respectively. Substituting the adjusted basic variables $x^{k+1}_{mB} = W * x^k_{mB}$ into the weighted DW restricted master problem, we have

$$\begin{aligned} (A^k_{mB} * N^{k+1}_B)x^{k+1}_{mB} &= A^k_{mB} * \begin{bmatrix} v_1 & & & & \\ & \ddots & & & \\ & & v_i + 1 & & \\ & & & \ddots & \\ & & & & v_{m_0+h} \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & \frac{v_i}{v_i+1} & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix} x^k_{mB} \\ &= A^k_{mB} * N^k_B x^k_{mB} = b_m \end{aligned} \tag{6.25}$$

Similarly, the objective

$$\begin{aligned} (N^{k+1}_B * c^k_{mB})^T x^{k+1}_{mB} &= (c^k_{mB})^T (N^{k+1}_B)^T (W x^k_{mB}) = (c^k_{mB})^T (N^{k+1}_B W) x^k_{mB} \\ &= (N^k_B * c^k_{mB})^T x^k_{mB} \end{aligned} \tag{6.26}$$

Thus, $W * x^k_{mB}$ is a feasible solution to problem (6.20) at iteration $k+1$ and the objective value remains the same, thereby showing the proposition. □

6.3.3 Switch Criteria

Different switch criteria can be employed in our hybrid algorithm. The most straightforward one is to use the relative duality gap, which generally measures the progress of an

algorithm. At the beginning, the gap could be huge due to the initial upper and lower bounds. The gap then reduces quickly as deep cuts are added in ACCPM [37]. However, a slow-down or even a long tail may occur to ACCPM at the end of the calculation. We can set a user supplied parameter ε_s (e.g., $\varepsilon_s = 0.1$), and switch to DW when the relative duality gap falls below this small value.

The second possible switch criterion comes from measuring the contribution of the subproblems. Our experiments reveal that in the last few iterations, there tends to be very few *new* cuts generated from the subproblems. This is especially the case for sparse problems. For the stochastic financial test problems, the average repetition rate of proposals is 75.5%. Merle, Goffin, and Vial [71] also reported the repeated proposal phenomenon, and proposed an improved algorithm 'ACCPM+' calling both ACCPM and Kelley's method to solve the master problem at each iteration. Since we use the weighted ACCPM, at every iteration, we already check whether a proposal is repeated. Therefore, the second switch criterion is based on the number of new cuts: if there is nothing new at all in the current iteration, which means that all subproblems return repeated proposals, switch to DW. Notice that the lack of new proposals at the current iteration does not mean that new ones will not be generated thereafter, but in fact, new cuts appear very sporadically even if there are, and it does not hurt to switch right away.

Although we have not implemented them, other switch criteria are possible. For example, one can set an arbitrary number k_s, then after iteration k_s, the hybrid algorithm switches from DW to ACCPM. The total number of proposals (cuts) already added can also act as a switch indicator.

6.3.4 Recovering a Basis

The crossover iteration has long been considered the main difficulty for hybrid algorithms. Megiddo [69] proposed a strongly polynomial time procedure to find an optimal (both primal and dual) basis from an interior point solution. This method was used and slightly modified in [8] and [9]. Andersen and Ye [3] designed an IPM-based search direction to recover a basis in any iteration of IPM.

At the crossover, we first need to create a weighted DW master problem from (6.18). Furthermore, artificial variables are eliminated (purification) at this time (see (6.27) on

page 126). IPM is then used to solve the purified restricted master problem, which is counted as one iteration of the whole procedure, and the current best upper bound is also updated. As mentioned earlier, by passing *central* prices [35], ACCPM can reduce the number of iterations. On the other hand, an analytic center, which is a geometric center of the localization set to maximize the product of the distance to every edge, is only a feasible but not optimal point to the restricted master problem, and we use it to update the upper bound in ACCPM. At the crossover, the restricted master problem is solved to optimality by IPM, and therefore the upper bound can usually be improved a lot.

Next, we identify a basis with which the DW master problem can start right after the switch. In MATLAB, the IPM solver, *lipsol*, returns a primal-dual pair of solution, so that it is easy to check the complementary slackness conditions. We loosely follow Megiddo's idea, with the focus on the primal basis recovery. The procedure is described in the following pseudo code (for details, see [69]):

1. Set zero tolerance parameters
   $\varepsilon_b = 10^{-4}$ for choosing basic variables
   $\varepsilon_0 = 10^{-6}$ for the zero tolerance of dual slack variables

2. Select the positive variables, $x_{mj} > \varepsilon_b$, and take the corresponding columns in A_m as a *candidate basis*

3. If the columns in the candidate basis are linearly dependent[3], then
   increase the value of ε_b and go back to step (2)
   End If

4. While the candidate basis has more rows than columns
   If there are columns in A_m with $s_j \leq \varepsilon_0$ available[4], then

[3]In MATLAB, this step was actually implemented as: if the candidate basis has more columns than rows | | the columns in the candidate basis are linearly dependent, where the logical operator '| |' stands for a short-circuit *OR* operation. That is, if the first operand ('more columns than rows'), which is easy to check, is true, the second operand ('columns linearly dependent'), which is computationally expensive, will not be evaluated.

[4]If $s_j = 0$, then the corresponding dual constraint is binding, which is a property of basic variables, by complementary slackness. That is why we consider adding these columns first.

If such a column is linearly independent of the candidate basis, then
add it to the candidate basis
End If
Else
add a linearly independent column with $s_j > \varepsilon_0$ to the candidate basis
End If

End While

We now have distinguished the potential basic/nonbasic variables, and obtained a non-singular basis. This basis is not optimal for the current restricted master problem, but close to it. Restoring the current basis to an optimal basis can be expensive, especially for highly degenerate problems [9]. We find in the experiments that using the *near-optimal* basis itself as a warm start for DW yields good results. The near-optimal basis is only used at the iteration right after the crossover. It lies in the middle of the whole hybrid algorithm, and there would be a number of DW master problem tableau updates afterwards. Therefore, using a near-optimal basis has little influence on the global convergence or the final precision in this case. In addition, the computation time of identifying a near-optimal basis is so little that it can be ignored.

6.3.5 Description of the Hybrid Solution Algorithm

We start with an initial primal feasible point $\widetilde{x}^0$ ($\widetilde{A}\widetilde{x}^0 = 0$), initial lower and upper bounds, and with the switch-flag set to FALSE. We first go with ACCPM, and then switch to DW when the switch flag meets the chosen criterion. The hybrid algorithm can be described as:

While relative duality gap $> \varepsilon$

1. If switch-flag is FALSE, then
   use primal Newton method (Section 2.3) to solve for an analytic center
   -pass central price to subproblems
   Else
   solve the weighted DW restricted master problem (Sections 3.1 and 3.2)

-pass dual price to subproblems
End If

2. Update upper bound (UB)

3. Solve subproblems
If a proposal is repeated, then
increase the weight of that column by one
Else
add a new column (a cut in the dual space)
End If

4. Update lower bound (LB)

5. Check if switch-flag should be changed to TRUE (Section 3.3)
If yes, then perform basis recovery (Section 3.4), End If

End While

6.4 Numerical Results

The codes for ACCPM, the hybrid approach, and DW were all written in MATLAB. We modified the simplex code in MATLAB so that a warm start can be initiated for the master problem and subproblems in DW, as well as for the master problem and subproblems in the hybrid approach (i.e., after the switch). To keep a consistent IPM approach in ACCPM, the subproblems are solved by *lipsol*, which is an IPM based solver in MATLAB[5] and does not support warm start.

We ran the test problems on a Sun Blade 2500 workstation with UNIX operating system. The CPU time of solving a problem is the sum of the time for the master problem, the

[5]Using IPM to solve subproblems means that the extreme points and rays are all approximate, but not exact. However, experiments show that this has little influence on the overall accuracy of ACCPM. In fact, the major numerical difficulty that ACCPM encounters lies in the large-sized master problem.

total time of subproblems, and the time for recovery (ACCPM only)[6]. Time for loading data and preprocessing is not included. When the relative duality gap falls below a small tolerance ε, which is set to 10^{-6}, the algorithms stop. However, for ACCPM, which is unable to achieve this tolerance on any of our problems, if it cannot make any further improvement (e.g., less than 10^{-8}) over the duality gap, the algorithm stops.

6.4.1 Stochastic Financial Model

The test results are organized under three headings: convergence (accuracy and number of iterations), timing (CPU time), and degeneracy issues.

Convergence

For the stochastic financial model, Table 6.1 reveals the convergence properties of ACCPM, the hybrid approach, and DW. For ACCPM, $iter_{1\%}$ is the number of iterations it takes to reach 1% relative duality gap; *iter* is the total number of iterations when ACCPM eventually reaches the relative duality gap (precision) shown in the next column gap_{accpm}. For the hybrid approach, $iter_a$ is the number of iterations with ACCPM, $iter_d$ is the iterations with DW, and gap_{hyb} represents the final precision reached. Similarly, under the columns labeled DW, $iter_{dw}$ is the number of iterations for Dantzig-Wolfe decomposition to reach a precision of gap_{dw}.

According to Table 6.1, the IPM based algorithm, ACCPM, converges rapidly at the beginning: it takes only 5.4 iterations on average to reach a 1% duality gap. Unfortunately, it slows down when pursuing a better accuracy. None of them could reach a relative duality gap of 10^{-6} or lower.

Figure 6.1 shows the typical convergence behavior of ACCPM (we randomly chose test problem 6 for illustration). In Figure 6.1 (a), ACCPM starts with a huge gap between the initial upper bound and lower bound. The two lines come quite close after just a couple of iterations. Figure 6.1 (b) is an amplified figure from the 4^{th} iteration to the end, from which we can see the slow-down tendency.

[6] After adding new cuts, we need to recover a new feasible point as a starting point for the next iteration. This also acts as a warm start in ACCPM.

Figure 6.1: Tailing Effect of ACCPM on Financial Prob.6 - LB and UB

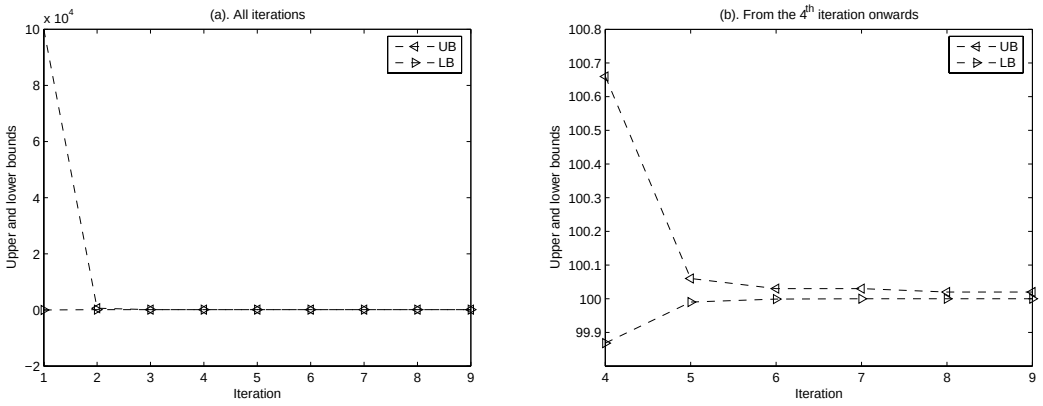


Figure 6.2: Tailing Effect of ACCPM on Financial Prob.6 - Relative Duality Gap

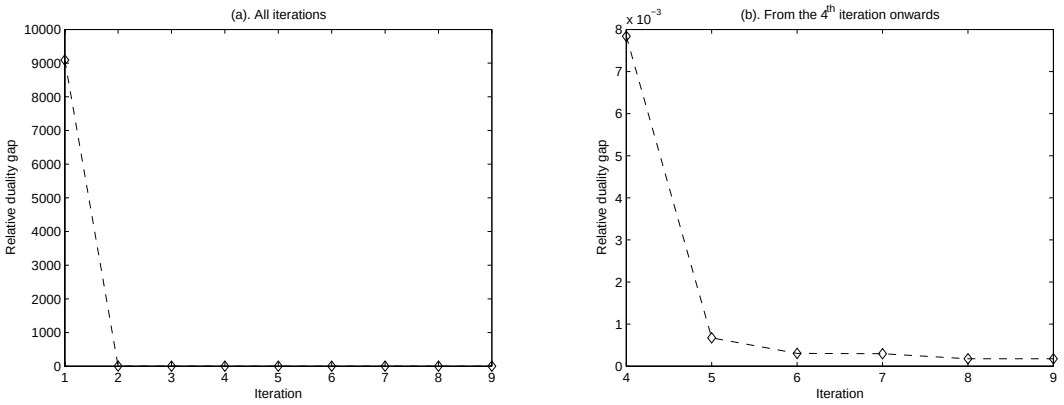


Table 6.1: Stochastic Financial Problem Convergence Properties

Prob.	n	h	ACCPM			Hybrid			DW	
			$iter_{1\%}$	$iter$	gap_{accpm}	$iter_a$	$iter_d$	gap_{hyb}	$iter_{dw}$	gap_{dw}
1	10	10	6	12	4.02E-05	5	1	4.32E-09	5	2.49E-15
2	10	40	4	7	1.72E-04	3	1	1.27E-11	3	-1.11E-14
3	10	70	5	6	2.55E-04	3	1	2.36E-10	3	1.02E-13
4	10	100	5	10	3.11E-04	3	1	1.58E-10	3	1.32E-14
5	40	10	8	13	3.39E-04	6	1	4.49E-09	5	8.68E-16
6	40	40	5	9	2.25E-04	3	1	3.03E-12	3	-3.60E-14
7	40	70	4	10	2.83E-04	3	1	3.11E-14	3	4.66E-14
8	40	100	5	13	5.06E-04	3	1	9.14E-11	3	2.06E-14
9	70	10	7	16	7.34E-05	8	1	9.68E-11	10	2.02E-16
10	70	40	6	10	5.85E-04	4	1	5.35E-12	5	-7.68E-14
11	70	70	5	10	5.19E-04	3	1	2.30E-13	3	1.07E-13
12	70	100	4	7	6.96E-04	3	1	1.20E-11	3	1.32E-14
13	100	10	7	13	1.58E-04	8	1	1.27E-11	9	4.32E-16
14	100	40	5	10	2.68E-04	3	1	2.10E-09	3	1.83E-14
15	100	70	5	8	8.33E-04	3	1	1.45E-10	3	5.36E-14
16	100	100	5	9	5.64E-04	3	1	6.64E-09	3	1.11E-13

Figure 6.2 (a) shows the relative duality gap (for problem 6) during the whole solving process. Figure 6.2 (b), which amplifies the interval of the $4^{th} \sim 9^{th}$ iteration, clearly indicates that toward the end, ACCPM could hardly make a 0.1% improvement from one iteration to the next. In fact, after the 9^{th} iteration, ACCPM can still continue but can make little progress as measured by the relative duality gap. If allowed to proceed, we observed that it continues for about 200 more iterations and finally crashes in MATLAB.

This tailing effect of ACCPM also occurred in the experiments of [4], wherein one of the test problems took 218.8 seconds to reach a relative duality gap of 0.026, while spending 2,239.8 seconds to reach 10^{-3}.

Because the proposals are frequently repeated, we use the second switch criterion in this model: when there is no *new* proposal returned from any subproblem at the current

iteration, switch to the Dantzig-Wolfe decomposition method. In this case, just one more iteration after the switch, all the test problems immediately reach a precision below 10^{-6} (more specifically, between 10^{-8} and 10^{-13}).

Originally, we devised the hybrid algorithm as an improvement of ACCPM. Since it involves DW, the numerical results of DW are also included in Tables 6.1 (and also in Table 6.2, below) for comparison purposes. DW has excellent convergence performance: for most of the problems, it takes only three iterations including phase I to reach an optimum.

Table 6.2: Stochastic Financial Problem Runtime and Proposals/cuts Added

Prob.	n	h	CPU time (seconds)				Repetition in ACCPM		
			ACCPM′ (10^{-2})	ACCPM (10^{-4})	Hybrid ($< 10^{-6}$)	DW ($< 10^{-6}$)	total	unique	r%
1	10	10	2.34	4.83	0.64	0.45	120	31	74.17%
2	10	40	9.01	16.06	8.60	9.97	280	80	71.43%
3	10	70	25.32	30.47	22.45	21.50	420	140	66.67%
4	10	100	48.00	99.33	54.79	50.04	1,000	200	80.00%
5	40	10	7.30	13.21	2.23	1.65	130	42	67.69%
6	40	40	36.46	66.79	15.08	16.13	360	80	77.78%
7	40	70	77.97	199.94	56.41	56.45	700	140	80.00%
8	40	100	187.30	504.48	152.63	124.25	1,300	200	84.62%
9	70	10	11.67	25.76	3.45	2.01	160	45	71.88%
10	70	40	113.17	200.11	29.02	33.17	400	88	78.00%
11	70	70	209.21	425.63	72.67	72.75	700	140	80.00%
12	70	100	320.54	570.63	196.57	153.66	700	200	71.43%
13	100	10	17.81	31.88	4.90	1.45	130	37	71.54%
14	100	40	141.87	293.55	28.81	31.93	400	80	80.00%
15	100	70	382.32	621.82	111.98	113.32	560	140	75.00%
16	100	100	675.29	1,237.90	340.32	383.91	900	200	77.78%
Average			141.60	271.40	68.78	67.04	516.25	115.19	75.50%

Timing

Timing data (in seconds) are given in Table 6.2. Notice that the proposal repetition rate ($r\%$) in the last column of Table 6.2 is as high as 80% for ACCPM. The number of total cuts added during an ACCPM solution procedure is large, but the unique ones are much less. This means that from a geometric perspective, the restricted master problem has obtained most of the extreme points (rays) it needs to reach the optimal solution a long time ago, yet the nature of ACCPM prevents it from approaching the boundaries of the feasible region, and it has to repeat some proposals again and again to raise the weights so that it can finally reach an acceptable near-optimal solution. This explains why the hybrid approach has a lower CPU time than ACCPM does, and takes only one more step after switching to DW to attain a good precision.

Notice that the hybrid approach seems always faster than ACCPM. Since this is attributable to the *long-tail* issue, for applications that do not require extreme accuracy, it may be inappropriate to count the total time after ACCPM has slowed down near the optimal solution. To include such cases, ACCPM$'$ in Table 6.2 provides the CPU time for ACCPM to reach a 1% relative precision.

As discussed above, DW converges quickly on the test problems, but it does not entirely outperform the hybrid in terms of solution time. This is because although the number of iterations is small, the average time for each iteration may be longer. The DW CPU time is slightly better than the hybrid CPU time on average, but generally speaking, they are comparable to each other.

Degeneracy

The test problems of the financial model are quite sparse. Generally speaking, all the three decomposition algorithms converge quickly on them. Yet, DW, as well as simplex, are occasionally prone to suffer from degeneracy, especially for large sparse problems. Once simplex passes the degenerate point, it can perform well [8]. In our experiments, some problems (not reported here) solved by DW kept cycling until hitting the maximum number of iterations allowed ($100,000$). In the hybrid algorithm, the early ACCPM steps usually help pass the degenerate points for the later DW steps, and we did not observe any cycling here. Incidentally, we observed in our experiments that after re-scaling, DW could

possibly work fine on the previously degenerate problems. In contrast, an ill-conditioned problem does affect ACCPM, but mainly on the accuracy that it could ever attain, and cycling was not observed. In this sense, ACCPM and IPM are more reliable than DW and simplex.

6.4.2 Multicommodity Network Flow Model

Convergence

For the multicommodity network flow problems, Table 6.3 reveals the convergence properties of ACCPM, the hybrid approach, and DW. The proposal repetition rate for this model is only 22.20% on average, which is much lower than that of the financial model. Therefore, the first switch criterion is adopted here: when the relative duality gap is less than ε_s, $\varepsilon_s = 0.1$, switch to DW. Compared to the one-more-step results in the financial model, multicommodity problems need more iterations after switching to DW perhaps because they are complicated with more linking constraints. Again, we see that both the hybrid and DW accomplish comparable accuracy, measured by the relative duality gap, which is much smaller than that of ACCPM. On average, DW takes 17.52% more iterations than the hybrid method.

Timing

Table 6.4 shows the CPU time (in seconds) and the number of proposals (cuts)[7] added for the three algorithms. The two columns under ACCPM′ are the CPU time and the number of cuts added for ACCPM when reaching a 1% relative duality gap.

Unlike the financial model, some of the problems here contain a high percentage of linking constraints which makes them quite difficult to solve. Table 6.4 indicates that generally, DW tends to be faster than ACCPM for smaller problems, while for larger ones, ACCPM appears more advantageous than DW, although we must point out its significantly lower accuracy. In comparing the hybrid algorithm with the other two, we see that on average, the hybrid is 11.16% faster and at least two digits more accurate than ACCPM;

[7]The number of cuts refers to the number of *unique* ones for ACCPM and the hybrid algorithm; it is the *total* number of proposals for DW as the DW code does not check for repetition.

the hybrid is 55.68% faster than DW and reaches almost the same accuracy. We then conclude that the hybrid approach, which combines the two decomposition algorithms, is competitive with the better CPU time of ACCPM or DW, on any sized problems. Further, since the weighted version of both ACCPM and DW are used in the hybrid approach, its number of proposals added during the whole procedure is less than that of the traditional DW method (see Table 6.4). This is helpful to control the size of the master problem, as larger matrix will lead to expensive computational effort, significant round-off error, and digit loss.

6.4.3 Discussion

According to our experiments, there are some other issues that affect the three decomposition algorithms and their relative efficiencies.

The Role of Linking Constraints

From the above numerical results, we found that the percentage of linking constraints greatly influences the performance of the algorithms. According to Tables 2.1 and 2.3, all the financial problems contain very low $m_0\%$ ($< 8\%$), while the multicommodity problems have a larger range of $m_0\%$ (from 2.13% to 62.85%). Therefore, we use the multicommodity model to assess the impact of variations in $m_0\%$.

Figure 6.3 illustrates the relative efficiency of ACCPM vs. DW, measured as CPU_{accpm}/CPU_{dw} on the vertical axis, versus $m_0\%$ on the horizontal axis. The points reveal an increasing overall tendency, i.e., DW is more efficient than ACCPM for a larger percentage of m_0. In fact, among the three algorithms, ACCPM suffers the most from a higher $m_0\%$, DW behaves the best in such situations, and the hybrid approach is somewhere between them (see Figure 6.4).

Why does ACCPM take so long when $m_0\%$ is high? Let us recall the ACCPM master problem (6.4). Compared to the classical DW restricted master problem (6.2), (6.4) has some artificial variables added to the linking constraints so that the localization set in the dual space can be bounded. For example, for prob.4, which has the highest $m_0\%$ of nearly 70% (see Table 2.3), there will be $2 * m_0 = 866$ artificial variables added in

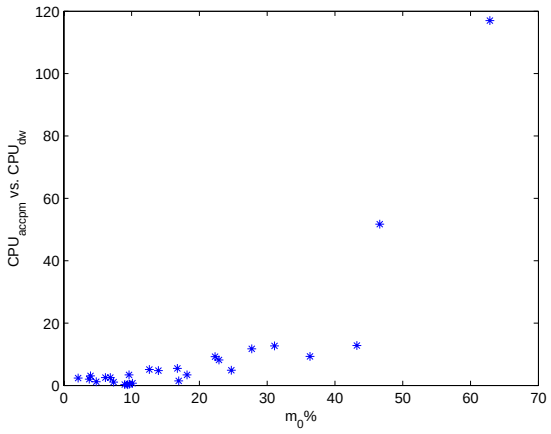
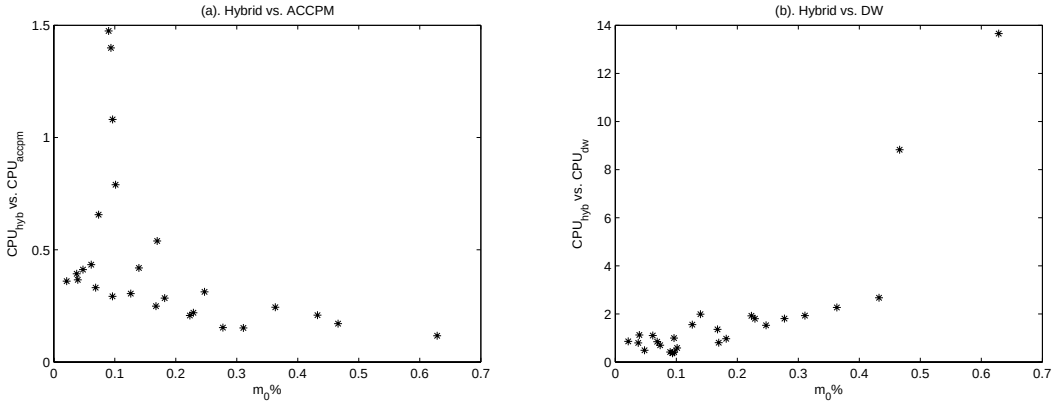


Figure 6.3: CPU Time of ACCPM vs. DW on Multicommodity Model

Figure 6.4: Effect of Varying the Percentage of Linking Constraints on Relative Efficiencies



the ACCPM master problem, while in the whole solution procedure, only 45 proposals are added from the subproblems. The huge number of artificial variables leads to a huge matrix $\widetilde{A}$ and then expensive computational effort for ACCPM. DW, instead, can remove those artificial variables in phase I after 4 iterations, and in the next 13 phase II iterations, it works with a much smaller coefficient matrix. This provides a plausible explanation as to why ACCPM's performance is relatively poor when dealing with problems with a high percentage of linking constraints.

In the implementation of the hybrid approach, we can usually leave out the artificial variables after the switch. Define

$$A_{dw} = \begin{bmatrix} \cdots A_l z_l^i \cdots & \cdots A_l z_l^j \cdots \\ E_{conv} & 0 \end{bmatrix}, \tag{6.27}$$

which is the coefficient matrix of problem (6.2). A_{dw} is a submatrix of A_m, and A_m is further a submatrix of $\widetilde{A}$, see (6.9) and (6.17). Without the Ms, both efficiency and accuracy can be improved. However, the feasibility of (6.2) is not guaranteed at the time of switch. Fortunately, the infeasible case rarely happens as we do the switch at quite a late stage. At this time there are usually enough proposals in the restricted master problem to make (6.2) feasible. Even if we do the switch earlier, it is easy to check if all the artificial variables are zero (or very close to zero in IPM).

Switch Time

A switch flag does not affect the hybrid's final precision, but only controls when the switch happens. On 20 multicommodity network flow problems, Table 6.5 compares how ε_s influences the convergence properties of the hybrid approach. It is easy to see from Table 6.5 that, when $\varepsilon_s = 0.1$, it takes more iterations with DW; and when $\varepsilon_s = 0.001$, it takes more iterations with ACCPM. The total numbers of iterations for the three ε_s values are almost the same.

Theoretically, we should go with ACCPM as far as possible for quick convergence purpose, and then go with a few DW iterations for accuracy. However, in practice, the *long-tail* problem of ACCPM negatively impacts its efficiency when close to optimal. Switching to DW before the slow-down might make it faster.

For the hybrid approach, switching to DW too early is usually not favored in terms of CPU time (except for problems with large $m_0\%$), convergence (avoiding degeneracy), and the feasibility of problem (6.2). On the other hand, switching too late will cause long-tail in ACCPM. The key is to choose a balance point depending on actual situations.

6.5 Conclusions

Generally speaking, simplex-based algorithms may encounter degeneracy, but are excellent at precision; IPM-based algorithms have polynomial worst-case complexity, but will never reach an exact solution. The hybrid decomposition approach that we present in the paper aims to take the best qualities from both ACCPM and DW: start by going through the interior of the polyhedron (dual space) to quickly reach a feasible point close to the optimum, recover this inner point to a nearby corner point (primal space), then move from vertex to vertex to achieve better accuracy.

In fact, accuracy was the initial motivation and main pursuit when we started working on the hybrid algorithm, because the IPM-based ACCPM sometimes has trouble ever attaining a satisfactory solution. By doing a few DW iterations near the end, the hybrid approach can keep the good convergence rate of ACCPM, as well as reach the same level of accuracy as DW. Furthermore, due to the dual equivalence, in any ACCPM iteration, all the information needed for the weighted DW restricted master problem is available from the primal Newton coefficient matrix. Therefore, very little effort is required at the switch to recover a near-optimal basis.

The superior convergence rate of early ACCPM iterations, noticed in the literature, may be counteracted by its tailing effect toward the end. Therefore, as shown in Table 6.5, the convergence properties of the hybrid approach are not sensitive to the precise choice of a switch flag: experiments with ε_s ranging from 0.001 to 0.1 yield similar average iteration counts.

Experiments with two types of block-angular test models show promising results for the hybrid approach. It is always more accurate, and usually faster than ACCPM. For the larger problems tested (i.e., those requiring at least hundreds of seconds of solution time), our hybrid algorithm appears to outperform DW as well, in terms of both convergence rate

and solution time. It is well known that DW (simplex) is faster than ACCPM (IPM) for smaller problems, and ACCPM (IPM) is faster for larger ones. Our tests indicate that the hybrid is competitive with the better of ACCPM and DW, regardless of problem size. In addition, degeneracy issues associated with classical DW are not observed in the proposed method. We, therefore, conclude that the hybrid approach is a potentially robust and reliable alternative to consider when solving block-angular linear programs.

According to the experiments, ACCPM seems undesirable when a problem has a high percentage of linking constraints, maybe because a huge number of artificial variables have to be added in the master problem. Despite this, it is unclear as to how the artificial variables could be removed in ACCPM as in DW. Even after plenty of iterations and with many cuts added in, the localization set in the dual space might still be unbounded without the box constraints. Moreover, checking the boundedness itself is very expensive if not impossible. Had an efficient way existed to eliminate the artificial variables for ACCPM, one could expect a significant improvement for large, complicated problems.

Decomposition algorithms are inherently suited to parallel computing. A cluster of inexpensive computers can solve subproblems in parallel and handle large-scale problems with millions of variables and constraints [29]. Since this research represents the first attempt at an ACCPM-DW hybrid, we did the experiments on a single workstation (i.e., a serial computing paradigm). Thus, for future research, it would be more representative of decomposition to conduct the experiments in a parallel computing environment. This will not affect the accuracy of the algorithms, but the time will be quite different due to the parallel solutions of subproblems and factors such as processor communications and load balancing.

Table 6.3: Multicommodity Network Flow Problem Convergence Properties

Prob	n	h	ACCPM			Hybrid			DW	
			$iter_{1\%}$	$iter$	gap_{accpm}	$iter_a$	$iter_d$	gap_{hyb}	$iter_{dw}$	gap_{dw}
1	64	4	8	11	7.41E-05	6	4	-3.13E-16	11	0*
2	64	4	15	22	7.78E-05	9	17	1.37E-16	30	1.37E-16
3	64	4	8	12	6.03E-05	5	6	-4.33E-16	10	0
4	64	4	8	17	8.45E-05	6	9	0	15	0
5	64	8	6	9	4.86E-05	5	4	1.46E-16	6	3.65E-16
6	64	8	10	17	7.68E-05	6	11	4.77E-16	17	-2.98E-16
7	64	8	11	21	8.57E-05	7	21	0	27	0
8	64	8	10	19	9.58E-05	7	14	-1.26E-16	22	-3.13E-16
9	64	16	6	8	8.50E-05	4	5	-1.32E-16	9	-4.95E-16
10	64	16	12	20	7.44E-05	8	14	8.16E-16	21	0
11	64	16	8	16	5.83E-05	6	13	3.61E-16	19	-4.51E-16
12	64	16	14	32	9.93E-05	8	21	-3.49E-06	47	-2.60E-16
13	64	32	8	11	5.80E-05	5	8	9.78E-07	12	9.30E-07
14	64	32	11	16	7.33E-05	7	11	9.99E-07	19	1.81E-07
15	64	32	7	13	8.47E-05	5	10	4.98E-07	17	1.81E-07
16	64	32	17	30	9.77E-05	9	29	9.80E-07	46	7.79E-07
17	64	64	7	11	5.90E-05	5	7	0	13	2.08E-07
18	64	64	9	16	4.18E-05	7	8	-2.17E-16	16	7.25E-07
19	64	64	8	15	7.43E-05	5	14	6.11E-07	23	4.91E-07
20	64	64	19	32	7.24E-05	11	29	6.26E-07	54	4.49E-07
21	64	64	21	34	7.01E-05	13	32	2.45E-07	60	9.58E-07
22	64	64	17	31	8.59E-05	9	34	7.32E-07	57	9.59E-07
23	128	8	15	30	6.27E-05	7	30	4.97E-07	44	1.12E-07
24	128	16	17	40	9.67E-05	6	57	8.06E-07	74	8.79E-07
25	128	32	14	27	9.79E-05	7	28	9.70E-07	46	9.36E-09
26	128	32	16	37	8.83E-05	8	51	7.36E-07	70	5.94E-07

*0 means a value $< eps$ (machine precision, about 10^{-16}).

Table 6.4: Multicommodity Network Flow Problem Runtime and Proposals/cuts Added

		CPU time (seconds)				# of cuts (proposals)			
n	h	ACCPM′ 10^{-2}	ACCPM 10^{-4}	Hybrid $< 10^{-6}$	DW $< 10^{-6}$	ACCPM unique	ACCPM′ unique	Hybrid unique	DW total
64	4	4.63	6.69	2.09	1.37	33	30	30	35
64	4	24.55	39.43	9.61	4.24	73	57	84	107
64	4	25.20	43.12	9.00	3.37	35	29	36	33
64	4	197.72	540.46	63.09	4.62	45	28	38	50
64	8	5.34	8.45	3.54	1.78	44	41	45	40
64	8	19.40	39.88	8.27	4.31	100	75	94	104
64	8	66.05	146.78	22.30	11.56	123	88	142	171
64	8	379.78	794.80	135.69	15.37	121	79	133	156
64	16	8.45	11.66	5.06	4.60	87	82	87	107
64	16	36.26	73.66	22.45	14.43	242	182	258	285
64	16	41.49	106.36	26.45	19.49	191	125	215	257
64	16	405.60	1,208.20	185.20	102.67	380	223	419	611
64	32	25.88	37.78	13.83	12.29	260	237	278	273
64	32	57.87	104.75	34.68	41.09	426	347	372	519
64	32	64.30	151.54	44.29	44.45	307	218	328	435
64	32	707.15	1,823.00	982.69	1,225.90	805	523	1,011	1,293
64	64	49.09	89.84	32.37	37.71	506	428	502	1,306
64	64	91.61	223.72	87.94	111.04	707	570	710	863
64	64	122.58	343.12	141.28	288.61	734	506	822	1,222
64	64	1,651.10	4,561.90	4,928.70	12,142.00	1,699	1,212	2,102	3,022
64	64	1,809.60	4,905.00	6,862.50	18,806.00	1,959	1,343	2,382	3,512
64	64	1,090.90	3,734.10	5,507.00	13,248.00	1,674	1,087	2,071	3,149
128	8	156.25	403.62	88.66	49.29	213	120	255	327
128	16	683.05	2,369.30	674.72	697.95	575	272	825	1,088
128	32	303.74	859.20	564.09	806.41	748	448	890	1,303
128	32	888.24	3,581.10	2,827.90	4,832.90	1,081	512	1,648	2,106
Average		342.92	1,007.98	895.52	2,020.44	506.46	340.85	606.81	860.54

Table 6.5: Influence of Switch Flag for the Hybrid Approach

Prob	$\varepsilon_s = 0.1$			$\varepsilon_s = 0.01$			$\varepsilon_s = 0.001$		
	$iter_a$	$iter_d$	$iter_{tot}$	$iter_a$	$iter_d$	$iter_{tot}$	$iter_a$	$iter_d$	$iter_{tot}$
1	5	5	10	7	3	10	9	1	10
2	9	1	10	10	1	11	12	1	13
3	5	5	10	7	3	10	11	1	12
4	6	10	16	8	9	17	13	3	16
5	5	4	9	7	4	11	8	2	10
6	6	12	18	9	5	14	12	2	14
7	7	19	26	11	17	28	17	9	26
8	7	17	24	11	10	21	17	6	23
9	4	5	9	6	3	9	7	1	8
10	9	13	22	13	10	23	17	5	22
11	7	15	22	10	8	18	15	5	20
12	9	30	39	15	31	46	25	13	38
13	5	8	13	8	5	13	10	4	14
14	7	11	18	11	7	18	13	4	17
15	5	10	15	7	12	19	10	7	17
16	8	29	37	16	17	33	25	10	35
17	5	7	12	7	5	12	9	3	12
18	7	9	16	9	5	14	12	4	16
19	6	15	21	11	10	21	14	6	20
20	11	29	40	18	19	37	26	11	37
Average			19.35			19.25			19

Chapter 7

Implementation Issues

7.1 Software

Five algorithms are involved in our experiments: DW, ACCPM, the hybrid method, simplex, and IPM. The first three decomposition codes are relatively complicated, while the other two codes, which are direct solution approaches, are much easier.

There are a few DW implementations available in the literature (e.g., [44][43][46]), but the size of our test problems exceeds their solving capacities. An ACCPM software package [40] is also available from Logilab[1] for research purposes, but is not flexible for decomposition applications. Therefore, we wrote all the codes by ourselves.

7.1.1 MATLAB

MATLAB is a high-level language for technical computing. Although it is not as efficient as low-level languages, such as C and FORTRAN, its easy-to-use environment enables ideas to be quickly implemented. Therefore, we chose MATLAB to be the main tool in our work for its quick implementation and powerful manipulation of matrices. All of the aforementioned five algorithms were developed and executed in MATLAB (version 7.0). However, due to efficiency reasons, some of these MATLAB codes further call other solvers (see Section 7.1.2).

[1] http://blogs.unige.ch/hec/logilab/templeet.php/projects.en.html

In MATLAB, *linprog* is a comprehensive linear programming solver. Users can specify one of the solution methods by changing the properties of *linprog*. The large-scale method is based on *lipsol*[2], which is a linear interior-point solver [86] and uses a variant of Mehrotra's predictor-corrector algorithm [70], a primal-dual interior-point method. For medium-scale linear programming problems, *linprog* uses a variant of the well-known simplex method [19]. Both the *lipsol* and simplex solvers ignore any user-supplied starting points.

When comparing the relative efficiency between DW and simplex in Chapter 4, for the direct solution approach (simplex), we call *linprog* to solve the problem as a whole, and for the decomposition approach (DW), we call *linprog* to solve both the master and subproblems.

However, we realized that in our experiments, the DW solution time was surprisingly long. Borchers once pointed out that the success of simplex in practice is mainly attributed to the warm start strategy [11]. Evidently, warm start is crucial for DW, too. The solver not supporting a starting point does not hurt the efficiency of the simplex method because a problem needs to be solved from phase I anyways. On the contrary, for DW, no allowed starting point means no warm start at all, and this made the DW solution time very uncompetitive. Therefore, we modified the MATLAB simplex solver to facilitate a warm start for both the DW master and subproblems. Warm start greatly improved DW's computational efficiency. For example, at the last iteration of a test problem, it used to take more than $8,000$ inner iterations to solve the master problem from scratch, but only took about 15 inner iterations with a warm start point from the previous iteration.

Another problem from the MATLAB simplex solver is that the iteration count that the solver returns does not include the number of iterations in Phase I. The Mathworks explains that this is because for the simplex method, phase I procedure is only to find a feasible initial point. This is undoubtedly correct in terms of the functionality of phase I, but in terms of performance evaluation, the effort put in phase I cannot be ignored. On our 100 stochastic financial test problems, there are $784,819$ phase I iterations in total and $994,920$ phase II iterations. The former accounts for about half of the total number of iterations as well as CPU time, and hence is not negligible. We then modified the simplex

[2]http://www.caam.rice.edu/ zhang/lipsol/

solver again to obtain the total number of iterations in simplex (phase I + phase II).

Theoretically, DW, as a simplex based algorithm, stops when all the reduced costs are positive (for a minimization problem). But in practice, researchers often use another stop criterion, the *relative duality gap*, which can show the progress of each iteration. When the relative duality gap is small enough, the current solution is considered a satisfactory solution. We also use this stopping criterion in our DW code: when the relative duality gap falls below a predetermined small tolerance ε, $\varepsilon = 10^{-6}$, the algorithm stops.

When comparing the relative efficiency between ACCPM and IPM in Chapter 5, for the direct solution approach (IPM), we call the *lipsol* based MATLAB solver to solve the problem as a whole, and for ACCPM, *lipsol* is only used to solve the subproblems. The ACCPM master problem is solved by the primal Newton method; see Section 7.4 for details.

7.1.2 CPLEX

The MATLAB simplex solver is based on the revised simplex method, with the smallest subscript pivoting rule. It has difficulties dealing with large problems, so the most efficient optimization software package, CPLEX (version 10.1), is also adopted in our experiments. CPLEX can automatically detect a problem's characteristics, and then determine a solution method, including primal simplex, dual simplex, primal-dual simplex, network simplex, etc. To avoid confusion, we set the solution method property as *primal simplex*.

CPLEX has considerably better computing efficiency than MATLAB. It can be faster than the MATLAB simplex solver by a factor of over $1,000$, which allows us to finish solving large problems in a reasonable period of time. However, CPLEX involves many advanced techniques, which, on one hand makes it highly efficient, and on the other hand affects our experiments in a negative way. For example, given a block-angular structured problem, we need to compare the relative efficiency of the direct solution approach vs. decomposition (simplex vs. DW), but the Cholesky factorization in CPLEX already takes advantage of the special structure even in the direct approach. In addition, since MATLAB cannot call CPLEX directly, we have to write a C++ code[3], which is called by MATLAB,

[3]The code was initially written by Musicant[73], and was designed for CPLEX 6.5. Because some callable libraries have changed in higher versions of CPLEX, we modified the corresponding part in Musi-

to further call CPLEX solvers. This kind of process causes a huge amount of data transfer, and hurts the decomposition efficiency greatly as MATLAB needs to do the C++ call and then CPLEX, for the master problem as well as for all the subproblems at every iteration. Finally, CPLEX uses the steepest-edge pivoting rule, which has been proven to be very efficient. Consequently, taking DW as a big simplex tableau, a counterpart of the steepest-edge rule should also be employed here. However, DW still uses the simple 'most-negative' reduced cost rule in our experiments. In this sense, the MATLAB solver makes the two approaches more comparable, although it is far less efficient.

Due to the trade-offs between the MATLAB and CPLEX solvers, we constructed two sets of codes: MATLAB based simplex and DW codes[4], and CPLEX based codes[5]. For smaller instances, we try to use the MATLAB based codes, but for larger instances, we have to use the CPLEX based codes. For the numerical results in Chapter 4, unless otherwise specified, problems of the financial model are solved by MATLAB (*linprog*) based codes, and problems of the multicommodity model are solved by CPLEX based codes.

7.1.3 GAMS

GAMS (the General Algebraic Modeling System) is a modeling language and optimization solver. The *algebraic* feature makes it convenient to handle large, complex, one-of-a-kind problems. For example, the 100 stochastic financial test problems can be generated by embedded loops of n and h, the number of assets and the number of scenarios, each from 10 to 100 with a step of 10.

In our experiments, we did not use GAMS as a solver, instead, we only used it to generate test problems. Therefore, it is important for GAMS to be compatible with other software. By a solver named MPSWRITE [33], GAMS can output problems in MPS (Mathematical Programming System), which is a standard input format adopted by most commercial codes.

Since we solve each test problem by both solution approaches, the direct and decom-

cant's code. In addition, we also expanded the code in order to support a warm start. See Appendix B.1 for details.

[4]The DW code calls *linprog* (modified) for both the master and subproblems.

[5]The DW code calls CPLEX for both the master and subproblems.

position, we need to generate these problems in two different forms. It is easy to generate an overall MPS file by MPSWRITE for the direct solution approach. For decomposition algorithms, we generate a series of MPS files, each of which contains a *pseudo-block*, *i.e.*, a block together with its corresponding linking constraints part,

$$\begin{array}{ll} \min & c_k^T x_k \\ \text{s.t.} & A_k x_k = b \\ & B_k x_k = d_k, \quad k = 1, 2 \cdots h \\ & x_k \geq 0 \end{array} \tag{7.1}$$

Yet, these problems do not have any economic interpretations. Rather, they only provide input data, such as A_k, B_k, d_k, and c_k, in a convenient way for future calculations.

However, the default output name from MPSWRITE is 'gams.mps', which cannot be easily changed. As a result, if you generate series of MPS files in a loop, the current one will erase the previous one, and in the end, only the last one remains. After consulting with GAMS, we used the CONVERT utility instead. The gams code to generate series of MPS files for problems with a block-angular structure is provided in Appendix B.2.

The output MPS files are a free format variation of the MPS format, which is compatible with CPLEX but not MATLAB. We need to convert them into the standard MPS format (see Appendix B.3). Since MATLAB does not support MPS format, we further converted them into .mat files (MATLAB binary data format) by a library in *lipsol*.

As for the multicommodity problems, we first tried generating some undirected flow test problems. However, it is difficult to ensure that the problems generated are 'realistic' - even the famous generator, Mnetgen [2], was once criticized as 'too easy' because the solution time with different solvers tend to decrease as the size increases [76][55]. This problem was fixed in the newer enhanced version of Mnetgen generator [30][14], which was recoded in C++. We adopted this generator in our experiments.

7.2 Hardware

All the test problems were executed on a Sun Blade 2500 workstation running the Unix Operating System. On any given problem, we collect the CPU time for the direct solution approaches, *i.e.*, simplex and IPM.

As for decomposition, *i.e.*, DW and ACCPM, it is well known that decomposition algorithms are inherently suited to distributed computing. Our university has a distributed system, flexor, with 52 CPUs in total[6]. Unfortunately, it is impossible for us to reserve the system for a fairly long time. In addition, the number of processors in the cluster is not enough for our experiments: our biggest test problem has as many as 256 blocks. In this case, if implemented in flexor, multiple subproblems will have to be assigned to each node, which makes the algorithm performance study more complex.

Due to the above reasons, we do a simulation of parallel computing for the empirical analysis, assuming that there are plenty of processors in the cluster. Therefore, the CPU time of solving a problem by decomposition is defined as an estimate of solution time in a distributed environment that has one machine for each subproblem, and another processor for the master problem. More specifically, a DW/ACCPM CPU time is collected as the sum of the time for the master problem and the maximum time of subproblems, because when the processor for the master problem is busy, the other processors for subproblems are usually idle, and vice versa [43]. Time for loading data and preprocessing is not included.

7.3 Floating Point Arithmetic for Large-Scale Computing

Most technical computing environments including MATLAB use floating-point arithmetic, which involves a finite set of numbers with finite precision [72]. By default, variables in MATLAB are in the IEEE double precision format, where a real number is expressed in a binary system. In a solution process, a floating-point number is obtained by truncating, rather than rounding. Most of the time, one can use MATLAB effectively without considering these details, but for large-scale computing, accumulated roundoff error caused by truncating leads to poor results.

Due to the large size of our test problems, we encountered great numerical difficulties in our experiments. Therefore, special attention needs to be paid when dealing with large-scale problems. For example, when solving a system of linear equations, instead of

[6]http://www.math.uwaterloo.ca/mfcf/computing-environments/HPC/flexor/hardware.html

computing inverses of matrices, one can use the *back slash* and *forward slash* operators in MATLAB for better efficiency and accuracy; to avoid big digit loss in calculation, one can use scale factors to adjust rows and columns in the coefficient matrix of a problem.

In practice, floating-point arithmetic greatly affects scientific computing. We will discuss more on this topic in Section 7.4.

7.4 Implementation of ACCPM in Decomposition

In this section, we briefly discuss some implementation issues of ACCPM in MATLAB.

7.4.1 Pseudo Code of ACCPM

In short, the ACCPM algorithm can be illustrated as the steps in the following pseudo code:

1. Preallocate and initialize variables
2. Load data
3. Preprocess
4. While the relative duality gap > tolerance
   (a) Solve the weighted master problem (primal Newton method);
   Update the current best upper bound;
   (b) Solve subproblems;
   Update the current best lower bound;
   (c) Correct roundoff error;
   (d) Recover primal feasibility;
   (e) Update big M[7].

[7]In equation (3.19) on page 28 of Chapter 3 , we used M_1 and M_2 to emphasize that the lower and upper bounds of the box constraints can be two different values, but in the implemention of ACCPM, we use $-M \leq \pi \leq M$.

End While

For large problems, preallocating arrays can improve program performance and memory usage. Next, we load and preprocess problems. The *while* loop is the main part of this algorithm. Put simply, we do this loop (one iteration) until the relative duality gap, defined as $\frac{|UB-LB|}{|LB|+1}$, is less than the predetermined small tolerance.

At each iteration, we solve the weighted master problem based on ideas from IPM theory. More specifically, the primal Newton method is used to solve for a search direction, and the step size is determined by the 'bi-section' technique [84]. When solving the subproblems, we distinguish whether a proposal (cut) is repeated: if yes, increase its corresponding weight by one; if no, add a new column in the restricted master problem. Next, we recover a primal feasible solution as the start point for the next iteration. For details, see [24].

To deal with the aforementioned roundoff error in floating-point computing, we employed a few special techniques in our codes.

First, we normalized extreme rays. It is worth noting that when solving subproblems, if extreme rays occur, the MATLAB solver returns a flag to indicate the unboundedness, and returns a big-valued vector, which often causes numerical difficulties. Therefore, we normalize all the extreme ray vectors, *i.e.*, we use $\frac{V_{ray}}{||V_{ray}||}$ for future calculations.

Second, we used force-out factors in the master problem. Theoretically, a primal Newton step in the master problem, denoted as $\Delta\widetilde{x}$, satisfies $\widetilde{A}\Delta\widetilde{x} = 0$, but practically, this primal feasibility is often violated due to gradually accumulated roundoff error. Once the direction $\Delta\widetilde{x}$ is not feasible to the master problem, a false upper bound will be generated, and then it is possible that the ACCPM code fails to solve the original problem. Therefore, we check the primal feasibility at every inner iteration in the master problem. We force the primal Newton method to stop if infeasibility occurs, use the current, still feasible point to update the upper bound, and then continue to solve the subproblems. On 100 test problems of the stochastic financial model, there were 431 out of 682 (63.2%) iterations forced out of the primal Newton loop; on 132 test problems of the multicommodity network flow problem, 1057 out of 2512 (42.1%) iterations were forced out.

Third, we corrected roundoff errors. Besides the inner feasibility *check*, we also try to *fix* the roundoff error after a few iterations. Denote the error vector as $err = |\widetilde{A}\widetilde{x}|$.

If the maximum element in err, *i.e.*, $max(err)$, is greater than 10^{-4}, then adjust the corresponding artificial variables in $\widetilde{x}$ by the amount of err. We call this process *correcting roundoff error*. After the correction, we usually have $|\widetilde{A}\widetilde{x}| < 10^{-10}$. On 100 test problems of the stochastic financial model, 471 out of 682 (69.1%) iterations called the roundoff-error-correction process; on 132 test problems of the multicommodity network flow problem, the figures were 1474 out of 2512 (58.7%).

7.4.2 Penalty Weighted Technique

When implementing ACCPM, we realized that on many test problems, there were no new cuts added over the last few iterations. In other words, all the cuts were repeated at that time. Algebraically, adding a repeated cut is equivalent to increasing its weight by one. Then, for constantly repeated cuts, can we push it more to improve ACCPM's convergence?

Recall that in the ACCPM master problem, a weighting mechanism does not affect a solution's primal feasibility. Therefore, we tried three different weighting strategies: the first is the ordinary one, $N = N + 1$, denoted as $w1$; the second, $N = 2 * N$, denoted as $w2$; the third, if $N = 1$, $N = N + 1$, else $N = N^2$, denoted as $w3$. The numerical results are shown in Table 7.1. We call the second and third weighting techniques 'penalty' as the weights are increased more if a cut is repeated.

At times, a penalty weight can accelerate the ACCPM code, but on average, the three weighting mechanisms are comparable.

7.4.3 Dynamic Update of M

The ACCPM code involves many parameters, among which M plays an important role in the process. M is supposed to have a big value. In the ACCPM master problem (primal space), M is associated with the artificial variables, so the big value of M, which is $>> \widetilde{c}$, makes the artificial variables to be zero whenever possible. If M tends to infinity, the artificial variables tend to zero. In this sense, the bigger the M, the more accurate the results. However, for floating-point arithmetic, big M leads to some tiny-valued variables, and hence the problem becomes 'badly scaled'. People usually choose a value that is moderately bigger than all other coefficients. Unfortunately, in some cases, it is very difficult to

Table 7.1: Different Weighting Strategies

Prob.	# of iterations			CPU time			Duality gap		
	$w1$	$w2$	$w3$	$w1$	$w2$	$w3$	$w1$	$w2$	$w3$
1	12	10	10	0.94	1.01	0.71	4.02E-05	8.21E-05	1.84E-04
2	7	10	9	0.74	1.04	0.93	1.72E-04	1.18E-04	1.25E-04
3	6	9	7	1.13	1.57	1.24	2.55E-04	1.74E-04	2.17E-04
4	10	14	9	3.02	4.08	2.79	3.11E-04	3.37E-04	5.15E-04
5	13	11	10	3.56	1.81	3.41	3.39E-04	2.93E-04	3.29E-04
6	9	9	8	2.86	2.98	2.65	2.25E-04	2.33E-04	3.28E-04
7	10	7	10	6.06	4.48	6.12	2.83E-04	3.72E-04	3.59E-04
8	13	12	10	11	10.44	8.49	5.06E-04	3.78E-04	6.39E-04
9	16	13	10	5.64	4.89	4.66	7.34E-05	6.18E-04	5.90E-04
10	10	9	10	18.32	21.12	27.76	5.85E-04	4.42E-04	6.35E-04
11	10	9	8	12.52	11.61	10.57	5.19E-04	3.08E-04	4.74E-04
12	7	7	12	14.51	14.48	22.68	6.96E-04	1.01E-03	4.75E-04
13	13	16	13	7.91	8.74	7.09	1.58E-04	1.19E-04	6.91E-04
14	10	11	9	14.22	14.98	12.71	2.68E-04	3.75E-04	3.66E-04
15	8	11	8	18.88	25.15	18.96	8.33E-04	5.52E-04	7.74E-04
16	9	9	8	32.25	30.84	28.02	5.64E-04	1.13E-03	8.97E-04
Average	10.19	10.44	9.44	9.60	9.95	9.92	3.64E-04	4.09E-04	4.75E-04

estimate the scale of all other coefficients, especially for decomposition algorithms, where the coefficients in the master problem do not have explicit economic interpretations. To overcome this problem, some researchers try to gradually update M during the process, so that a better estimation can be made. For example, see [60].

Recall the ACCPM master problem in the dual space, where M corresponds to the box constraints, $-M \leq \widetilde{y} \leq M$. As one of the cutting plane methods, ACCPM approaches an optimal solution by generating more and more cuts. In other words, with more and more cuts added, the feasible region becomes smaller and smaller, and eventually shrinks to a satisfactory solution. Therefore, the feasible region from the current iteration contains all the feasible solutions in the next iteration, *i.e.*, $\widetilde{y}_k \supseteq \widetilde{y}_{k+1}$. This feature enables us to

Table 7.2: Dynamic Update of M

Prob.	# of iterations			CPU time			Duality gap			M_{dyn}
	10^3	10^5	*dyn*	10^3	10^5	*dyn*	10^3	10^5	*dyn*	
1	7	7	7	0.44	0.5	0.59	5.47E-04	2.25E-04	2.95E-04	93
2	5	6	5	0.58	0.77	0.6	1.79E-04	8.17E-04	8.80E-04	99
3	5	5	4	0.96	1.14	0.86	2.55E-04	3.60E-04	6.37E-04	100
4	6	5	5	1.87	1.98	1.69	4.60E-04	7.18E-04	7.53E-04	100
5	9	9	9	1.61	1.59	1.47	5.29E-04	7.23E-04	9.64E-04	86
6	7	6	6	2.33	2.57	2.16	5.81E-04	8.43E-04	1.94E-04	98
7	6	14	6	3.87	9.71	3.74	9.01E-04	1.60E-03	5.05E-04	100
8	8	5	7	6.7	5.3	6.39	9.72E-04	6.51E-04	6.55E-04	100
9	10	11	10	3.8	5.32	4.24	7.57E-04	7.90E-04	7.84E-04	59
10	7	10	11	7.07	10.61	9.48	7.45E-04	2.91E-03	1.06E-02	97
11	6	5	5	8.09	7.35	6.55	7.76E-04	6.32E-04	7.43E-04	99
12	6	6	7	12.74	14.54	12.58	6.98E-04	4.64E-03	9.63E-04	100
13	8	10	10	5.56	8.13	6.68	9.99E-04	8.04E-04	8.26E-04	41
14	6	8	7	9.2	12.59	9.92	9.88E-04	3.62E-03	7.62E-04	97
15	7	6	6	16.67	17.42	14.09	8.36E-04	9.37E-04	6.96E-04	99
16	5	17	6	19.73	67.42	21.38	8.51E-04	2.33E-03	7.44E-04	100
Average	6.75	8.13	6.94	6.33	10.43	6.40	6.92E-04	1.41E-03	1.31E-03	91.75

estimate M_{k+1}, the box constraints, according to $\widetilde{y}_k$. Table 7.2 compares the influence by different M values: $M = 10^3$, $M = 10^5$, and M with dynamically updated values. By dynamic updating, it means that starting with $M = 10^5$, $M_{k+1} = 2||\widetilde{y}_k||$. The constant, 2, can be any other user specified values. The last column in Table 7.2 shows the actual value of M at the end of the process.

We can see that $M = 10^3$ generates better results than $M = 10^5$. In fact, with bigger values, such as $M = 10^8$, some test problems failed to achieve a satisfactory solution. With the dynamic updating technique, for problems that are difficult to estimate solution range, we can start with a big value of M, without worrying about the numerical problems caused by the big value, as M will automatically become a moderate number according to the

problem coefficients. As shown in the last column of Table 7.2, M started as 10^5, but was much smaller in the end.

Chapter 8

Concluding Remarks and Future Research

We try to understand how algorithmic performance varies in response to dimensional parameters. A few preliminary conclusions have been drawn based on our experiments. Most of the regression models have strong explanatory capability. However, the conclusions are quite different between the two test problem classes. Although the models have different economic interpretations, both have block-angular structures. Therefore, it seems that any general predictions for all models will require more information than just the dimensional parameters of the block diagonal structure. For now, we have to be content with predictions for instances within a class of models.

We have shown that empirical analysis can provide us valuable hints on how algorithms would behave in practice, and hence help us choose the right solver for a given problem. For problems that need to be solved repeatedly, people can use historical data to aid future choice of algorithms. Our work is just an attempt of empirical analysis applied in the optimization field. It is our hope that this work motivates further studies along the same lines, so that more people can realize that empirical science is as important as deductive analysis.

In the future, if time permits, this kind of empirical analysis can be conducted with more numerical examples, which have been collected in several libraries accessible to researchers. For example, about ten multicommodity network flow models in various applications are

collected and well documented in a university website[1]. For more rigorous experimental design to evaluate algorithms' performance, one can also generate their own test problems, so that the characteristics can be flexibly controlled.

We wrote all the codes in MATLAB (calling modified *linprog* and CPLEX) for quick implementation, but the tradeoff is the efficiency. Generally speaking, for scientific computing, MATLAB is not as fast as the low level languages, such as C/C++ and FORTRAN. In addition, developing an efficient general-purpose decomposition code is a worthy goal to work toward.

As a new and promising solution method, ACCPM has good convergence rate, but its accuracy is not ideal. We try to improve our implementation of ACCPM by a few techniques, including the hybrid decomposition approach, different weighting strategies, and the dynamic update of M. In addition to these, for large-scale applications, dropping some cuts once in a while can reduce the size of the master problem, but it is computationally expensive to do the redundancy check itself [23]. Therefore, this is still an open topic.

In our experiments, we used a single server, and simulated a parallel computing paradigm. In a real parallel computing environment, more factors such as the overhead, inter communications between processors, and load balancing need to be carefully addressed.

[1] http://www.di.unipi.it/optimize/Data/MMCF.html

Appendix A

Problem Characteristics

Table A.1: Stochastic Financial Problem Characteristics

Prob.	n	h	m_0	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
1	10	10	11	321	3.43%	111	1,531	4.30%	31	11
2	10	20	11	1,031	1.07%	421	5,651	1.30%	51	21
3	10	30	11	2,141	0.51%	931	12,371	0.62%	71	31
4	10	40	11	3,651	0.30%	1,641	21,691	0.36%	91	41
5	10	50	11	5,561	0.20%	2,551	33,611	0.24%	111	51
6	10	60	11	7,871	0.14%	3,661	48,131	0.17%	131	61
7	10	70	11	10,581	0.10%	4,971	65,251	0.12%	151	71
8	10	80	11	13,691	0.08%	6,481	84,971	0.10%	171	81
9	10	90	11	17,201	0.06%	8,191	107,290	0.08%	191	91
10	10	100	11	21,111	0.05%	10,101	132,210	0.06%	211	101
11	20	10	21	431	4.87%	111	2,741	5.73%	41	11
12	20	20	21	1,241	1.69%	421	10,061	1.93%	61	21
13	20	30	21	2,451	0.86%	931	21,981	0.96%	81	31
14	20	40	21	4,061	0.52%	1,641	38,501	0.58%	101	41
15	20	50	21	6,071	0.35%	2,551	59,621	0.38%	121	51
16	20	60	21	8,481	0.25%	3,661	85,341	0.27%	141	61

Continued on next page

Table A.1 – continued from previous page

Prob.	n	h	m_0	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
17	20	70	21	11,291	0.19%	4,971	115,660	0.21%	161	71
18	20	80	21	14,501	0.14%	6,481	150,580	0.16%	181	81
19	20	90	21	18,111	0.12%	8,191	190,100	0.13%	201	91
20	20	100	21	22,121	0.09%	10,101	234,220	0.10%	221	101
21	30	10	31	541	5.73%	111	3,951	6.58%	51	11
22	30	20	31	1,451	2.14%	421	14,471	2.37%	71	21
23	30	30	31	2,761	1.12%	931	31,591	1.23%	91	31
24	30	40	31	4,471	0.69%	1,641	55,311	0.75%	111	41
25	30	50	31	6,581	0.47%	2,551	85,631	0.51%	131	51
26	30	60	31	9,091	0.34%	3,661	122,550	0.37%	151	61
27	30	70	31	12,001	0.26%	4,971	166,070	0.28%	171	71
28	30	80	31	15,311	0.20%	6,481	216,190	0.22%	191	81
29	30	90	31	19,021	0.16%	8,191	272,910	0.18%	211	91
30	30	100	31	23,131	0.13%	10,101	336,230	0.14%	231	101
31	40	10	41	651	6.30%	111	5,161	7.14%	61	11
32	40	20	41	1,661	2.47%	421	18,881	2.70%	81	21
33	40	30	41	3,071	1.34%	931	41,201	1.44%	101	31
34	40	40	41	4,881	0.84%	1,641	72,121	0.90%	121	41
35	40	50	41	7,091	0.58%	2,551	111,640	0.62%	141	51
36	40	60	41	9,701	0.42%	3,661	159,760	0.45%	161	61
37	40	70	41	12,711	0.32%	4,971	216,480	0.34%	181	71
38	40	80	41	16,121	0.25%	6,481	281,800	0.27%	201	81
39	40	90	41	19,931	0.21%	8,191	355,720	0.22%	221	91
40	40	100	41	24,141	0.17%	10,101	438,240	0.18%	241	101
41	50	10	51	761	6.70%	111	6,371	7.54%	71	11
42	50	20	51	1,871	2.73%	421	23,291	2.96%	91	21
43	50	30	51	3,381	1.51%	931	50,811	1.61%	111	31
44	50	40	51	5,291	0.96%	1,641	88,931	1.02%	131	41

Continued on next page

Table A.1 – continued from previous page

Prob.	n	h	m_0	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
45	50	50	51	7,601	0.67%	2,551	137,650	0.71%	151	51
46	50	60	51	10,311	0.49%	3,661	196,970	0.52%	171	61
47	50	70	51	13,421	0.38%	4,971	266,890	0.40%	191	71
48	50	80	51	16,931	0.30%	6,481	347,410	0.32%	211	81
49	50	90	51	20,841	0.24%	8,191	438,530	0.26%	231	91
50	50	100	51	25,151	0.20%	10,101	540,250	0.21%	251	101
51	60	10	61	871	7.00%	111	7,581	7.84%	81	11
52	60	20	61	2,081	2.93%	421	27,701	3.16%	101	21
53	60	30	61	3,691	1.65%	931	60,421	1.76%	121	31
54	60	40	61	5,701	1.07%	1,641	105,740	1.13%	141	41
55	60	50	61	8,111	0.75%	2,551	163,660	0.79%	161	51
56	60	60	61	10,921	0.56%	3,661	234,180	0.59%	181	61
57	60	70	61	14,131	0.43%	4,971	317,300	0.45%	201	71
58	60	80	61	17,741	0.34%	6,481	413,020	0.36%	221	81
59	60	90	61	21,751	0.28%	8,191	521,340	0.29%	241	91
60	60	100	61	26,161	0.23%	10,101	642,260	0.24%	261	101
61	70	10	71	981	7.24%	111	8,791	8.07%	91	11
62	70	20	71	2,291	3.10%	421	32,111	3.33%	111	21
63	70	30	71	4,001	1.77%	931	70,031	1.88%	131	31
64	70	40	71	6,111	1.16%	1,641	122,550	1.22%	151	41
65	70	50	71	8,621	0.82%	2,551	189,670	0.86%	171	51
66	70	60	71	11,531	0.62%	3,661	271,390	0.64%	191	61
67	70	70	71	14,841	0.48%	4,971	367,710	0.50%	211	71
68	70	80	71	18,551	0.38%	6,481	478,630	0.40%	231	81
69	70	90	71	22,661	0.31%	8,191	604,150	0.33%	251	91
70	70	100	71	27,171	0.26%	10,101	744,270	0.27%	271	101
71	80	10	81	1,091	7.42%	111	10,001	8.26%	101	11
72	80	20	81	2,501	3.24%	421	36,521	3.47%	121	21

Continued on next page

Table A.1 – continued from previous page

Prob.	n	h	m_0	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
73	80	30	81	4,311	1.88%	931	79,641	1.98%	141	31
74	80	40	81	6,521	1.24%	1,641	139,360	1.30%	161	41
75	80	50	81	9,131	0.89%	2,551	215,680	0.93%	181	51
76	80	60	81	12,141	0.67%	3,661	308,600	0.69%	201	61
77	80	70	81	15,551	0.52%	4,971	418,120	0.54%	221	71
78	80	80	81	19,361	0.42%	6,481	544,240	0.43%	241	81
79	80	90	81	23,571	0.34%	8,191	686,960	0.36%	261	91
80	80	100	81	28,181	0.29%	10,101	846,280	0.30%	281	101
81	90	10	91	1,201	7.58%	111	11,211	8.41%	111	11
82	90	20	91	2,711	3.36%	421	40,931	3.59%	131	21
83	90	30	91	4,621	1.97%	931	89,251	2.07%	151	31
84	90	40	91	6,931	1.31%	1,641	156,170	1.37%	171	41
85	90	50	91	9,641	0.94%	2,551	241,690	0.98%	191	51
86	90	60	91	12,751	0.71%	3,661	345,810	0.74%	211	61
87	90	70	91	16,261	0.56%	4,971	468,530	0.58%	231	71
88	90	80	91	20,171	0.45%	6,481	609,850	0.47%	251	81
89	90	90	91	24,481	0.37%	8,191	769,770	0.38%	271	91
90	90	100	91	29,191	0.31%	10,101	948,290	0.32%	291	101
91	100	10	101	1,311	7.70%	111	12,421	8.54%	121	11
92	100	20	101	2,921	3.46%	421	45,341	3.69%	141	21
93	100	30	101	4,931	2.05%	931	98,861	2.15%	161	31
94	100	40	101	7,341	1.38%	1,641	172,980	1.44%	181	41
95	100	50	101	10,151	0.99%	2,551	267,700	1.03%	201	51
96	100	60	101	13,361	0.76%	3,661	383,020	0.78%	221	61
97	100	70	101	16,971	0.60%	4,971	518,940	0.62%	241	71
98	100	80	101	20,981	0.48%	6,481	675,460	0.50%	261	81
99	100	90	101	25,391	0.40%	8,191	852,580	0.41%	281	91
100	100	100	101	30,201	0.33%	10,101	1,050,300	0.34%	301	101

Table A.2: Multicommodity Network Flow Problem Characteristics

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
1	64	4	196	84	340	24.71%	720	1,754	0.7165%	64	180
2	64	4	200	79	335	23.58%	721	1,721	0.7125%	64	180
3	64	4	189	85	341	24.93%	722	1,773	0.7201%	64	181
4	64	4	194	146	402	36.32%	721	1,982	0.6838%	64	180
5	64	4	201	167	423	39.48%	720	2,039	0.6695%	64	180
6	64	4	203	157	413	38.02%	722	1,997	0.6697%	64	181
7	64	4	520	195	451	43.24%	1,537	3,653	0.5270%	64	384
8	64	4	528	207	463	44.71%	1,539	3,684	0.5170%	64	385
9	64	4	532	227	483	47.00%	1,536	3,726	0.5022%	64	384
10	64	4	537	433	689	62.85%	1,539	4,335	0.4088%	64	385
11	64	4	522	410	666	61.56%	1,539	4,288	0.4184%	64	385
12	64	4	524	407	663	61.39%	1,537	4,270	0.4190%	64	384
13	64	8	200	83	595	13.95%	1,433	3,448	0.4044%	64	179
14	64	8	192	72	584	12.33%	1,433	3,388	0.4048%	64	179
15	64	8	198	104	616	16.88%	1,437	3,626	0.4096%	64	180
16	64	8	195	147	659	22.31%	1,435	3,978	0.4207%	64	179
17	64	8	203	163	675	24.15%	1,439	4,007	0.4125%	64	180
18	64	8	191	148	660	22.42%	1,437	3,980	0.4197%	64	180
19	64	8	541	231	743	31.09%	3,004	7,319	0.3279%	64	376
20	64	8	557	230	742	31.00%	3,006	7,207	0.3231%	64	376
21	64	8	531	212	724	29.28%	3,002	7,176	0.3302%	64	375
22	64	8	561	447	959	46.61%	3,006	8,375	0.2905%	64	376
23	64	8	553	435	947	45.94%	3,000	8,367	0.2945%	64	375
24	64	8	532	425	937	45.36%	3,000	8,419	0.2995%	64	375
25	64	16	187	67	1,091	6.14%	2,872	6,766	0.2159%	64	180

Continued on next page

Table A.2 – continued from previous page

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
26	64	16	208	79	1,103	7.16%	2,865	6,865	0.2172%	64	179
27	64	16	199	78	1,102	7.08%	2,865	6,845	0.2168%	64	179
28	64	16	202	148	1,172	12.63%	2,866	7,830	0.2331%	64	179
29	64	16	185	140	1,164	12.03%	2,865	7,875	0.2361%	64	179
30	64	16	193	163	1,187	13.73%	2,833	8,050	0.2394%	64	177
31	64	16	515	206	1,230	16.75%	5,501	13,121	0.1939%	64	344
32	64	16	522	201	1,225	16.41%	5,501	13,170	0.1954%	64	344
33	64	16	529	216	1,240	17.42%	5,500	13,190	0.1934%	64	344
34	64	16	514	393	1,417	27.74%	5,506	15,195	0.1948%	64	344
35	64	16	520	411	1,435	28.64%	5,511	15,388	0.1946%	64	344
36	64	16	497	391	1,415	27.63%	5,507	15,291	0.1962%	64	344
37	64	32	208	84	2,132	3.94%	5,738	13,935	0.1139%	64	179
38	64	32	200	73	2,121	3.44%	5,743	13,632	0.1119%	64	179
39	64	32	187	71	2,119	3.35%	5,737	13,663	0.1124%	64	179
40	64	32	200	151	2,199	6.87%	5,736	15,761	0.1250%	64	179
41	64	32	201	157	2,205	7.12%	5,748	15,990	0.1262%	64	180
42	64	32	196	171	2,219	7.71%	5,756	16,545	0.1295%	64	180
43	64	32	525	218	2,266	9.62%	11,002	26,695	0.1071%	64	344
44	64	32	550	222	2,270	9.78%	11,004	26,530	0.1062%	64	344
45	64	32	537	218	2,266	9.62%	11,019	26,663	0.1068%	64	344
46	64	32	520	418	2,466	16.95%	11,017	30,890	0.1137%	64	344
47	64	32	511	398	2,446	16.27%	11,006	30,439	0.1131%	64	344
48	64	32	509	404	2,452	16.48%	11,000	30,569	0.1133%	64	344
49	64	64	211	89	4,185	2.13%	11,490	27,798	0.0578%	64	180
50	64	64	199	79	4,175	1.89%	11,489	27,372	0.0571%	64	180
51	64	64	217	92	4,188	2.20%	11,490	27,758	0.0577%	64	180
52	64	64	201	160	4,256	3.76%	11,495	32,099	0.0656%	64	180
53	64	64	192	154	4,250	3.62%	11,458	32,143	0.0660%	64	179

Continued on next page

Table A.2 – continued from previous page

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
54	64	64	198	166	4,262	3.89%	11,509	32,688	0.0666%	64	180
55	64	64	521	206	4,302	4.79%	22,012	52,457	0.0554%	64	344
56	64	64	517	195	4,291	4.54%	22,009	52,616	0.0557%	64	344
57	64	64	509	186	4,282	4.34%	22,018	51,887	0.0550%	64	344
58	64	64	539	437	4,533	9.64%	22,048	61,777	0.0618%	64	345
59	64	64	522	424	4,520	9.38%	22,039	62,125	0.0624%	64	344
60	64	64	511	405	4,501	9.00%	22,004	61,519	0.0621%	64	344
61	128	4	388	165	677	24.37%	1,441	3,493	0.3581%	128	360
62	128	4	391	168	680	24.71%	1,441	3,504	0.3576%	128	360
63	128	4	386	158	670	23.58%	1,442	3,485	0.3607%	128	361
64	128	4	378	305	817	37.33%	1,442	4,055	0.3442%	128	361
65	128	4	379	307	819	37.49%	1,440	4,053	0.3437%	128	360
66	128	4	383	312	824	37.86%	1,440	4,055	0.3417%	128	360
67	128	4	1,017	398	910	43.74%	3,000	7,216	0.2643%	128	750
68	128	4	1,010	410	922	44.47%	3,000	7,224	0.2612%	128	750
69	128	4	989	398	910	43.74%	3,000	7,222	0.2645%	128	750
70	128	4	993	805	1,317	61.12%	3,003	8,456	0.2138%	128	751
71	128	4	1,011	818	1,330	61.50%	3,001	8,428	0.2112%	128	750
72	128	4	997	799	1,311	60.95%	3,000	8,402	0.2136%	128	750
73	128	8	392	163	1,187	13.73%	2,884	6,983	0.2040%	128	361
74	128	8	396	160	1,184	13.51%	2,880	6,945	0.2037%	128	360
75	128	8	388	166	1,190	13.95%	2,884	7,018	0.2045%	128	361
76	128	8	392	304	1,328	22.89%	2,886	8,027	0.2094%	128	361
77	128	8	391	313	1,337	23.41%	2,882	8,045	0.2088%	128	360
78	128	8	384	303	1,327	22.83%	2,886	8,044	0.2100%	128	361
79	128	8	1,066	440	1,464	30.06%	6,004	14,489	0.1648%	128	751
80	128	8	1,084	442	1,466	30.15%	6,004	14,439	0.1640%	128	751
81	128	8	1,075	468	1,492	31.37%	6,005	14,591	0.1629%	128	751

Continued on next page

Table A.2 – continued from previous page

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
82	128	8	1,092	848	1,872	45.30%	6,002	16,672	0.1484%	128	750
83	128	8	1,076	860	1,884	45.65%	6,004	16,754	0.1481%	128	751
84	128	8	1,089	851	1,875	45.39%	6,006	16,701	0.1483%	128	751
85	128	16	404	161	2,209	7.29%	5,761	13,828	0.1087%	128	360
86	128	16	410	166	2,214	7.50%	5,770	13,900	0.1088%	128	361
87	128	16	414	183	2,231	8.20%	5,769	14,051	0.1092%	128	361
88	128	16	387	316	2,364	13.37%	5,760	16,242	0.1193%	128	360
89	128	16	383	315	2,363	13.33%	5,775	16,300	0.1195%	128	361
90	128	16	390	318	2,366	13.44%	5,769	16,241	0.1190%	128	361
91	128	16	1,154	455	2,503	18.18%	12,001	28,813	0.0959%	128	750
92	128	16	1,154	469	2,517	18.63%	12,008	28,935	0.0957%	128	751
93	128	16	1,138	487	2,535	19.21%	12,006	29,132	0.0957%	128	750
94	128	16	1,122	887	2,935	30.22%	12,000	33,530	0.0952%	128	750
95	128	16	1,123	933	2,981	31.30%	12,007	33,834	0.0945%	128	750
96	128	16	1,114	911	2,959	30.79%	12,002	33,686	0.0949%	128	750
97	128	32	399	150	4,246	3.53%	11,538	27,456	0.0560%	128	361
98	128	32	400	156	4,252	3.67%	11,528	27,509	0.0561%	128	360
99	128	32	405	158	4,254	3.71%	11,521	27,497	0.0561%	128	360
100	128	32	413	325	4,421	7.35%	11,527	32,131	0.0631%	128	360
101	128	32	412	313	4,409	7.10%	11,527	31,898	0.0628%	128	360
102	128	32	413	325	4,421	7.35%	11,525	32,051	0.0629%	128	360
103	128	32	1,178	462	4,558	10.14%	24,009	57,390	0.0524%	128	750
104	128	32	1,127	447	4,543	9.84%	24,030	57,545	0.0527%	128	751
105	128	32	1,138	463	4,559	10.16%	24,005	57,866	0.0529%	128	750
106	128	32	1,147	918	5,014	18.31%	24,002	67,113	0.0558%	128	750
107	128	32	1,173	945	5,041	18.75%	24,002	67,172	0.0555%	128	750
108	128	32	1,141	906	5,002	18.11%	24,007	66,983	0.0558%	128	750
109	128	64	388	158	8,350	1.89%	23,038	55,360	0.0288%	128	360

Continued on next page

Table A.2 – continued from previous page

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
110	128	64	394	160	8,352	1.92%	23,050	55,392	0.0288%	128	360
111	128	64	381	151	8,343	1.81%	23,046	55,346	0.0288%	128	360
112	128	64	412	338	8,530	3.96%	23,020	64,935	0.0331%	128	360
113	128	64	406	331	8,523	3.88%	23,012	64,872	0.0331%	128	360
114	128	64	397	323	8,515	3.79%	23,040	64,833	0.0330%	128	360
115	128	64	1,184	482	8,674	5.56%	48,018	115,020	0.0276%	128	750
116	128	64	1,154	457	8,649	5.28%	48,033	115,320	0.0278%	128	751
117	128	64	1,138	455	8,647	5.26%	48,009	115,520	0.0278%	128	750
118	128	64	1,182	947	9,139	10.36%	48,019	134,770	0.0307%	128	750
119	128	64	1,201	967	9,159	10.56%	48,004	134,820	0.0307%	128	750
120	128	64	1,171	936	9,128	10.25%	48,015	134,360	0.0307%	128	750
121	128	128	396	159	16,543	0.96%	46,004	110,390	0.0145%	128	359
122	128	128	399	158	16,542	0.96%	46,033	110,170	0.0145%	128	360
123	128	128	388	151	16,535	0.91%	46,040	109,990	0.0144%	128	360
124	128	128	412	338	16,722	2.02%	46,016	130,130	0.0169%	128	360
125	128	128	401	325	16,709	1.95%	46,034	129,460	0.0168%	128	360
126	128	128	407	336	16,720	2.01%	46,065	130,340	0.0169%	128	360
127	128	128	1,221	502	16,886	2.97%	98,035	235,900	0.0143%	128	766
128	128	128	1,224	486	16,870	2.88%	98,032	234,020	0.0142%	128	766
129	128	128	1,212	479	16,863	2.84%	98,034	234,180	0.0142%	128	766
130	128	128	1,213	968	17,352	5.58%	98,038	275,780	0.0162%	128	766
131	128	128	1,223	975	17,359	5.62%	98,015	275,200	0.0162%	128	766
132	128	128	1,204	979	17,363	5.64%	98,017	277,010	0.0163%	128	766
133	256	4	771	297	1,321	22.48%	2,900	6,939	0.1811%	256	725
134	256	4	783	303	1,327	22.83%	2,901	6,933	0.1801%	256	725
135	256	4	784	309	1,333	23.18%	2,901	6,946	0.1796%	256	725
136	256	4	774	628	1,652	38.02%	2,900	8,154	0.1702%	256	725
137	256	4	784	635	1,659	38.28%	2,902	8,145	0.1692%	256	726

Continued on next page

Table A.2 – continued from previous page

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
138	256	4	785	636	1,660	38.31%	2,901	8,154	0.1693%	256	725
139	256	4	2,007	805	1,829	44.01%	6,000	14,428	0.1315%	256	1,500
140	256	4	2,034	795	1,819	43.71%	6,003	14,335	0.1313%	256	1,501
141	256	4	2,022	822	1,846	44.53%	6,000	14,397	0.1300%	256	1,500
142	256	4	2,027	1,650	2,674	61.71%	6,000	16,906	0.1054%	256	1,500
143	256	4	2,013	1,616	2,640	61.21%	6,001	16,837	0.1063%	256	1,500
144	256	4	2,023	1,633	2,657	61.46%	6,000	16,857	0.1057%	256	1,500
145	256	8	796	327	2,375	13.77%	5,800	13,975	0.1015%	256	725
146	256	8	797	332	2,380	13.95%	5,802	14,019	0.1015%	256	725
147	256	8	793	319	2,367	13.48%	5,801	13,982	0.1018%	256	725
148	256	8	795	648	2,696	24.04%	5,804	16,375	0.1047%	256	726
149	256	8	806	651	2,699	24.12%	5,801	16,349	0.1044%	256	725
150	256	8	794	651	2,699	24.12%	5,802	16,410	0.1048%	256	725
151	256	8	2,152	919	2,967	30.97%	12,002	29,064	0.0816%	256	1,500
152	256	8	2,194	904	2,952	30.62%	12,000	28,974	0.0818%	256	1,500
153	256	8	2,168	891	2,939	30.32%	12,007	28,937	0.0820%	256	1,501
154	256	8	2,187	1,787	3,835	46.60%	12,005	33,854	0.0735%	256	1,501
155	256	8	2,179	1,761	3,809	46.23%	12,002	33,808	0.0740%	256	1,500
156	256	8	2,165	1,764	3,812	46.28%	12,007	33,850	0.0740%	256	1,501
157	256	16	804	326	4,422	7.37%	11,609	27,964	0.0545%	256	726
158	256	16	799	331	4,427	7.48%	11,606	28,036	0.0546%	256	725
159	256	16	795	331	4,427	7.48%	11,600	28,026	0.0546%	256	725
160	256	16	824	650	4,746	13.70%	11,603	32,355	0.0588%	256	725
161	256	16	820	649	4,745	13.68%	11,614	32,363	0.0587%	256	726
162	256	16	850	674	4,770	14.13%	11,604	32,380	0.0585%	256	725
163	256	16	2,252	920	5,016	18.34%	24,004	58,016	0.0482%	256	1,500
164	256	16	2,316	950	5,046	18.83%	24,007	57,895	0.0478%	256	1,500
165	256	16	2,271	931	5,027	18.52%	24,005	57,826	0.0479%	256	1,500

Continued on next page

Table A.2 – continued from previous page

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
166	256	16	2,275	1,801	5,897	30.54%	24,006	67,090	0.0474%	256	1,500
167	256	16	2,325	1,868	5,964	31.32%	24,009	67,134	0.0469%	256	1,501
168	256	16	2,308	1,842	5,938	31.02%	24,000	67,106	0.0471%	256	1,500
169	256	32	820	336	8,528	3.94%	23,213	55,974	0.0283%	256	725
170	256	32	817	332	8,524	3.89%	23,217	55,913	0.0283%	256	726
171	256	32	812	337	8,529	3.95%	23,202	56,020	0.0283%	256	725
172	256	32	812	639	8,831	7.24%	23,200	64,454	0.0315%	256	725
173	256	32	810	649	8,841	7.34%	23,204	64,666	0.0315%	256	725
174	256	32	837	653	8,845	7.38%	23,222	64,434	0.0314%	256	726
175	256	32	2,347	947	9,139	10.36%	48,013	115,560	0.0263%	256	1,500
176	256	32	2,318	937	9,129	10.26%	48,004	115,760	0.0264%	256	1,500
177	256	32	2,327	958	9,150	10.47%	48,008	115,790	0.0264%	256	1,500
178	256	32	2,331	1,860	10,052	18.50%	48,008	133,810	0.0277%	256	1,500
179	256	32	2,318	1,849	10,041	18.42%	48,011	134,260	0.0279%	256	1,500
180	256	32	2,314	1,829	10,021	18.25%	48,005	133,790	0.0278%	256	1,500
181	256	64	811	318	16,702	1.90%	46,438	110,970	0.0143%	256	726
182	256	64	836	338	16,722	2.02%	46,404	111,150	0.0143%	256	725
183	256	64	827	334	16,718	2.00%	46,447	111,430	0.0144%	256	726
184	256	64	794	649	17,033	3.81%	46,406	130,720	0.0165%	256	725
185	256	64	808	655	17,039	3.84%	46,425	130,520	0.0165%	256	725
186	256	64	801	646	17,030	3.79%	46,435	130,520	0.0165%	256	726
187	256	64	2,318	944	17,328	5.45%	96,006	231,100	0.0139%	256	1,500
188	256	64	2,330	912	17,296	5.27%	96,037	228,970	0.0138%	256	1,501
189	256	64	2,362	966	17,350	5.57%	96,017	230,960	0.0139%	256	1,500
190	256	64	2,336	1,867	18,251	10.23%	96,012	268,700	0.0153%	256	1,500
191	256	64	2,334	1,865	18,249	10.22%	96,006	268,770	0.0153%	256	1,500
192	256	64	2,320	1,842	18,226	10.11%	96,059	268,950	0.0154%	256	1,501
193	256	128	805	298	33,066	0.90%	92,800	219,700	0.0072%	256	725

Continued on next page

Table A.2 – continued from previous page

n	h	m	m_0	$rows'$	$rows$	$m_0\%$	$cols$	nz	$nz\%$	row_s	col_s
194	256	128	821	309	33,077	0.93%	92,873	220,000	0.0072%	256	726
195	256	128	815	310	33,078	0.94%	92,819	220,320	0.0072%	256	725
196	256	128	812	653	33,421	1.95%	92,833	260,700	0.0084%	256	725
197	256	128	813	658	33,426	1.97%	92,853	260,530	0.0084%	256	725
198	256	128	810	657	33,425	1.97%	92,854	260,860	0.0084%	256	725
199	256	128	2,297	885	33,653	2.63%	192,080	456,330	0.0071%	256	1,501
200	256	128	2,388	928	33,696	2.75%	192,100	457,940	0.0071%	256	1,501
201	256	128	2,347	942	33,710	2.79%	192,070	460,780	0.0071%	256	1,501
202	256	128	2,342	1,873	34,641	5.41%	192,010	538,330	0.0081%	256	1,500
203	256	128	2,359	1,892	34,660	5.46%	192,040	538,580	0.0081%	256	1,500
204	256	128	2,358	1,870	34,638	5.40%	192,020	536,630	0.0081%	256	1,500
205	256	256	822	315	65,851	0.48%	185,000	439,940	0.0036%	256	723
206	256	256	825	316	65,852	0.48%	185,020	441,570	0.0036%	256	723
207	256	256	824	314	65,850	0.48%	185,020	440,750	0.0036%	256	723
208	256	256	836	673	66,209	1.02%	185,100	519,760	0.0042%	256	723
209	256	256	827	685	66,221	1.03%	185,200	523,080	0.0043%	256	723
210	256	256	827	682	66,218	1.03%	185,030	522,400	0.0043%	256	723
211	256	256	2,186	831	66,367	1.25%	360,200	856,740	0.0036%	256	1,407
212	256	256	2,174	880	66,416	1.33%	360,180	864,590	0.0036%	256	1,407
213	256	256	2,143	825	66,361	1.24%	360,050	857,050	0.0036%	256	1,406
214	256	256	2,188	1,769	67,305	2.63%	360,040	1,010,500	0.0042%	256	1,406
215	256	256	2,178	1,772	67,308	2.63%	360,050	1,011,900	0.0042%	256	1,406
216	256	256	2,204	1,802	67,338	2.68%	360,060	1,015,400	0.0042%	256	1,406

Appendix B

Some Source Codes

B.1 Calling CPLEX in MATLAB

```
/*************************************************************************
 * Program: cplexwarm_mex.c                                              *
 * Calling CPLEX in MATLAB with a warm start point                       *
 * Designed for CPLEX 10.1                                               *
 * Originally coded by David R. Musicant (musicant@cs.wisc.edu)          *
 * Modified by Jiarui Dang (jrdang@gmail.com)    *
 * This software is free for academic and research use only.             *
 *************************************************************************/

#include "/.software/share/cplex/cplex101/include/ilcplex/cplex.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "mex.h"

#define MINIMIZE 1
#define MANDATORY_ARGS 5
```

```
#define MAX_ITER_DEFAULT 10000

/* Input arguments */
#define C_IN   0
#define A_IN   1
#define B_IN   2
#define L_IN   3
#define U_IN   4
#define LE_IN  5
#define GE_IN  6
#define MI_IN  7
#define OP_IN  8
#define PRE_CSTAT_IN 9
#define PRE_RSTAT_IN 10

/* Output arguments */
#define OBJ_OUT    0
#define X_OUT      1
#define PI_OUT     2
#define STAT_OUT   3
#define CSTAT_OUT  4
#define ITER_OUT   5
#define RSTAT_OUT  6

void mexFunction (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
  /* MATLAB memory structures */
  const mxArray *c,*A,*b,*l,*u,*le,*ge,*maxIterPtr,*optPtr,*pre_cstat,*pre_rstat;

  /* Return arguments */
  double   *matlpstat,*objval,*x,*pi,*cstat,*itcnt,*rstat;
```

```
/* Other declarations */
char *sense,errorMsg[255];
int rows,cols,maxIter,*matbeg,*matcnt,*matind,opt;
double *c_ptr,*b_ptr,*matval,*l_ptr,*u_ptr,*slack,*dj,*pre_cstat_ptr,*pre_rstat_pt
int matrixSize,status,i,j,le_size,ge_size,m,n,netfind;
double *le_ptr = NULL,*ge_ptr = NULL;
int *istat,lpstat,*jstat,method;
CPXENVptr env;
CPXLPptr lp = NULL;

/* Assign pointers to MATLAB memory stuctures */
c = prhs[C_IN];
A = prhs[A_IN];
b = prhs[B_IN];
l = prhs[L_IN];
u = prhs[U_IN];
pre_cstat = prhs[PRE_CSTAT_IN];
pre_rstat = prhs[PRE_RSTAT_IN];

/* Get an mxArray's real data elements */
c_ptr = mxGetPr(c);
b_ptr = mxGetPr(b);
l_ptr = mxGetPr(l);
u_ptr = mxGetPr(u);
rows  = mxGetM(b);
cols  = mxGetM(c);
pre_cstat_ptr = mxGetPr(pre_cstat);
pre_rstat_ptr = mxGetPr(pre_rstat);
```

```
  int pre_c_stat[cols], pre_r_stat[rows];

  /* Build the matrix of coefficients, taking sparsity into account. */
  if (mxIsSparse(A)){
    /* Sparse */
    matbeg = mxGetJc(A);   /* beginnings of each column */
    matcnt = (int*)mxCalloc(cols,sizeof(int)); /* # of entries in each col */
    for (i = 0; i < cols; i++)
      matcnt[i] = matbeg[i+1] - matbeg[i];
    matind = mxGetIr(A);   /* row locations */
    matval = mxGetPr(A);   /* actual coefficients */

  } else {
    /* Dense */
    m = mxGetM(A);
    n = mxGetN(A);
    matbeg = (int*)mxCalloc(n,sizeof(int));
    matcnt = (int*)mxCalloc(n,sizeof(int));
    matind = (int*)mxCalloc(m*n,sizeof(int));
    matval = mxGetPr(A);
    for (j = 0; j < n; j++) {
      matbeg[j] = j*m;
      for (i = 0; i < m; i++)
matind[j*m + i] = i;
      matcnt[j] = m;
    }
  }

  /* Initialize all constraints to be equality constraints (default). */
  sense = (char*)mxCalloc(rows,sizeof(char));
  for(i = 0; i < rows; i++)
```

```
    sense[i] = 'E';

  /* If "<=" constraints given, set them up. */
  if(nrhs > MANDATORY_ARGS){
    le = prhs[LE_IN];
    le_ptr = mxGetPr(le);
    le_size = mxGetM(le);
    for(i = 0; i < le_size; i++)
      sense[(int)(le_ptr[i]-1)] = 'L';
  }

  /* If ">=" constraints given, set them up. */
  if(nrhs > MANDATORY_ARGS + 1){
    ge = prhs[GE_IN];
    ge_ptr = mxGetPr(ge);
    ge_size = mxGetM(ge);
    for(i = 0; i < ge_size; i++)
      sense[(int)(ge_ptr[i]-1)] = 'G';
  }

  /* Set up maximum number of iterations */
  if (nrhs > MANDATORY_ARGS + 2) {
    maxIterPtr = prhs[MI_IN];
    maxIter = (int)mxGetScalar(maxIterPtr);
  } else
    maxIter = MAX_ITER_DEFAULT;

/* Set up optimizer */
  if (nrhs > MANDATORY_ARGS + 3) {
    optPtr = prhs[OP_IN];
    opt = (int)mxGetScalar(optPtr);
```

```
} else
  opt = 0;

/* Output to MATLAB */
plhs[OBJ_OUT]    = mxCreateDoubleMatrix(1,1,mxREAL);
plhs[X_OUT]      = mxCreateDoubleMatrix(cols,1,mxREAL);
plhs[PI_OUT]     = mxCreateDoubleMatrix(rows,1,mxREAL);
plhs[STAT_OUT]   = mxCreateDoubleMatrix(1,1,mxREAL);
plhs[CSTAT_OUT]  = mxCreateDoubleMatrix(cols,1,mxREAL);
plhs[ITER_OUT]   = mxCreateDoubleMatrix(1,1,mxREAL);
plhs[RSTAT_OUT]   = mxCreateDoubleMatrix(rows,1,mxREAL);

objval    = mxGetPr(plhs[OBJ_OUT]);
x         = mxGetPr(plhs[X_OUT]);
pi        = mxGetPr(plhs[PI_OUT]);
matlpstat = mxGetPr(plhs[STAT_OUT]);
cstat     = mxGetPr(plhs[CSTAT_OUT]);
istat     = (int*)mxCalloc(cols,sizeof(int));
itcnt     = mxGetPr(plhs[ITER_OUT]);
rstat     = mxGetPr(plhs[RSTAT_OUT]);
jstat     = (int*)mxCalloc(cols,sizeof(int));

/* Open CPLEX environment */
env = CPXopenCPLEX(&status);
if (!env) {
  printf(CPXgeterrorstring(env,status,errorMsg));
  mexErrMsgTxt("\nCould not open CPLEX environment.");
}

/* Create CPLEX problem space */
lp = CPXcreateprob(env, &status, "matlab");
```

```
if (!lp) {
  printf(CPXgeterrorstring(env,status,errorMsg));
  CPXcloseCPLEX(&env);
  mexErrMsgTxt("\nCould not create CPLEX problem.");
}

/* Copy LP into CPLEX environment */
status = CPXcopylp(env, lp, cols, rows, MINIMIZE, c_ptr, b_ptr, sense,
   matbeg, matcnt, matind, matval, l_ptr, u_ptr, NULL);
if (status) {
  printf(CPXgeterrorstring(env,status,errorMsg));
  CPXfreeprob(env,&lp);
  CPXcloseCPLEX(&env);
  mexErrMsgTxt("\nCould not copy CPLEX problem.");
}

/* Set iteration limit. */
status = CPXsetintparam(env, CPX_PARAM_ITLIM, maxIter);
if (status) {
  printf(CPXgeterrorstring(env,status,errorMsg));
  CPXfreeprob(env,&lp);
  CPXcloseCPLEX(&env);
  mexErrMsgTxt("\nCould not set number of iterations.");
}

 /* Now copy the basis */
 status = CPXsetintparam (env, CPX_PARAM_ADVIND, CPX_ON);
 if ( status ) {
    fprintf (stderr,
             "Failure to turn on screen indicator, error %d.\n", status);
 CPXfreeprob(env,&lp);
```

```
   CPXcloseCPLEX(&env);
   }

  /* convert real number to iteger */
  for (i=0; i < cols; i++)
    pre_c_stat[i] = pre_cstat_ptr[i];
  for (i=0; i < rows; i++)
    pre_r_stat[i]= pre_rstat_ptr[i];

   status = CPXcopybase (env, lp, pre_c_stat, pre_r_stat);
   /* status = CPXcopystart (env, lp, pre_c_stat, pre_r_stat,
NULL, NULL, NULL, NULL);*/
   if ( status ) {
      fprintf (stderr, "Failed to copy the basis.\n");
      CPXfreeprob(env,&lp);
      CPXcloseCPLEX(&env);
   }

  /* Perform optimization  depending on the optimizer */
  netfind=0;
  if (opt==0) method = CPX_ALG_PRIMAL;
  if (opt==1) status = CPX_ALG_BARRIER;
  if (opt==2) status = CPXhybnetopt(env,lp,netfind);
  if (opt==3) status = CPXhybbaropt(env,lp,netfind);

   status = CPXsetintparam (env, CPX_PARAM_LPMETHOD, method);
/*   status = CPXsetintparam (env, CPX_PARAM_LPMETHOD, CPX_ALG_AUTOMATIC);
   status = CPXsetintparam (env, CPX_PARAM_LPMETHOD, CPX_ALG_BARRIER);
   status = CPXsetintparam (env, CPX_PARAM_LPMETHOD, CPX_ALG_DUAL); */
   if ( status ) {
      fprintf (stderr,
```

```
            "Failed to set the optimization method, error %d.\n", status);
 }

status = CPXlpopt(env,lp);
if (status) {
  printf(CPXgeterrorstring(env,status,errorMsg));
  CPXfreeprob(env,&lp);
  CPXcloseCPLEX(&env);
  mexErrMsgTxt("\nOptimization error");
}

/* Obtain solution */
status = CPXsolution(env, lp, &lpstat, objval, x, pi, NULL, NULL);
*matlpstat = lpstat;
if (status) {
  printf(CPXgeterrorstring(env,status,errorMsg));
  CPXfreeprob(env,&lp);
  CPXcloseCPLEX(&env);
  mexErrMsgTxt("\nFailure when retrieving solution.");
}

/* Get status of columns */
status = CPXgetbase(env, lp, istat, jstat);
if (status) {
  printf(CPXgeterrorstring(env,status,errorMsg));
  CPXfreeprob(env,&lp);
  CPXcloseCPLEX(&env);
  mexErrMsgTxt("\nUnable to get basis status.");
}

/* Copy int column values to double column values */
```

```
  for (i=0; i < cols; i++)
    cstat[i] = istat[i];
  for (i=0; i<rows; i++)
    rstat[i]=jstat[i];

  /* Get iteration count */
  *itcnt = (double)CPXgetitcnt(env,lp);
  /* printf ("Iteration count = %d\n\n", CPXgetitcnt (env, lp));
  printf("check index: %d, %d \n", CPXgetnumcols(env,lp), CPXgetnumrows(env,lp)); */

  /* Clean up problem  */
  CPXfreeprob(env,&lp);
  CPXcloseCPLEX(&env);

}
```

B.2 Generating Pseudo Blocks (MPS) by GAMS

```
* Stochastic Portfolio Management(Dual block-angular)
* User needs to set the number of assets and senarios
* Expected returns are generted with random numbersf [0.8,1.2]
* Generate series of pseudo blocks in MPS format

OPTION RESLIM = 10000;
OPTION ITERLIM = 25000;
OPTION LIMROW = 0;
OPTION LIMCOL = 0;
OPTION SOLPRINT = OFF;
```

```
SETS

S Scenarios
/
$offlisting
$include s.csv
$onlisting
/

A Assets
/
$offlisting
$include a.csv
$onlisting
/;

ALIAS (S,O);

SCALARS
RC Cash rate of return /1.05/
WI Initial capital /50/
WF Goal /55/;

TABLE P(S,A) Asset rates of return
$offlisting
$ondelim
$include p.csv
$offdelim
$onlisting

PARAMETERS
```

```
P_s(A);

VARIABLES
EU

POSITIVE VARIABLES
C Cash Period 0
X(A) Asset Period 0
C1_s Cash Period 1
Y_s(A) Asset Period 1
U_s(O) Surplus Period 2
V_s(O) Deficit Period 2;

EQUATIONS
OBJECTIVE Calculating the expectation of the utility function
G Balance of Financial Flows Period 0
H_s Balance of Financial Flows Period 1
L_s(O) Balance of Financial Flows Period 2;

OBJECTIVE.. EU=E=SUM(O,5*U_s(O)-20*V_s(O));
G.. C+SUM(A,X(A)) =L= WI;
H_s.. -RC*C-SUM(A,X(A)*P_s(A)) =L= -C1_s-SUM(A,Y_s(A));
L_s(O).. -RC*C1_s-SUM(A,Y_s(A)*P(O,A)) =L= -WF-U_s(O)+V_s(O);

MODEL PORT /ALL/;

file mpsopt /convert.opt/;

option lp=convert;
PORT.optfile=1;
```

```
loop(S,
    P_s(A) = P(S,A);

* Generates an mps file for each subproblem called 'sub*.mps'
     put mpsopt "cplexmps port" card(S):0:0 "_sub" ord(S):0:0 ".mps";
     putclose mpsopt;
     Solve PORT using lp maximizing EU;

);
```

B.3 Converting Free MPS Format to Standard

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Converting free mps format files to standard mps    %
% MATLAB .m code    %
% Input: cplexmps.mps (free MPS format)    %
% Output: stand.mps (industrial standard MPS format)    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fid=fopen('cplexmps.mps', 'rt');

rows={}; columns={}; rhs={}; bounds={};
numrows=0; numcols=0; numrhs=0; numbds=0;
tag=0;

while 1
    tline = fgetl(fid);
    if ~ischar(tline), break, end
```

```
[first, rest] = strtok(tline);

switch first            %indicator lines
    case 'NAME'
        filename=strtrim(rest);
    case 'ROWS'
        tag=1;
        disp(first);
    case 'COLUMNS'
        tag=2;
        disp(first);
    case 'RHS'
        tag=3;
        disp(first);
    case 'BOUNDS'
        tag=4;
        disp(first);
    case 'ENDATA'
        break;
    otherwise
        switch tag      %data lines
            case 1      %rows
                numrows=numrows+1;
                rows{end+1}=strtrim(first);
                rows{end+1}=strtrim(rest);
            case 2      %columns
                [two, three] = strtok(rest);
                numcols=numcols+1;
                columns{end+1}=strtrim(first);
                columns{end+1}=strtrim(two);
```

```
                    columns{end+1}=strtrim(three);
                case 3      %rhs
                    [two, three] = strtok(rest);
                    numrhs=numrhs+1;
                    rhs{end+1}=strtrim(first);
                    rhs{end+1}=strtrim(two);
                    rhs{end+1}=strtrim(three);
                case 4      %bounds
                    [two, three] = strtok(rest);
                    [three, four]=strtok(three);
                    numbds=numbds+1;
                    bounds{end+1}=strtrim(first);
                    bounds{end+1}=strtrim(two);
                    bounds{end+1}=strtrim(three);
                    bounds{end+1}=strtrim(four);
                otherwise
            end
    end
end
fclose(fid);

fid=fopen('stand.mps', 'wt');        %industrial standard MPS format
fprintf(fid, 'NAME          ');
fprintf(fid, '%s\n', filename);
fprintf(fid, 'ROWS\n');

for i=1:numrows           %data of rows
    fprintf(fid, ' %s  %s\n',rows{2*i-1},rows{2*i});
end

fprintf(fid, 'COLUMNS\n');      %data of columns
```

```
for i=1:numcols
    fprintf(fid, '    %s',columns{3*(i-1)+1});
    for j =1:(10-length(columns{3*(i-1)+1}));
        fprintf(fid, ' ');
    end
    fprintf(fid,columns{3*(i-1)+2});
    for j=1:(10-length(columns{3*(i-1)+2}))
        fprintf(fid, ' ');
    end
    fprintf(fid, '%s\n', columns{3*i});
end

fprintf(fid, 'RHS\n');          %data of rhs
for i=1:numrhs
    fprintf(fid, '    %s',rhs{3*(i-1)+1});
    for j =1:(10-length(rhs{3*(i-1)+1}));
        fprintf(fid, ' ');
    end
    fprintf(fid,rhs{3*(i-1)+2});
    for j=1:(10-length(rhs{3*(i-1)+2}))
        fprintf(fid, ' ');
    end
    fprintf(fid, '%s\n', rhs{3*i});
end

fprintf(fid, 'BOUNDS\n');          %data of bounds
for i=1:numbds
    fprintf(fid, ' %s',bounds{4*(i-1)+1});
    fprintf(fid, ' %s',bounds{4*(i-1)+2});
    for j =1:(10-length(bounds{4*(i-1)+2}));
        fprintf(fid, ' ');
```

```
    end
    fprintf(fid,bounds{4*(i-1)+3});
    for j=1:(10-length(bounds{4*(i-1)+3}))
        fprintf(fid, ' ');
    end
    fprintf(fid, '%s\n', bounds{4*i});
end

fprintf(fid, 'ENDATA\n');

fclose(fid);
```

Appendix C

Weights in Robust Regression

C.1 Robust Regression: DW vs. Simplex

C.2 Robust Regression: ACCPM vs. IPM

Table C.1: Weights in Random Prediction on $R_{DW/Simplex}$ (Financial)

col. 1-20	21-40	41-60	61-80	81-85
0.88	0.9986	0.9997	0.9911	0.9541
0.9997	0.9469	0.8933	0.9867	0.9832
0.999	0.9805	0.9957	0.991	0.6456
0.9963	0.5695	0.9344	0.9788	0.9936
0.9735	0.9748	0.9755	0.9133	0.5487
0.9566	0.9832	0.9726	0.7323	
0.9754	0.9868	0.9567	0.9885	
0.9863	0.9961	0.6896	0.8523	
0.9853	0.8123	0.9992	0.741	
0.9801	0.9727	0.9997	0.7802	
0.9907	0.9977	1	0.6698	
0.9968	0.965	0.9138	0.9696	
0.9077	0.9931	0.9301	0.9139	
0.8632	0.7476	0.8497	0.9598	
0.8846	0.8552	0.9746	0.8483	
0.9299	0.8632	0.877	0.9758	
0.916	0.9186	0.993	0.8432	
0.7778	0.9683	0.9972	0.8363	
0.9807	0.9646	0.8559	0.9641	
0.8186	0.9523	0.915	0.9654	

Table C.2: Weights in Random Prediction on $R_{DW/Simplex}$ (Multicommodity)

col. 1-30	31-60	61-90	91-120	121-150	151-180	181-206
0.7997	0.9997	0.9883	0.9472	1	0.8355	0.9017
0.7608	0.9936	0.9817	0.9726	0.9895	0.9988	0.9179
0.8231	0.8708	0.8352	0.9997	0.9566	0.9968	0.8742
0.7741	0.8306	0.9691	0.9993	0.9925	0.9988	0.9348
0.8303	0.975	0.9959	1	0.9999	0.9469	0.9778
0.8758	0.8722	0.9566	0.9997	0.999	0.9452	0.8416
0.9995	0.7963	0.9826	0.9508	0.9966	0.8993	0.939
0.9945	0.8161	0.9192	0.9964	0.992	0.8842	0.9944
0.9281	0.9741	0.8797	0.9875	0.8147	0.9093	0.929
0.9684	0.9783	0.9932	0.999	0.8401	0.7204	0.899
0.8971	0.9377	0.9828	1	0.9269	0.8136	0.8321
0.9258	0.9536	0.9932	0.9916	0.993	0.9137	0.9937
0.8267	0.9578	0.998	0.9952	0.9612	0.9334	0.9188
0.9998	0.7572	0.9958	0.999	0.8975	0.9738	0.9404
0.905	0.8367	0.9954	0.987	0.8249	1	0.946
0.91	0.7938	0.8689	0.9684	0.9492	0.9985	0.9836
0.9055	0.9411	0.9183	0.9358	0.8752	0.981	0.952
0.9428	0.9218	0.8687	0.9957	0.9597	0.9192	0.9418
0.9914	0.9875	0.9273	0.9967	0.916	0.8512	0.9954
0.9874	0.9971	0.8577	0.9975	0.9226	0.949	0.9726
0.9945	0.9934	0.9996	0.997	0.9968	0.9714	0.9815
0.9502	0.9982	0.9801	0.9978	0.9996	0.9732	0.9357
0.9413	0.9983	0.9865	0.9806	0.9398	0.99	0.879
0.986	0.9988	0.9307	0.9757	0.968	0.9391	0.9908
0.6637	0.957	0.975	0.9973	0.9781	0.974	0.9809
0.9237	0.9341	0.9985	0.9829	0.8331	0.9829	0.9715
0.7704	0.8245	0.7887	0.9739	0.7569	0.8534	
0.9171	0.9738	0.798	0.9652	0.951	0.9082	
0.8859	0.9608	0.8564	0.9999	0.5594	0.8594	
0.9253	0.948	0.9527	0.9911	0.8257	0.9925	

Table C.3: Weights in Extrapolation on $R_{DW/Simplex}$ (Financial)

col. 1-20	21-40	41-60	61-80	81-85
0.9277	0.9434	0.8639	0.9622	0.9835
0.9999	0.9728	0.9999	0.9994	0.9811
0.9978	0.6596	0.9013	0.9774	0.9608
0.998	0.9617	0.9491	0.6849	0.7649
0.9793	0.9702	0.9959	0.9476	0.9996
0.9584	0.9738	0.9648	0.8165	
0.9761	0.9867	0.6986	0.9693	
0.9864	0.9522	0.9995	0.9212	
0.9856	0.998	0.9939	0.8433	
0.9798	0.9781	0.9942	0.8451	
0.9978	0.9987	0.8777	0.9671	
0.9127	0.8198	0.8925	0.9362	
0.8661	0.9094	0.8013	0.9452	
0.9136	0.918	0.9579	0.9074	
0.9517	0.9591	0.8863	0.9952	
0.9965	0.9715	0.989	0.7955	
0.9063	0.9714	1	0.7819	
0.8296	0.9781	0.9125	0.877	
0.881	0.9748	0.8816	0.9904	
0.9988	0.9947	0.9714	0.9584	

Table C.4: Weights in Extrapolation on $R_{DW/Simplex}$ (Multicommodity)

col. 1-30	31-60	61-90	91-120	121-150	151-180	181-206
0.8184	0.9996	0.9636	0.7963	0.987	0.915	0.9958
0.7793	0.9969	0.951	0.8056	0.9788	0.95	0.9527
0.8415	0.9926	0.9846	0.8643	0.9708	0.963	0.9835
0.7966	0.8576	0.9852	0.9575	1	0.795	0.9961
0.8525	0.8143	0.9972	0.9521	0.9933	0.7104	0.9763
0.8958	0.9708	0.8114	0.9767	0.9999	0.9311	0.9721
0.9999	0.8765	0.9709	0.9999	0.9778	0.5629	0.7971
0.9922	0.7982	0.9969	0.9988	0.9615	0.8345	0.863
0.9166	0.8183	0.9584	0.9999	0.9335	0.8444	0.8043
0.9598	0.9792	0.983	0.9997	0.9835	0.9969	0.9792
0.8798	0.9697	0.9175	0.9489	0.999	0.994	0.8584
0.9116	0.9832	0.8765	0.9959	1	0.9997	0.8782
0.8402	0.9356	0.9938	0.9927	0.9977	0.9199	0.8964
0.9997	0.9521	0.9835	1	0.9935	0.918	0.9522
0.918	0.9566	0.9938	0.9988	0.8132	0.8623	0.9882
0.9244	0.742	0.9968	0.9952	0.8264	0.8478	0.8611
0.9204	0.8262	0.9939	0.9978	0.9194	0.8774	0.9533
0.955	0.7808	0.9965	0.9967	0.9935	0.6639	0.9863
0.9922	0.9427	0.8729	0.9894	0.9716	0.8274	0.8841
0.9849	0.9225	0.9228	0.9714	0.9125	0.9258	0.7655
0.9931	0.9891	0.9122	0.9391	0.8415	0.9444	0.9645
0.9416	0.9984	0.8693	0.9965	0.9322	0.9801	0.9936
0.9318	0.9952	0.9292	0.9974	0.8472	0.9991	0.9696
0.9822	0.9965	0.858	0.9964	0.9447	0.9957	0.9585
0.6675	0.9989	0.9995	0.9996	0.9208	0.9621	0.9994
0.9316	0.9994	0.9789	0.9928	0.9274	0.8815	0.9532
0.7778	0.9589	0.9859	0.9958	0.8939	0.8005	
0.9276	0.9326	0.9242	0.9874	0.995	0.9215	
0.897	0.8183	0.972	0.9832	0.9999	0.9506	
0.9361	0.974	0.9988	0.9995	0.8875	0.953	

Table C.5: Weights in Random Prediction on $R_{ACCPM/IPM}$ (Financial)

col. 1-20	21-40	41-60	61-80	81-90
0	0.9855	0.9715	0.6246	0.9998
0.5637	1	0.8408	0.9943	0.9962
0.7234	0.9908	0.9938	0.9896	0.9987
0.9889	0.9092	0.7785	0.9646	0.9282
0.9716	0.9942	0.9328	0.9963	0.9725
0.9388	0.8158	0.8644	0.9238	0.9846
0.9998	0.9344	0	0.7024	0.9504
0.9947	0.9326	0	0	0.9264
0.6577	0.7182	0.982	0.948	0.9641
0.4027	0.9901	0.9158	0.9983	0.9865
0.809	0.9465	0.9088	0.9587	
0.9994	0.9828	0.9943	0.9975	
0.9472	0.8869	0.9375	0.9795	
0.9758	0.9986	0.9899	0.7248	
0.6848	1	0.5906	0	
0.9984	0.8425	0.7048	0.9827	
0.9253	0.8319	0.601	0.9599	
0.9727	0.8382	0.9952	0.8261	
0.9933	0.9825	0.904	0.9335	
0.9927	0.899	0.9991	0.9897	

Table C.6: Weights in Random Prediction on $R_{ACCPM/IPM}$ (Multicommodity)

col. 1-20	21-40	41-60	61-80	81-100	101-120	121-122
0.9735	0.8956	0.9818	0.8637	0.9322	0.8182	0.9905
0.972	0.901	0.9522	1	0.9489	0.9894	0.9997
0.9349	0.9572	0.9946	0.7646	0.9919	0.9826	
0.9578	0.9351	1	0.7075	0.9983	0.9529	
0.9709	0.984	0.8263	0.0712	0.8183	0.9917	
0.935	0.9908	0.8822	0.1386	0.7992	0.9993	
0.9901	0.9906	0.8925	0.3909	0.6693	0.9978	
0.9914	0.952	0.7808	1	0.9885	1	
0.9995	0.9682	0.785	1	0.9991	0.8793	
0.903	0.9254	0.7961	0.9458	0.9982	0.9295	
0.9743	0.8986	0.8966	0.8913	0.989	0.8782	
0.968	0.8719	0.9306	0.8862	0.9996	0.7647	
0.9621	0.967	0.9993	0.9671	0.9949	0.8434	
0.9044	0.9976	0.8967	0.9792	0.9845	0.996	
0.8955	0.9578	0.9319	0.9727	0.8847	0.999	
0.9989	0.9911	0.963	0.9999	0.8949	0.9723	
0.999	0.9902	0.9726	0.4254	0.8889	0.838	
0.9906	0.9996	0.9683	0.4619	0.9125	0.7117	
0.9835	0.9578	0.9609	0.607	0.8088	0.7168	
0.9377	0.9998	0.6075	0.9775	0.8424	0.9952	

Table C.7: Weights in Extrapolation on $R_{ACCPM/IPM}$ (Financial)

col. 1-20	21-40	41-60	61-80	81-90
0	1	0.9746	0.5458	0.9915
0.6576	1	0.8847	0.999	0.9994
0.7523	0.966	0.9731	0.9708	0.8816
0.9855	0.9991	0.8481	0.9849	0
0.985	0.9903	0.9991	0.9993	0.9975
0.9887	0.9161	0.6866	0.8315	0.9556
0.9754	0.9818	0.8709	0	0.9896
0.9849	0.8547	0	0.9469	0.9559
0.9924	0.9696	0	0.9602	0.8971
0.4215	0.719	0.9794	1	0.8558
0.5148	0.9743	0.9121	0.9753	
0.8761	0.9198	0.9128	0.9997	
0.998	0.9759	0.9985	0.9454	
0.9269	0.8784	0.9563	0.8204	
0.9754	0.9999	0.9989	0	
0.6898	0.9977	0.4504	0.9473	
0.9178	0.7645	0.4542	0.9782	
0.9975	0.7338	0.988	0.8512	
0.9696	0.9008	0.8782	0.9547	
0.9987	0.8012	0.9966	0.9689	

Table C.8: Weights in Extrapolation on $R_{ACCPM/IPM}$ (Multicommodity)

col. 1-20	21-40	41-60	61-80	81-100	101-120	121-122
0.9661	0.8831	0.9996	0.9179	1	0.9825	0.9995
0.9647	0.9128	0.9794	0.9735	0.4357	0.8764	0.9679
0.9226	0.9655	0.9504	0.9522	0.4732	0.8901	
0.9665	0.9449	0.9943	0.6239	0.6202	0.8842	
0.9789	0.9837	1	0.8783	0.976	0.9084	
0.9174	0.9878	0.8112	0.9991	0.9284	0.79	
0.9934	0.9902	0.871	0.7764	0.9451	0.8256	
0.9868	0.9924	0.8818	0.7198	0.9815	0.8003	
0.998	0.9455	0.7636	0.8932	0.989	0.9861	
0.9187	0.9621	0.7665	0.0788	0.9969	0.9774	
0.9827	0.9189	0.7801	0.1499	0.805	0.9894	
0.9774	0.8909	0.8913	0.4099	0.7845	0.9822	
0.9566	0.8621	0.9265	0.9999	0.6482	0.9516	
0.8959	0.9591	0.9991	0.9998	0.9836	0.9917	
0.8828	0.9953	0.8818	0.9389	0.9975	0.9984	
0.9998	0.9488	0.887	0.9	0.9961	1	
0.9971	0.9898	0.9242	0.8953	0.9872	0.7376	
0.986	0.9883	0.9915	0.9728	0.998	0.7531	
0.9867	0.9992	0.9627	0.9827	0.9998	0.821	
0.9278	0.9581	0.9722	0.9766	0.9938	0.9946	

Appendix D

Correlation Among Dimensional Parameters

D.1 Correlation Matrix for Financial Model

D.2 Correlation Matrix for Multicommodity Model

Table D.1: Correlation Matrix for 100 Financial Problems

corr.	h	m_0	$\overline{m}$	$m_0\%$	$\overline{n}$	$nz\%$	m_{sub}	n_{sub}
h	1	0	0.96	-0.7	0.97	-0.7	0.89	0.95
m_0	0	1	0.19	0.19	0.14	0.17	0.45	0.32
$\overline{m}$	0.96	0.19	1	-0.6	1	-0.6	0.95	0.97
$m_0\%$	-0.7	0.19	-0.6	1	-0.6	1	-0.6	-0.6
$\overline{n}$	0.97	0.14	1	-0.6	1	-0.6	0.93	0.96
$nz\%$	-0.7	0.17	-0.6	1	-0.6	1	-0.6	-0.6
m_{sub}	0.89	0.45	0.95	-0.6	0.93	-0.6	1	0.99
n_{sub}	0.95	0.32	0.97	-0.6	0.96	-0.6	0.99	1

Table D.2: Correlation Matrix for 216 Multicommodity Problems

corr.	h	m_0	$\overline{m}$	$m_0\%$	$\overline{n}$	$nz\%$	m_{sub}	n_{sub}
h	1	0.2	0.96	-0.6	0.89	-0.5	0.32	0.23
m_0	0.2	1	0.32	0.18	0.43	-0.4	0.61	0.86
$\overline{m}$	0.96	0.32	1	-0.5	0.94	-0.4	0.48	0.38
$m_0\%$	-0.6	0.18	-0.5	1	-0.4	0.56	-0.1	0.1
$\overline{n}$	0.89	0.43	0.94	-0.4	1	-0.4	0.44	0.48
$nz\%$	-0.5	-0.4	-0.4	0.56	-0.4	1	-0.5	-0.5
m_{sub}	0.32	0.61	0.48	-0.1	0.44	-0.5	1	0.79
n_{sub}	0.23	0.86	0.38	0.1	0.48	-0.5	0.79	1

Bibliography

[1] I. Alder and A. Ulkucu. On the number of iterations in Dantzig-Wolfe decomposition. In D. M. Himmelblau, editor, *Decomposition of large scale problems*, pages 181–187, North-Holland, Amsterdam, 1973.

[2] A. Ali and J. L. Kennington. Mnetgen program documentation. Technical report 77003, Department of Industrial Engeneering and Operations Research, Southern Methodist University, Dallas, 1977.

[3] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms for linear programming. *Management Sci.*, 42(12):1719–1731, 1996.

[4] F. Babonneau, O. D. Merle, and J.-P. Vial. Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-accpm. *Oper. Res.*, 54(1):184–197, 2006.

[5] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, Belmont, MA, 1997.

[6] R. E. Bixby. Implementing the simplex method: the initial basis. *ORSA J. Comput.*, 4:267–284, 1992.

[7] R. E. Bixby. Solving real-world linear programs: a decade and more of progress. *Oper. Res.*, 50(1):3–15, 2002.

[8] R. E. Bixby, J. W. Gregory, I. J. Lustig, R. E. Marsten, and D. F. Shanno. Very large-scale linear programming: a case study in combining interior point and simplex methods. *Oper. Res.*, 40(5):885–897, 1992.

[9] R. E. Bixby and M. J. Saltzman. Recovering an optimal basis from an interior point solution. *Oper. Res.*, 15(4):169–178, 1993.

[10] R. G. Bland. New finite pivoting rules for the simplex method. *Math. Oper. Res.*, 2:103–107, 1977.

[11] B. Borchers and J. E. Mitchell. *Using an interior point method in a branch-and-bound algorithm for integer programming.* Rensselaer Polytechnic Institute, Troy, N. Y., 1992.

[12] K. H. Borgwardt. *The Simplex Method:a probabilistic analysis.* Springer-Verlag, 1980.

[13] S. P. Bradley, A. C. Hax, and T. L. Magnanti. *Applied Mathematical Programming.* Addison-Wesley Publishing Company, 1977.

[14] P. Cappanera and A. Frangioni. Symmetric parallelization of a cost-decomposition algorithm for multicommodity flow problems. *INFORMS J. Comput.*, 15(4):369–384, 2003.

[15] E. W. Cheney and A. A. Goldstein. Newton's method for convex programming and Tchebycheff approximation. *Numeriche Mathematik*, 1:253–268, 1959.

[16] V. Chvatal. *Linear Programming.* W.H. Freeman and Company, New York/San Francisco, 1983.

[17] D. K. D. and J. M. Wilson. Developments in linear and integer programming. *JSTOR: Journal of the Operational Research Society*, 53(10):1065–2072, 2002.

[18] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity analysis of production and allocation*, pages 488–499, Wiley, New York, 1951.

[19] G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific J. Math.*, 5:183–195, 1955.

[20] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Oper. Res.*, 8:101–111, 1960.

[21] G. B. Danzig. *Linear programming and extensions.* Princeton University Press, Princeton, New Jersey, 1963.

[22] G. B. Danzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29:767–778, 1961.

[23] S. Elhedhli. *Interior-point decomposition methods for integer programming: theory and application.* PhD thesis, Faculty of Management, McGill university, 2001.

[24] S. Elhedhli and J.-L. Goffin. The integration of an interior-point cutting plane method within a branch-and-price algorithm. *Math. Program.*, 100(A):267–294, 2004.

[25] S. Elhedhli, J.-L. Goffin, and J.-P. Vial. Nondifferentiable optimization: Cutting plane methods. In C. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, Kluwer Academic Publishers, 2001.

[26] S. Elhedhli, J.-L. Goffin, and J.-P. Vial. Nondifferentiable optimization: Introduction, applications and algorithms. In C. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, Kluwer Academic Publishers, 2001.

[27] A. V. Fiacco and G. P. McCormick. *Nonlinear programming: sequential unconstrained minimization techniques.* John Wiley and Sons, New York, 1968.

[28] R. Fourer. 2003 software survey: linear programming. *OR/MS Today*, December 2003:34–43, 2003.

[29] E. Fragniere, J. Gondzio, and J.-P. Vial. Building and solving large-scale stochastic programs on an affordable distributed computing system. *Ann. Oper. Res.*, 99:167–187, 2000.

[30] A. Frangioni and G. Gallo. A bundle type dual-ascent approach to linear multicommodity min cost flow problems. *INFORMS J. Comput.*, 11(4):370–393, 1999.

[31] K. R. Frisch. The logarithmic potential method of convex programming. Technical report, University Institute of Economics, Oslo, Norway, 1955.

[32] K. Fukuda and T. Terlaky. On the existence of short admissible pivot sequences. *PUMA: Mathematics of Optimization*, 10(4):431–488, 2000.

[33] GAMS. Mpswrite, 2000. http://www.gams.com/solvers/mpswrite.pdf.

[34] J.-L. Goffin, A. Haurie, and J.-P. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Sci.*, 38(2):284–302, 1992.

[35] J.-L. Goffin, A. Haurie, J.-P. Vial, and D. L. Zhu. Using central prices in the decomposition of linear programs. *European J. Oper. Res.*, 64:393–409, 1993.

[36] J.-L. Goffin and J.-P. Vial. On the computation of weighted analytic centers and dual ellipsoids with the projective algorithm. *Math. Program.*, 60:81–92, 1993.

[37] J.-L. Goffin and J.-P. Vial. Shallow, deep and very deep cuts in the analytic center cutting plane method. Technical report, Logilab technical report, University of Geneva, Switzerland, 1996.

[38] J.-L. Goffin and J.-P. Vial. Multiple cuts in the analytic center cutting plane method. *SIAM J. Optim*, 11(1):805–867, 2000.

[39] J. Gondzio. Warm start of the primal-dual method applied in the cutting-plane scheme. *Math. Program.*, 83:125–143, 1998.

[40] J. Gondzio, O. d. Merle, R. Sarkissian, and J.-P. Vial. Accpm a library for convex optimization based on an analytic center cutting plane method. Technical report, Logilab, 1996.

[41] J. Gondzio, R. Sarkissian, and J.-P. Vial. Using an interior point method for the master problem in a decomposition approach. *European J. Oper. Res.*, 101:577–587, 1997.

[42] H. J. Greenberg. Computational tesing: why, how and how much. *ORSA J. Comput.*, 2(1):94–97, 1990.

[43] J. K. Ho, T. C. Lee, and R. P. Sundarraj. Decomposition of linear programs using parallel computation. *Math. Program.*, 42:391–405, 1988.

[44] J. K. Ho and E. Loute. An advanced implementation of the Dantzig-Wolfe decomposition algorithm for linear programming. *Math. Program.*, 20(1):303–326, 1981.

[45] J. K. Ho and E. Loute. Computational experience with advanced implementation of decomposition algorithms for linear programming. *Math. Program.*, 27:283–290, 1988.

[46] J. K. Ho and R. P. Sundarraj. *DECOMP: an implementation of Dantzig-Wolfe decomposition for linear programming.* Spring-Verlag New York Berlin Heidelberg, 1989.

[47] D. C. Hoaglin, V. Klema, and S. C. Peters. Exploratory data analysis in a study of the performance of nonlinear optimization routines. *ACM Trans. Math. Software*, 8(2):145–162, 1982.

[48] A. J. Hoffman, M. Mannos, D. Sokolowsky, and N. Wiegmann. Computational experience in solving linear programs. *J. SIAM*, 1:17–33, 1953.

[49] J. N. Hooker. Needed: an empirical science of algorithms. *Oper. Res.*, 42(2):201–212, 1994.

[50] T. Illes and T. Terlaky. Pivot versus interior point methods: Pros and cons. *European J. Oper. Res.*, 140:170–190, 2002.

[51] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[52] J. E. Kelley Jr. The cutting plane method for solving convex programs. *J. SIAM*, 8:703–712, 1960.

[53] L. G. Khachiyan. A polynomial algorithm in linear programming (russian). *Doklady Akademii Nauk SSSR*, 244(5):1093–1096, 1979.

[54] V. Klee and G. J. Minty. How good is the simplex algorithm. In O. Shisha, editor, *Inequalities-III*, New York, 1972. Academic Press.

[55] S. Kontogiorgis, R. D. Leone, and R. R. Meyer. Alternating directions splitting for block angular parallel optimization. *J. Optim. Theory Appl.*, 90(1):1–29, 1996.

[56] T. C. Koopmans, editor. *Activity analysis of production and allocation.* Wiley, New York, 1951.

[57] T. Larsson and D. Yuan. An augmented Lagrangian algorithm for large scale multicommodity routing. *Comput. Optim. Appl.*, 27(2):187–215, 2004.

[58] R. Levkotitz and G. Mitra. Experimental investigations in combining primal dual interior point method and simplex based LP solvers. *Ann. Oper. Res.*, 58:19–38, 1995.

[59] H. Luh and R. Tsaih. An efficient search direction for linear programming problems. *Comput. Oper. Res.*, 29:195–203, 2002.

[60] I. J. Lustig, R. E. Marsten, and D. F. Shanno. On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM J. Optim.*, 2(3):435–449, 1992.

[61] P. Mahey. Decomposition methods for mathematical programming. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of applied optimization*, pages 337–351, New York, 2002. Oxford Univeristy Press, Inc.

[62] R. Marstern, R. Subramanian, M. Saltzman, I. Lustig, and D. Shanno. Interior point methods for linear programming: Just call Newton, Lagrange, and Fiacco and Mccormick. *Interfaces*, 20(4):105–116, 1990.

[63] R. K. Martinson and J. Tind. An interior point method in Dantzig-Wolfe decomposition. *Comput. Oper. Res.*, 26:1195–1216, 1999.

[64] The MathWorks. *MATLAB documentation.* http://www.mathworks.com/access/helpdesk/help/techdoc/.

[65] C. C. McGeoch. Analyzing algorithms by simulation: variance reduction techniques and simulation speedups. *ACM Comput. Surveys*, 24(2):195–212, 1992.

[66] C. C. McGeoch. Toward an experimental method for algorithm simulation. *INFORMS J. Comput.*, 8(1):1–15, 1996.

[67] C. C. McGeoch. Experimental analysis of algorithms. *Notice of the AMS*, 48(3):304–311, 2001.

[68] C. C. McGeoch. Experimental analysis of optimization algorithms. In P. M. Pardalos and M. G. C. Resende, editors, *Handbook of applied optimization*, pages 1044–1052, New York, 2002. Oxford Univeristy Press, Inc.

[69] N. Megiddo. On finding primal- and dual-optimal bases. *ORSA J. Comput.*, 3:63–65, 1991.

[70] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optim*, 2:575–601, 1992.

[71] O. D. Merle, J.-L. Goffin, and J.-P. Vial. On improvements to the analytic center cutting plane method. *Comput. Optim. Appl.*, 11:37–52, 1998.

[72] C. B. Moler. *Numerical computing with MATLAB.* Society for Industrial and Applied Mathematics, Philadelphia, 2004.

[73] D. R. Musicant. Matlab/cplex mex-files, 2000. www.cs.wisc.edu/∼musicant/data/cplex/.

[74] G. L. Nemhauser. The age of optimization: solving large-scale real-world problems. *Oper. Res.*, 42(1):5–13, 1994.

[75] W. Orchard-Hays. *Advanced linear-programming computing techniques.* McGraw-Hill, New York, 1968.

[76] G. Schultz and R. Meyer. An interior-point method for block-angular optimization. *SIAM J. Optim.*, 1(4):583–682, 1991.

[77] R. Sedgewick. *Algorithms.* Addison-Wesley, 1988.

[78] J. N. Singh and D. Singh. Interior-point methods for linear programming: a review. *Internat. J. Math. Ed. Sci. Tech.*, 33(3):405–423, 2002.

[79] G. Sonnevend. An analytical center for polyhedrons and new classes of global algorithms for linear smooth, convex programming. In *Lecture Notes in Control and Information Sciences 84*, pages 866–876, New York, 1985. Springer.

[80] G. Sonnevend. Applications of the notion of analytic center in approximation estimation problem. *J. Comput. Appl. Math.*, 28:349–358, 1989.

[81] M. J. Todd. A Dantzig-Wolfe-like variant of Karmarkar's interior-point linear programming algorithm. *Oper. Res.*, 38(6):1006–1018, 1990.

[82] M. J. Todd. The many facets of linear programming. *Math. Program. Ser. B*, 91:417–436, 2002.

[83] R. J. Vanderbei. *Linear programming: foundations and extensions.* Kluwer's international series. Kluwer Academic Publishers, Boston/London/dordrecht, 1998.

[84] W. L. Winston. *Operations research: applications and algorithms.* PWS-Kent Pub. Co, Boston, 1991.

[85] Y. Ye. Complexity analysis of the analytic center cutting plane method that uses multiple cuts. *Math. Program.*, 78:85–104, 1997.

[86] Y. Zhang. Solving large-scale linear programs by interior-point methods under the MATLAB environment. Technical report, Department of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, MD, 1995.

[87] G. Zhao. Interior-point methods with decomposition for solving large-scale linear programs. *J. Optim. Theory Appl.*, 102(1):169–192, 1999.