COMBINED OBJECTIVE LEAST SQUARES AND LONG STEP PRIMAL DUAL SUBPROBLEM SIMPLEX METHODS

A Thesis
Presented to
The Academic Faculty

by

Sheng Xu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
May 2016

COMBINED OBJECTIVE LEAST SQUARES AND LONG STEP PRIMAL DUAL SUBPROBLEM SIMPLEX METHODS

Approved by:

Professor Ellis Johnson,
Committee Chair
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Ellis Johnson, Advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Earl Barnes
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Martin Savelsbergh
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Joel Sokol
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor John-Paul Clarke
Daniel Guggenheim School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 17 December 2015

ACKNOWLEDGEMENTS

I would like to give my deepest gratitude to Dr. Ellis Johnson for being my advisor and his insightful guidance and encouragement throughout this research work. I am honored to work with him and have learned a great deal from him. I also would like to thank Mrs. Johnson for her kindness during my multiple visits to the Hundred Acre Farm.

A special thanks to Dr. Earl Barnes who gave me many suggestions and provided so much help in organizing my proposal and defense meetings.

My sincere thanks also goes to Dr. Martin Savelsbergh, Dr. Joel Sokol, and Dr. John-Paul Clarke for taking time out of their busy schedules to serve on my dissertation committee and give me many constructive comments and suggestions. Dr. Savelsbergh provided many detailed suggestions for my thesis and helped to improve its quality significantly.

I would like to extend my thanks to Dr. Alan Erera and Ms. Amanda Ford from whom I have received all kinds of help.

Last but not the least, I am also so grateful to my family for their continuous love, encouragement and support that helps me so much.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The first part of this research work is based on Combined Objective Least Squares (COLS). We took a deeper look at matrix decomposition algorithms that are the dominating components in COLS algorithms, in terms of computational performance and numerical stability. In addition to the traditional QR decomposition approaches, this work studied other possible approaches, such as augmented system matrix and normal equations.

This research work proposed a normal equations approach for COLS, which solves linear programming problems efficiently. Even though this approach is only stable under certain conditions according to numerical analysis, we found it stable in practice and provided some possible explanations for such a phenomenon.

We also proposed a hybrid approach that could take advantage of the numerical stability of QR decomposition and the efficiency of Cholesky factorization updates so that linear programming problems could be solved reliably and efficiently. The resulting problem becomes a system of semi-normal equations, which may be further improved to achieve higher quality solutions through iterative refinement.

The second part of this research work is an improvement to the primal dual subproblem simplex method for set partitioning/packing/covering problems with convexity constraints. Primal dual subproblem simplex methods are very successful in solving large-scale set partitioning problems. In each step of the primal dual subproblem simplex method, dual feasibility is maintained and subsets of columns are selected based on a threshold value to form the restricted master problem. The optimal dual solution from the restricted master problem is used to update the current dual feasible solution and the step size used to update the dual feasible solution is calculated.

For set partitioning/packing/covering problems with convexity constraints, we discovered that longer step sizes could be selected because dual values corresponding to convexity constraints could be adjusted to maintain dual feasibility. Additionally, we found that the dual objective is a piecewise linear concave function of the step size and subsequently worked out an algorithm to find the optimal step size to maximize dual objective, so that the convergence rate could be improved. We used the long step primal dual subproblem simplex method (LPD) to solve large-scale multicommodity flow problems(MCF), and with this work, we achieved better performance than primal dual subproblem simplex methods (PD) and Dantzig-Wolfe (DW) decomposition approaches.

CHAPTER I

INTRODUCTION

Since its discovery by George Dantzig in 1947, the simplex method is one of the most widely used and successful algorithms for linear programming. Although the simplex method is not polynomially bounded, it performs as well or better than polynomial time interior point methods on a wide range of linear programming problems.

One type of linear programming problem, in which simplex methods do not perform well, is large-scale set partitioning/packing/covering problems, when the phenomenon of degeneracy influences the efficiency and convergence. When simplex methods encounter degeneracy, they might perform a number of iterations to change the basis without improving the objective value. Various methods are proposed in the literature to address the degeneracy problem.

Leichner, Dantzig and Davis [22] proposed a Nonnegative Least Squares (NNLS) method, which is impervious to degeneracy and performs better than simplex methods in solving linear programming phase I problems. Gopalakrishnan [15] and Kong [18] extended the NNLS approach to a Combined Objective Least Squares (COLS) method to solve general linear programming problems and reported improved performance for COLS compared to simplex methods.

The first part of this research work is based on COLS. We took a deeper look at matrix decomposition algorithms that are the dominating components in COLS algorithms, in terms of computational performance and numerical stability. We proposed a normal equations approach for COLS, which solves linear programming problems efficiently. Even though this approach is only stable under certain conditions according to numerical analysis, we found it stable in practice and provided some possible explanations for such a phenomenon. We also proposed a hybrid approach that could take advantage of the numerical stability of QR decomposition and the efficiency of Cholesky factorization updates so that linear programming problems could be solved reliably and efficiently. The resulting problem becomes a system of semi-normal equations, which may be further improved to achieve higher quality solutions through iterative refinement.

The second part of this research work is an improvement to the primal dual subproblem simplex method for set partitioning/packing/covering problems with convexity constraints. For this type of problems, we discovered that longer step sizes could be selected because dual values corresponding to convexity constraints could be adjusted to maintain dual feasibility. Additionally, we found that the dual objective is a piecewise linear concave function of the step size and subsequently worked out an algorithm to find the optimal step size to maximize dual objective, so that the convergence rate could be improved. We used the long step primal dual subproblem simplex method (LPD) to solve large-scale multicommodity flow problems(MCF), and with this work, we achieved better performance than primal dual subproblem simplex methods (PD) and Dantzig-Wolfe (DW) decomposition approaches.

1.1 Major Contributions

In this research work, we made multiple contributions to COLS and long step primal dual subproblem simplex methods. In COLS, we discovered that the computational bottleneck of QR decomposition approaches were steps to update and apply the Q matrix and found that linear programming problems were typically well conditioned. As a result, we concluded that the normal equations approach was suitable to solve linear programming problems using COLS. Additionally, we proposed a hybrid approach to solve least squares problems (LSQ) by taking advantage of the numerical stability of QR decomposition and the efficiency of Cholesky factorization updates. We achieved significant performance improvement over the QR decomposition approach.

The hybrid approach has its theoretical root as semi-normal equations, which can be further improved through one-step of iterative refinement, if needed. We implemented COLS variants using MATLAB and C++ and achieved improved computational performance over Gurobi optimization solvers.

In LPD, we worked out the theory and algorithm for long step primal dual subproblem simplex methods for set partitioning/packing/covering problems with convexity constraints and proved its correctness. We discovered that the dual objective value was a piecewise linear concave function and worked out an efficient algorithm to find the optimal step size for LPD. We solved the largest MCF cases Planar2500 and Chicago-Region to global optimality, which had not been solved to optimality in the literature. Additionally, we worked out an approach to form restricted master problems (RMP) using ε residual networks for large-scale MCF problems and then applied it in both PD and LPD methods for MCF problems.

Because the threshold ε is positive, phase I LPD or phase I PD approaches may end at a suboptimal solution. To find the global optimal solution, we proposed two-phase PD and LPD methods. In the first phase, phase I type RMPs are solved until a primal feasible solution or near optimal solution is found. Then we used phase II PD and LPD methods to achieve global optimality. In order to solve large-scale MCF cases, we applied column generation approaches to solve RMP problems and calculate step sizes for both PD and LPD methods. In order to solve RMP problems more efficiently, we applied row generation approaches for PD, LPD and DW methods. We also conducted extensive computational experiments on large-scale MCF problems using PD, LPD and DW methods. In these experiments, LPD methods performed much better than PD methods, which illustrated the optimal step size in LPD could significantly improve global convergence. We also demonstrated that LPD outperformed DW approaches for large-scale MCF problems.

1.2 Thesis Outline

The overall structure of the thesis has two major parts. In the first part, we explored the computational efficiency and numerical stability in solving linear programming problems using COLS and proposed Cholesky factorization based approaches to solve them efficiently. In the second part, we proposed a long step primal dual subproblem simplex method for set partitioning/packing/covering problems with convexity constrains and studied its computational performance. The thesis is organized as follows:

In Chapter 2, we provided a brief survey of solution methods for the least squares problem that was the core component for COLS. We reviewed definitions for numerical stability and the stability of various famous matrix algorithms. Applications of normal equations on linear programming and its stable performance in practice were contrasted with its instability in theory.

In Chapter 3, we discussed NNLS that solves the linear programming phase I problem using active set algorithms, in which a set of basic columns are selected at each iteration, add one more column to the basis and (if necessary) remove one or more columns from the basis across consecutive iterations. Columns in the basis are automatically independent at all times. We explored different ways to solve NNLS, such as QR decomposition, augmented systems, normal equations and compared their computational performance and numerical stability. Algorithms to update factorization were explored so that a column could be added to or removed from the basis efficiently, which was critical in active set methods.

In Chapter 4, we discussed COLS as an extension of NNLS by adding a linear term to the objective. The computation in each step of COLS closely resembles NNLS but we need extra effort to maintain independence of columns in the basis. The COLS approach solves the linear programming problems in a significantly fewer number of iterations than normally in use with simplex methods because it moves outside of the feasible region and avoids degeneracy. Computational results are provided to compare the performance with different matrix factorization approaches as well as the simplex method.

In Chapter 5, we proposed the long step primal dual subproblem simplex method and the algorithm to calculate the optimal step size for set partitioning/packing/covering problems with convexity constraints.

In Chapter 6, we used long step primal dual subproblem simplex method to solve large-scale multicommodity flow problems, which are modeled as a set packing problem with convexity constraints. We solved two previous unsolved problems in the literature to global optimality and compared its computational performance with primal dual subproblem simplex methods and Dantzig-Wolfe decomposition.

In Chapter 7, we concluded this research work and provided suggestions for future research.

CHAPTER II

LEAST SQUARES PROBLEMS AND MATRIX TECHNIQUES

In this chapter, we introduce classical solution methods for least squares problems, such as normal equations, QR decomposition and singular value decomposition. We compare the theoretical computational performance and numerical stability for least squares and linear programming problems. Finally, we discuss the popularity and computational performance of LU decomposition and normal equations in linear programming problems in spite of their less desirable numerical stability in theory.

2.1 Least Squares Problems

The least squares problem was first introduced by Gauss in 1800's and used widely in many scientific areas including statistics and optimization.

Given a linear system $Ax = b$, it does not have a solution if b is not in the range of matrix A. Least squares problems (LSQ) want to minimize the Euclidean length of $b - Ax$.

Definition 1. *Let $A \in \Re^{m \times n}$ and $b \in \Re^m$. The least squares problem finds a vector $x^* \in \Re^n$ such that*

$$min_x \|b - Ax\|^2 = \|b - Ax^*\|^2$$

We assume that matrix A and vector b consist of real numbers and that matrix A has full column rank, i.e. the rank of A is n, where n is the number of columns, m is the number of rows of matrix A and $m \geq n$.

Least squares problems are solved through orthogonal projection as illustrated in Figure 2.1, where $r = b - Ax$ is the residual, $P \in R^{m \times m}$ is the orthogonal projector and $y = Pb = Ax$ is vector b's orthogonal projection on $range(A)$.

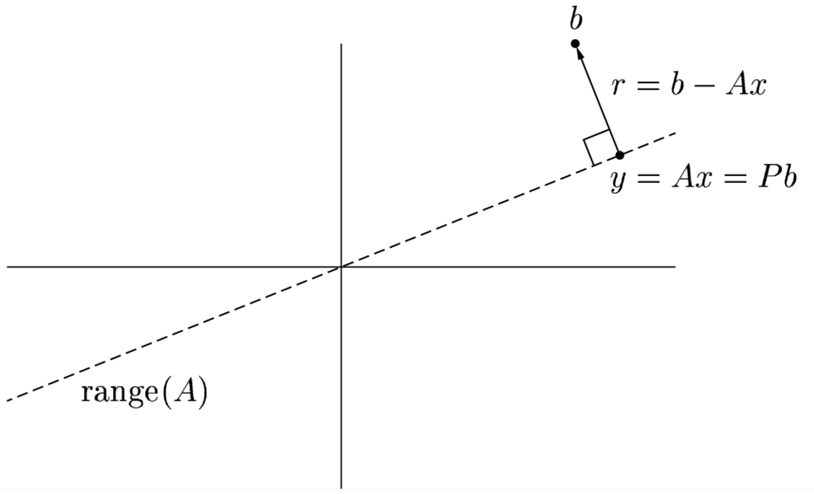


Figure 2.1: Least Squares and Orthogonal Projection [27].

2.2 Optimality Conditions and Solution Methods for LSQ

The least squares problem is to find the closest point Ax so that the norm of residual r is minimized. Geometrically the optimal solution is found when r is orthogonal to $range(A)$, that is $A^T r = 0$, or $A^T Ax = A^T b$ if we plug in $r = b - Ax$.

As $A^T A$ is a normal matrix, $A^T Ax = A^T b$ is known as the normal equation. The optimal solution of LSQ must satisfy this equation.

Theorem 1. *Let matrix $A \in \Re^{m \times n}$ with full column rank and vector $b \in \Re^m$, a vector $x \in \Re^n$ solves the least squares problem* $\min \|b - Ax\|^2$, *if*

$$A^T Ax = A^T b$$

Proof. As matrix A has full column rank, $A^T A$ is positive definite and $\nabla^2 f(x) \succ 0$. From $A^T Ax = A^T b$, we have $\nabla f(x) = 0$. Therefore, x is the optimal solution for

$\min \|b - Ax\|^2$. Additionally, as A^TA is nonsingular, the least squares solution is unique.

□

2.2.1 Normal Equations

As matrix A is a real matrix with full column rank, matrix $C = A^TA$ is symmetric positive definite (or Hermitian positive definite if A is complex). The normal equation $A^TAx = A^Tb$ has a unique solution x^* that solves the least squares problem.

Numerical solution methods for normal equations date back to Gauss, who solved for x using back substitution, while preserved symmetry of the normal matrix by elimination. Gauss' method is related closely to Cholesky factorization, which was discovered by Andre-Louis Cholesky.

Theorem 2. *(Cholesky factorization) If $A \in \Re^{n\times n}$ is symmetric positive definite, then there exists a unique lower triangular matrix $L \in \Re^{n\times n}$ with positive diagonal elements such that $A = LL^T$.*

Proof. See [25].

□

As C is symmetric, only the low triangular part needs to be stored. To find Cholesky factorization, we first form the $C = A^TA \in R^{n\times n}$ matrix and calculate the entries of L using the following formulas:

$$L_{jj} = \sqrt{(C_{jj} - \sum_{k=1}^{j-1} L_{jk}^2)} \tag{2.2.1}$$

$$L_{ij} = \frac{1}{L_{jj}}(C_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk}), \text{where } i < j \tag{2.2.2}$$

There is an alternative form for Cholesky factorization, $C = LDL^T$, where $L \in \Re^{n\times n}$ is a lower triangular matrix with diagonal elements as 1 and $D \in \Re^{n\times n}$ is a positive diagonal matrix. Entries of L and D are calculated using the following formulas:

$$\begin{aligned} D_j &= (C_{jj} - \sum_{k=1}^{j-1} L_{jk}^2) && (2.2.3) \\ L_{ij} &= \frac{1}{D_j}(C_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk}D_k), \text{where } i < j && (2.2.4) \end{aligned}$$

The LDL^T factorization is slightly faster than the LL^T factorization, because it does not calculate square roots and as a result, it can avoid some possible rounding errors. However, these factorizations are equivalent and it is easy to convert between them. In this thesis, we use LDL^T and LL^T factorizations interchangeably for the Cholesky factorization. Many other algorithms for Cholesky factorization are available and some of them can be found in the references [7][10].

After $C = LL^T$ factorization, two triangular systems are solved for the least squares solution x:

$$\begin{aligned} L^T y &= A^T b && (2.2.5) \\ Lx &= y && (2.2.6) \end{aligned}$$

2.2.2 QR Decomposition for Least Squares

The normal equations approach was the default method for least squares problems until Golub [11] developed a stable method to use Householder QR factorization to solve least squares problems in 1965.

Given a QR decomposition $QR = A$, where $Q \in \Re^{m \times m}$ is an orthogonal matrix, that is $QQ^T = Q^T Q = I$, where I is an identity matrix, $R \in \Re^{m \times n}$ is upper diagonal with 0 entries in row $n + 1$ to m, the least squares problem is solved by:

$$\begin{aligned} y &= Q^T b && (2.2.7) \\ R_1 x &= y_1 && (2.2.8) \end{aligned}$$

where $R = \begin{pmatrix} R_1 \\ R_2 \end{pmatrix}$, $R_1 \in \Re^{n\times n}$ is upper triangular, $R_2 \in \Re^{(m-n)\times n}$ is a zero matrix and y_1 is the first m elements of vector y.

There are many algorithms to calculate QR factorization, such as Gram-Schmidt decomposition, Modified Gram-Schmidt decomposition, Householder transformation and Givens rotation. Details of these algorithms can be found in additional resources [27].

2.2.3 Singular Value Decomposition for Least Squares

Theorem 3. *(Singular Value Decomposition) If matrix $A \in \Re^{m\times n}$ with rank $k \leq \min(m, n)$, then there is an orthogonal matrix $U \in \Re^{m\times m}$, an orthogonal matrix $V \in \Re^{n\times n}$ and a diagonal matrix $S \in \Re^{m\times n}$ such that*

$$U^T AV = S, A = USV^T$$

Diagonal entries of S (called singular values of matrix A) are nonnegative and non-increasing and exactly k of them are strictly positive.

Proof. See [21].

□

Singular value decomposition can be widely used for many applications, in addition to solving least squares problems. However, its usage was limited until Golub and Kahan [12] proposed the first stable approach for singular value decomposition in 1965.

In Golub and Kahan's approach, matrix A was first reduced to bi-diagonal form. A bidiagonal matrix is a matrix with nonzero diagonal entries and either nonzero on the diagonal above or the diagonal below.

Here is an example of an upper bidiagonal matrix:

$$\begin{pmatrix} \times & \times & & \\ & \times & \times & \\ & & \times & \times \\ & & & \times \end{pmatrix}$$

and a lower bidiagonal matrix:

$$\begin{pmatrix} \times & & & \\ \times & \times & & \\ & \times & \times & \\ & & \times & \times \end{pmatrix}$$

Then we compute the singular value decomposition from the bidiagonal matrix using an iterative method.

The first step to reduce matrix A to the bidiagonal matrix can be achieved using Householder transformation.

Given a singular value decomposition $U \begin{pmatrix} S_r^{-1} & 0 \\ 0 & 0 \end{pmatrix} V^T = A$, where $U \in \Re^{m \times m}$ is an orthogonal matrix, $V \in \Re^{n \times n}$ is an orthogonal matrix and $S_r \in \Re^{k \times k}$ is diagonal, where k is the rank of matrix A.

Theorem 4. *(Least squares problems by Singular Value Decomposition) If matrix $A \in \Re^{m \times n}$ with rank $k \leq \min(m, n)$, then the least squares problem* $\min \|b - Ax\|^2$ *has a unique solution*

$$x = V \begin{pmatrix} S_r^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T b$$

where $S_r \in \Re^{k \times k}$ *is a diagonal matrix,* $S = \begin{pmatrix} S_r & 0 \\ 0 & 0 \end{pmatrix}$ *and* $A = USV^T$ *is the singular value decomposition for matrix* A.

Proof. See [7].

□

If matrix A is rank-deficient or ill-conditioned, Cholesky factorization and QR decomposition approaches may not be stable, then singular value decomposition will be the best in solving these least squares problems.

2.3 Augmented System Approaches for LSQ

From the normal equation, we have $A^T Ax = A^T b$ and we let the residual be $r = b - Ax$, then the optimal residual has $Ar^* = Ab - A^T Ax^* = 0$.

$A^T x^*$ is the projection of $b \in R^n$ onto the space of the columns of A^T and the residual lies in the null space of A. Rewrite the above equations in matrix notation, we have the augmented system matrix:

$M = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}$, which is nonsingular if matrix A has full column rank. The augmented system matrix M is a symmetric indefinite matrix, because $M = M^T$ and its eigenvalues can be either positive or negative.

Given matrix $A \in \Re^{m \times n}$, $b \in \Re^m$, we can use LU decomposition approach to solve the least squares problem $\min \|b - Ax\|^2$.

Let $M \in \Re^{(m+n) \times (m+n)}$, where $M = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}$. The least square problem can be solved by

$$M \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

where $x \in \Re^m$ and $s \in \Re^n$ are the solution of the least squares problem and $s = b - Ax$ is the residual.

Theorem 5. *The least squares problem* $\min \|b - Bx\|^2$ *is solved by*

$$\begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} b \\ 0 \end{pmatrix}$$

Proof. We can rewrite the equation as $\begin{cases} Ax + s = b \\ B^T s = 0 \end{cases}$. Rearrange it and we have $s = b - Ax$ and plug-in to the second equation, we have $A^T(b - Ax) = 0$, which is the normal equation for the least squares problem.

□

As M is nonsingular, we have $M = LU$, where $L \in \Re^{(m+n)\times(m+n)}$ and lower triangular, and $U \in \Re^{(m+n)\times(m+n)}$ and upper triangular. We can solve for y using back substitution on $Ly = \begin{pmatrix} b \\ 0 \end{pmatrix}$, where $y = U \begin{pmatrix} s \\ x \end{pmatrix}$. Then, we solve for $\begin{pmatrix} s \\ x \end{pmatrix}$ using back substitution.

2.4 Numerical Stability

Numerical stability is one of the important criteria in use to evaluate algorithms. Let a mathematical problem be a function $f : X \to Y$ from a vector space X of inputs to a vector space Y of solutions and let an algorithm for the mathematical problem as a function $\overline{f} : X \to Y$. $\epsilon_{machine}$ is the machine epsilon, which is 2^{-24} and 2^{-53} for IEEE single and double precision arithmetic [27].

Definition 2. *The algorithm $\overline{f}$ for a problem f is stable if for each $x \in X$, we have [27]:*

$$\frac{\|\overline{f}(x) - f(\overline{x})\|}{\|f(\overline{x})\|} = O(\epsilon_{machine})$$

for some $\overline{x}$ such that

$$\frac{\|\overline{x} - x\|}{\|x\|} = O(\epsilon_{machine})$$

Definition 3. *The algorithm $\overline{f}$ for a problem f is backward stable if for each $x \in X$, we have [27]:*

$$\overline{f}(x) = f(\overline{x})$$

for some $\overline{x}$ such that

$$\frac{\|\overline{x} - x\|}{\|x\|} = O(\epsilon_{machine})$$

It is well known that back substitution, QR decomposition using Householder transformation, Givens rotation, Modified Gram-Schmidt decomposition and Cholesky factorization are backward stable [27]. However, normal equations approaches for least squares problems are stable only under certain conditions that we will elaborate on in Theorem 6.

Theorem 6. *Stability of least squares problems*

1. *Least squares problem using QR decomposition is backward stable.*

2. *Least squares problem using singular value decomposition is backward stable.*

3. *Least squares problem using normal equations is stable only if $\kappa(A)$ is uniformly bounded above or $\frac{\tan\theta}{\eta}$ is uniformly bounded below, where $\kappa(A)$ is the condition number of matrix A, $\theta = \cos^{-1}\frac{\|Ax\|}{\|b\|}$ is the closeness of the fit, $\eta = \frac{\|A\|\|x\|}{\|Ax\|}$.*

Proof. See [27].

□

2.4.1 An Illustrative Example

Trefethen and Bau [27] provided an illustrative example to show the stability of various methods for least squares problems.

The A matrix is a 100 by 15 Vandermonde matrix, which is well known for its ill conditioning. The following is an $m \times n$ Vandermonde matrix:

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^{n-1} \end{pmatrix}$$

The right hand side is a function $e^{sin(4\tau)}$ on the interval $[0, 1]$.

We setup the matrix as outlined in [27], the condition number of matrix A is $2.271777310158213 \times 10^{10}$, $\theta = 3.746111084567305 \times 10^{-6}$, $\eta = 210355.9748332728$ and $\frac{\tan\theta}{\eta} = 1.780843680600934 \times 10^{-11}$. This least squares problem is expected to be challenging for normal equations according to Theorem 6 .

Five different methods were used to solve the least squares problem and their results were illustrated in Table 2.1 by showing the value of the 15th element of the solution vector x, i.e. $x(15)$, together with the expected solution.

Table 2.1: Stability of least squares algorithms

Method	Value
The expected solution	1.0
Householder transformation	1.00000031528723
Gram-Schmidt decomposition	1.02926594532672
Modified Gram-Schmidt decomposition	1.00000005653399
Singular value decomposition	0.99999998230471
Normal equations	0.39339069870283

From Table 2.1, Householder transformation, modified Gram-Schmidt and singular value decomposition achieved similar solution accuracy. The normal equations approach caused significant errors, while Gram-Schmidt decomposition was not accurate either, though it was better than the normal equations approach.

2.4.2 Practical Performance of Normal Equations

Although the normal equations approach can be unstable under certain conditions, it is in use in various areas such as linear programming (interior point methods). It achieved good computational results and numerical problems are rare even for ill-conditioned matrices.

We focused on interior point methods for linear programming because of its popularity and well-known success in practice. Interior point methods provide competitive computational performance in linear programming problems in comparison with classical simplex methods. An interior point method computes a series of direction vectors and moves toward the optimal solution. In each iteration of the interior point method, a linear system $A\Sigma^2 A^T x = b$ is solved, where the A matrix remains the same throughout the algorithm and the matrix Σ is a diagonal matrix that changes in each iteration. Primal, dual and primal dual variants of the interior point method differ by the way matrix Σ is formed. This normal equation is typically solved using Cholesky factorization $LDL^T = A\Sigma^2 A^T$ [24].

It is known that the matrix $A\Sigma^2 A^T$ becomes ill conditioned when the algorithm moves near the optimal solution. However, the impact of ill conditioning of the normal matrix does not have the dramatic effect on solution accuracy [14].

An alternative approach for normal equations is the *augmented system* approach, in which a symmetric indefinite augmented matrix

$$\begin{pmatrix} \Sigma^{-2} & A^T \\ A & 0 \end{pmatrix}$$

is factorized. The augmented system approach provides highly accurate results, but is on average about 40% less efficient than normal equations approaches [14].

2.4.3 Stability and Practical Performance of LU Decomposition

LU decomposition is the dominate part of simplex methods. An efficient and numerical stabile LU solver is important given its success in solving linear programming problems.

From numerical analysis, LU decomposition algorithm with Gaussian elimination is not stable but Gaussian elimination with partial pivoting is backward stable [27].

However, being backward stable does not free it from numerical problems. Trefethen and Bau [27] gave an illustrative example for Gaussian elimination with partial pivoting:

$$A = \begin{pmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ -1 & -1 & 1 & & \\ -1 & -1 & -1 & 1 & \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{pmatrix}$$

For this matrix, the growth factor is $\rho = 16$. For an $m \times m$ matrix, the growth factor is exponential, i.e. $\rho = 2^{m-1}$, which corresponds to a loss of order $m - 1$ bits of precision.

The above example illustrates that Gaussian elimination with partial pivoting is backward stable in theory, but it can still produce catastrophic results in practice [27].

It is interesting to note that Gaussian elimination with partial pivoting is very stable in practice, even though there are worst-case examples with exponential growth factors. It seems that Gaussian elimination did not encounter such worst cases in practice. Trefethen and Bau [27] claimed that no matrix problems that cause instability in Gaussian elimination are known under natural circumstances in 50 years of computing. Golub and Van Loan [13] also said Gaussian elimination with partial pivoting could be used with confidence and such exponential growth was highly unlikely in practice.

Trefethen and Bau [27] gave an explanation based on statistical reasoning: matrices encountered in practice were not random and matrices for which Gaussian elimination was unstable were so rare; therefore, statistically it was almost impossible to encounter such a matrix in practice.

2.4.4 Summary

Normal equations approaches are significantly faster than QR decomposition approaches, especially when a series of least squares problems are solved. However QR approaches are backward stable, which can reliably solve all types of least squares problems with full column rank. Therefore, it is suitable to serve as a black box least squares solver.

Normal equations approaches are only suitable for problems with certain features. There are known problems that cannot be solved by normal equations such as Vandermonde matrices as we illustrated. However, normal equations approaches are successful in solving linear programming problems, without encountering numerical problems.

In this research work, we chose to use normal equations to solve least squares problems as subproblems in solving linear programming, so that we could solve each subproblem more efficiently. At the same time, we were fully aware of the risk associated with normal equations and prepared approaches to address that if needed.

CHAPTER III

NONNEGATIVE LEAST SQUARES

In this chapter, we introduce solution methods for nonnegative least squares problems. We first introduce the classical active set method and its implementation using QR decomposition. We then compare the active set method with the phase I simplex method and highlight the similarities and differences. Finally, we solve the nonnegative least squares problem by solving a series least squares problem in each iteration.

3.1 Introduction

The Nonnegative Least Squares problem (NNLS) is defined as follows: given a real $m \times n$ matrix A of rank $\min\{m, n\}$, i.e.$A \in \Re^{m \times n}$, $b \in \Re^m$, find $0 \leq x \in \Re^n$ minimizing the square of the Euclidean length of $Ax - b$, i.e. $\|Ax - b\|^2$.

The Euclidean length or Euclidean norm of a vector $v \in \Re^m$, denoted by $\|v\|$, is defined by $\|v\| = \sqrt{v'v} = \sqrt{\sum_{i=1}^{m} v_i^2}$.

The NNLS problem can be formulated as follows:

$$\min \|Ax - b\|^2 \tag{3.1.1}$$

Subject to:

$$x \geq 0 \tag{3.1.2}$$

Or equivalently

$$\min\{\|b - Ax\|^2 : x \geq 0\} \tag{3.1.3}$$

The NNLS problem is a special case of least squares problems with linear inequality constraints (LSI):

$$\min\{\|b - Ax\|^2 : l \leq Dx \leq u\} \tag{3.1.4}$$

when $l = 0$, $u = +\infty$ and D is an identity matrix.

If D is an identity matrix the LSI problem becomes the Bound-constrained least squares problem(BLS):

$$\min\{\|b - Ax\|^2 : l \leq x \leq u\} \tag{3.1.5}$$

If the matrix A has full column rank, BLS is a strictly convex optimization problem. Then BLS has a unique solution for any vector b and is known to be solvable in polynomial time [7].

In general, problems with linear inequality constraints are often solved using *active set methods*, based on the following observations: at the optimal solution for problem LSI, a certain subset of constraints $l \leq Dx \leq u$ will be active, i.e. with equality. If this subset was known, the solution to original LSI would be the same as the LSI problem with the active constraints only and these constraints can be replaced by equality constraints(LSE):

$$\min\{\|b - Ax\|^2 : Dx = d\} \tag{3.1.6}$$

There are many efficient solution methods available for least squares problem with equality constraints including the methods of direct elimination and null space method [7] [21].

As for the NNLS problem, if the subset of active constraints are known *a priori*, then it becomes a least squares problem with only the columns in matrix A to correspond to the active constraints.

The least squares problem

$$\min\{\|b - Ax\|^2\} \tag{3.1.7}$$

can be solved using normal equations, QR decomposition or Singular Value Decomposition as introduced in the previous chapter.

The active set method for the NNLS problem will be introduced as follows [21]: index set P and Z will be defined and modified in the course of the active set algorithm, where P denote the active set of columns. Variables indexed in set Z will be set to zero. Variables indexed in set P will be determined using least squares methods. If a variable indexed in set P takes a non-positive value, the algorithm will move its index from set P to set Z.

On termination of the algorithm, the solution vector x is:

$$x_i > 0 \qquad i \in P \tag{3.1.8}$$

$$x_i = 0 \qquad i \in Z \tag{3.1.9}$$

where $x_i > 0, i \in P$ is the solution of the least square problem

$$A_p x_p = b \tag{3.1.10}$$

where A_p is the set of columns with indices in P, $x_p > 0$ is the solution vector with indices in P, $x_z = 0$ is the solution vector with indices in Z and $x_p \cup x_z = x$.

The dual vector w satisfies

$$w_i = 0 \qquad i \in P \tag{3.1.11}$$

$$w_i \leq 0 \qquad i \in Z \tag{3.1.12}$$

$$w = A^T(b - Ax) \tag{3.1.13}$$

The above conditions constitute the KKT conditions for NNLS, which is introduced in the next section, and thus the active set method terminates with the optimal solution for the NNLS problem.

See additional reference [21] for details of the active set method, correctness proof and finite termination of the algorithm.

3.1.1 KKT Conditions for NNLS

Theorem 11 specifies the optimality solution for Problem NNLS.

Theorem 7. *A vector $\hat{x}$ is a solution for the NNLS problem* $\min\{\|Ax-b\|^2 : x \geq 0\}$ *if and only if there exists $\hat{w} \in \Re^m$ and a partitioning of the integers 1 through m into subsets P and Z such that*

1. $\hat{w} = A^T(A\hat{x}-b)$
2. $\hat{x}_i = 0$, *where* $i \in Z$; $\hat{x}_i > 0$, *where* $i \in P$;
3. $\hat{w}_i \geq 0$, *where* $i \in Z$; $\hat{w}_i = 0$, *where* $i \in P$;

Proof. See [15] and [18].

□

3.1.2 Properties of NNLS

Theorem 8. *If A_p is a basis, x is the solution for* $\min\|b - A_p x\|^2$, $r = b - Bx$ *and A_s is the incoming column with $r^T A_s > 0$, there are following properties:*

1. *(ρ is orthogonal to columns of A_p)* $r^T A_p = 0$
2. *(Strict Improvement)* $\min\|b - A_p x_p - A_s x_s\|^2 < \min\|b - A_p x_p\|^2$
3. *(Positive solution value for the incoming column) The problem* $\min\|b - A_p x_p - A_s x_s\|^2$, *has solution* $x_s > 0$
4. *(Independent incoming column)* $[A_p, A_s]$ *is an independent set of columns*
5. *(Independent columns in the set of non-basic columns) If* $r^T A_t \neq 0, t \in Z$, $[A_p, A_t]$ *is an independent set of columns*

Proof. See [15] and [18].

□

3.2 The Implementation

The least squares method being solved in the active set methods differ between consequent iterations only by the addition of one more column or the deletion of one or more columns. To solve least square problems more efficiently, efficient updating techniques can be used to compute the updated matrix factorization for the new problem based upon the matrix factorization from the previous iteration.

The ability to compute solutions to a sequence of least squares problems in an efficient manner is essential for successful NNLS algorithms. Well-known least squares solution methods such as normal equations via Cholesky factorization and QR decomposition have algorithms to accomplish these tasks efficiently.

3.2.1 Update Matrix factorization in QR decomposition

There are different algorithms to update matrix factorizations across iterations in QR decomposition:

- Use Householder transformation to add a column and use Givens rotation to remove columns.
- Use Householder transformation to add and remove columns.
- Use Givens rotation to add and remove columns

As the Givens rotation approach is better in handling sparse matrices, we chose the last approach.

3.2.1.1 Removing a Column

If the column to be removed is the last column, it will be easy to eliminate. Suppose the active set is $A_p = [\overline{A_p}, A_s]$, where A_s is the column to be removed. We have

$$Q^T A_p = Q^T[\overline{A_p}, A_s] = \begin{pmatrix} R_{11} & r_{1s} \\ 0 & r_{ss} \\ 0 & 0 \end{pmatrix} \tag{3.2.1}$$

The updated QR decomposition is

$$Q^T \overline{A_p} = \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \tag{3.2.2}$$

that is, simply drop the last column in R matrix and keep matrix Q unchanged.

If the column to drop is not the last column, the resulting R matrix may not be upper triangular, i.e.

$$Q^T \overline{A_p} = \begin{pmatrix} R_{11} & \hat{R_{12}} \\ 0 & \hat{R_{22}} \\ 0 & 0 \end{pmatrix} \tag{3.2.3}$$

The submatrix R_{11} is still upper triangular. The submatrix $\hat{R_{22}}$ is not upper triangular, but it is upper triangular with non-zeros immediately below the diagonal elements i.e.

$$\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{pmatrix} \tag{3.2.4}$$

A series of Givens rotations can be used to eliminate the nonzero elements below the diagonal and make the submatrix $\hat{R_{22}}$ upper triangular.

The general approach to remove the sth column is as follows:

1. Drop the sth column

2. Apply QR decomposition on submatrix $\hat{R_{22}}$ to make it upper triangular, where $\hat{Q}$ is the corresponding orthogonal matrix

3. The updated $Q := \hat{Q}Q$

3.2.1.2 Adding a Column

Suppose the active set is A_p, its QR decomposition is $QR = A_p$ and A_s is the column to be added, the updated active set of columns will be $\overline{A_p} = [A_p A_s]$. Multiple Q^T on $\overline{A_p}$, we have

$$Q^T\overline{A_p} = Q^T[A_p, A_s] = \begin{pmatrix} R & r_{1s} \\ 0 & r_{2s} \end{pmatrix} \tag{3.2.5}$$

If A_s is independent of A_p, $Q^T A_s = \begin{pmatrix} r_{1s} \\ r_{2s} \end{pmatrix}$, where $r_{1s} \in R^{|P|}$ and $r_{2s} \neq 0$.

The resulting matrix is of the form:

$$\begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{pmatrix} \tag{3.2.6}$$

A series of Givens rotations can be used to eliminate the nonzero elements in r_{2s}, except the first row.

The general approach to add a column is as follows:

1. Add the sth column to the right of the existing columns.

2. Multiple Q^T on column A_s.

3. Apply QR decomposition on vector r_{2s} to make it upper triangular, where $\hat{Q}$ is the corresponding orthogonal matrix.

4. The updated $Q := \hat{Q}Q$.

3.2.1.3 Re-factorization

As columns are added by iterations, the ordering of columns in set P may not be the best to reduce fill-ins. Additionally, after multiple iterations, the number of Givens rotation matrices can grow and it might be more time consuming to update than to conduct the QR decomposition from scratch. Therefore, it is necessary to do re-factorization.

By refactoring a matrix after certain number of iterations, we can reorder columns to minimize fill-ins and reduce the Givens rotation matrix size, thus significantly reducing consequent computational burdens.

3.2.2 Update Matrix factorization in Normal equations using Cholesky factorization

In normal equations, we can update matrix factorizations using rank-1 updates.

3.2.2.1 Removing a Column

Davis and Hager [9] proposed a row modification algorithm to remove a row and column simultaneously by making a rank-1 update of the Cholesky factorization. In this section, we extend Davis and Hager's algorithm to remove a column from the basis and generate the updated Cholesky factorization without explicitly forming the normal equation.

Given a basis $B = \{a_1, ..., a_k\}$ in the combined objective least squares problem, the Cholesky factorization is

$$LDL^T = B^TB = \begin{pmatrix} a_1^T \\ \vdots \\ a_k^T \end{pmatrix} \begin{pmatrix} a_1 & \dots & a_k \end{pmatrix} = \begin{pmatrix} a_1^Ta_1 & \dots & a_k^Ta_1 \\ \vdots & \ddots & \vdots \\ a_1^Ta_k & \dots & a_k^Ta_k \end{pmatrix} \tag{3.2.7}$$

Suppose column r is dropped from the basis, we set $a_r = 0$ and

$\bar{B} = \{a_1, ..., a_{r-1}, 0, a_{r+1}, ..., a_k\}$.

Let $B^T B = \begin{pmatrix} C_{11} & c_{12} & C_{31}^T \\ c_{12}^T & c_{22} & c_{32}^T \\ C_{31} & c_{32} & C_{33} \end{pmatrix}$ where $c_{12}^T = [a_1, ..., a_{r-1}]^T a_r$, $c_{22} = a_r^T a_r$, $c_{32}^T = [a_{r+1}, ..., a_k]^T a_r$, then

$$\bar{B}^T \bar{B} = \begin{pmatrix} C_{11} & 0 & C_{31}^T \\ 0^T & 0 & 0^T \\ C_{31} & 0 & C_{33} \end{pmatrix} \tag{3.2.8}$$

Let

$$L = \begin{pmatrix} L_{11} & & \\ l_{12}^T & 1 & \\ L_{31} & l_{32} & L_{33} \end{pmatrix} \tag{3.2.9}$$

and

$$D = \begin{pmatrix} D_1 & & \\ & d_2 & \\ & & D3 \end{pmatrix} \tag{3.2.10}$$

We have

$$B^T B = LDL^T = \begin{pmatrix} L_{11} & & \\ l_{12}^T & 1 & \\ L_{31} & l_{32} & L_{33} \end{pmatrix} \begin{pmatrix} D_1 & & \\ & d_2 & \\ & & D3 \end{pmatrix} \begin{pmatrix} L_{11} & & \\ l_{12}^T & 1 & \\ L_{31} & l_{32} & L_{33} \end{pmatrix}^T$$

$$= \begin{pmatrix} L_{11} D_1 L_{11}^T & L_{11} D_1 l_{12} & L_{11} D_1 L_{31}^T \\ l_{12}^T D_1 L_{11}^T & l_{12}^T D_1 l_{12} + d_2 & l_{12}^T D_1 L_{31}^T + d_2 l_{32}^T \\ L_{31} D_1 L_{11}^T & L_{13} D_1 l_{12} + l_{32} d_2 & L_{31} D_1 L_{31}^T + l_{32} d_2 l_{32}^T + L_{33} D_3 L_{33}^T \end{pmatrix}$$

After setting $a_r = 0$, we have

$$\bar{B}^T \bar{B} = \bar{L} \bar{D} \bar{L}^T = \begin{pmatrix} L_{11} & & \\ 0^T & 1 & \\ L_{31} & 0 & \bar{L}_{33} \end{pmatrix} \begin{pmatrix} D_1 & & \\ & 0 & \\ & & D3 \end{pmatrix} \begin{pmatrix} L_{11} & & \\ 0^T & 1 & \\ L_{31} & 0 & \bar{L}_{33} \end{pmatrix}^T$$

$$= \begin{pmatrix} L_{11}D_1L_{11}^T & 0 & L_{11}D_1L_{31}^T \\ 0^T & 0 & 0^T \\ L_{31}D_1L_{11}^T & 0 & L_{31}D_1L_{31}^T + \bar{L}_{33}\bar{D}_3\bar{L}_{33}^T \end{pmatrix}$$

We have the following two equations from the original factorization and the new factorization:

$$C_{33} = L_{31}D_1L_{31}^T + l_{32}d_2l_{32}^T + L_{33}D_3L_{33}^T \tag{3.2.11}$$

$$C_{33} = L_{31}D_1L_{31}^T + \bar{L}_{33}\bar{D}_3\bar{L}_{33}^T \tag{3.2.12}$$

Combine these two equations and we have

$$\bar{L}_{33}\bar{D}_3\bar{L}_{33}^T = L_{33}D_3L_{33}^T + l_{32}d_2l_{32}^T = L_{33}D_3L_{33}^T + ww^T \tag{3.2.13}$$

where $w = l_{32}\sqrt{d_2}$.

The row deletion algorithm is very similar to Davis and Hager's algorithm [9]:

Step 1. $\bar{l}_{12} = 0$, $\bar{d}_2 = 0$, $\bar{l}_{32} = 0$.

Step 2. $w = l_{32}\sqrt{d_2}$.

Step 3. Perform the rank-1 update $\bar{L}_{33}\bar{D}_3\bar{L}_{33}^T = L_{33}D_3L_{33}^T + ww^T$.

Rank-1 update of Cholesky factorization: The rank-1 update can be solved efficiently using referenced algorithms [10].

3.2.2.2 Adding a Column

To add a column to the Cholesky factorization, we use a similar approach as a variant of Cholesky factorization, namely up-looking Cholesky factorization [10]:

Suppose we have 2 by 2 block Cholesky factorization $LL^T = A$ where $L = \begin{pmatrix} L_{11} & \\ l_{12}^T & l_{22} \end{pmatrix}$ and $A = \begin{pmatrix} A_{11} & a_{12} \\ a_{12}^T & a_{22} \end{pmatrix}$, lower triangular matrix $L_{11} \in \Re^{(n-1)\times(n-1)}$ and $A_{11} \in \Re^{(n-1)\times(n-1)}$.

Suppose we know $L_{11}L_{11}^T = A$, the up-looking method calculates the current Cholesky factorization by solving recursively the following two equations:

$$\begin{aligned} L_{11}l_{12} &= a_{12} && (3.2.14) \\ l_{12}^T l_{12} + l_{22}^2 &= a_{22} && (3.2.15) \end{aligned}$$

We extend the up-looking Cholesky factorization to the row addition algorithm. Suppose the active set is A_p, its Cholesky factorization is $LDL^T = A_p^T A_p$ and A_s is the column to be added, which is independent of A_p, the new active set is $\bar{A}_p = [A_p A_s]$ and the normal equation is $\bar{A}_p^T \bar{A}_p = \begin{pmatrix} A_p^T \\ A_s^T \end{pmatrix} [A_p A_s] = \begin{pmatrix} A_p^T A_p & A_p^T A_s \\ A_s^T A_p & A_s^T A_s \end{pmatrix}$

Let the new Cholesky factorization be

$$\bar{L}\bar{D}\bar{L}^T = \begin{pmatrix} L & \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} D & \\ & d \end{pmatrix} \begin{pmatrix} L^T & l_{21}^T \\ & l_{22} \end{pmatrix} = \begin{pmatrix} LDL^T & LDl_{21}^T \\ l_{21}DL^T & l_{21}Dl_{21}^T + l_{22}dl_{22} \end{pmatrix} \quad (3.2.16)$$

As $\bar{A}_p^T \bar{A}_p = \bar{L}\bar{D}\bar{L}^T$, we have the following equations:

$$\begin{aligned} A_p^T A_s &= LDl_{21}^T && (3.2.17) \\ A_s^T A_s &= l_{21}Dl_{21}^T + l_{22}dl_{22} && (3.2.18) \end{aligned}$$

Use back substitution on the first equation to solve for l_{21} and plug in to the second equation to calculate l_{22}.

The updated active set of columns will be $\overline{A_p} = [A_p A_s]$.

3.2.2.3 Re-factorization

As columns are added by multiple iterations, the ordering of columns in set P may not be the best to reduce fill-ins similar to the QR approach. Additionally, the row deletion algorithm might introduce additional rounding errors rather than starting from scratch. Then, it is necessary to do re-factorization after a certain number of iterations.

3.2.3 A Hybrid Approach

In the previous sections, we introduced NNLS implementations using QR decomposition and Cholesky factorization. However both approaches have potential drawbacks. For the QR approach, it can take a great deal of computer processing time and memory to update Q matrix when columns are added and removed from the basis. Kong [18] stored the Q^T matrix as a product form of transpose with four elements *cos*, *sin* and two row indices i, j based on Givens rotation.

The right hand side becomes $Q^T b = G_k G_{k-1} ... G_2 G_1 b$. Because the right hand side b is unchanged, we can accumulatively multiple it by G_i at each step i. However, when a new column A_s is introduced into the basis, $Q^T A_s$ is calculated in forming the products $G_k G_{k-1} ... G_2 G_1 A_s$, where $G_i, i = 1, ..., k$ are the Givens rotations formed in each iteration. When k is large, this step can be time consuming and rounding errors can accumulate. Therefore, the update step for QR decomposition can become more time consuming, especially in iterations when a column is added to the basis.

The time complexity of updating a Cholesky factorization is not changing across iterations. However, Cholesky factorization can encounter numerical difficulties due to rounding errors. For example, suppose the basis is $A_p = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1-10^{-8} \end{pmatrix}$ and $C = A_p^T A_p = \begin{pmatrix} 3 & 2.99999999 \\ 2.99999999 & 2.99999998 \end{pmatrix}$ whose rank is 1, eigenvalues $\begin{pmatrix} 0 \\ 5.99999998 \end{pmatrix}$, Cholesky factorization is not available because $A^T A$ is not positive definite. However, the Cholesky factor $\begin{pmatrix} 1.732050807568877 & \\ 1.732050801795375 & 0.000000008164966 \end{pmatrix}$ can still be calculated using QR decomposition on matrix A.

Theorem 9. *If $A \in \Re^{m \times n}$ has full column rank, $QR = A$ is the QR decomposition of matrix A, matrix $R \in \Re^{m \times n}$ is a upper triangular matrix with positive diagonal* *entries,* $R_1^{n\times n}$ *and* $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$, *then the Cholesky factorization of the normal matrix* $C = A^TA$ *is* $C = LL'$, *where* $L = R_1^T$.

Proof. We have $A = QR = Q\begin{pmatrix} R_1 \\ 0 \end{pmatrix}$. Then $C = A^TA = (QR)^TQR = R^TQ^TQR = R^TR = \begin{pmatrix} R_1^T & 0 \end{pmatrix}\begin{pmatrix} R_1 \\ 0 \end{pmatrix} = R_1^TR_1 = LL^T$

Because Cholesky factorization is unique and matrix L has positive diagonal entries, $LL^T = A^TA$ is a Cholesky factorization.

□

Therefore, we proposed a hybrid approach as follows:

Step 1. Find QR decomposition of matrix A_p, convert R to the Cholesky factor L.

Step 2. When a column is added to matrix A_p, use the row addition approach for Cholesky factorization to update L.

Step 3. When a column is removed from matrix A_p, use either the row deletion approach for QR decomposition or Cholesky factorization approach to update.

When the Cholesky factor L is available, solve the semi-normal equation:

$$LL^Tx = A^Tb \tag{3.2.19}$$

According to numerical analysis [25], solutions computed by semi-normal equation are not better than these computed by the corresponding normal equation. However, we can extend it to *Corrected Normal Equations* to achieve a higher quality solution, if one-step of iterative refinement is added.

In the next chapter, we use the hybrid approach to solve linear programming problems, but no correction is needed because these problems are well conditioned.

We keep Corrected Normal Equations as our future work for ill-conditioned problems if encountered in practice.

CHAPTER IV

COMBINED OBJECTIVE LEAST SQUARES

Degeneracy is a phenomenon often encountered in solving linear programming problems using simplex methods. It can unfavorably influence its efficiency and convergence. Various techniques have been proposed to avoid stalls due to degeneracy. In this chapter, we focus on a simplex-like method based on least squares, i.e. the combined objective least squares method (COLS) that is completely impervious to degeneracy. The COLS method is an extension of the NNLS method introduced in the previous chapter and they share many similar features.

In this chapter, we first introduce the basic theory for the combined objective least squares problem (COLS) and its solution methods. The COLS problem is similar to the NNLS problem and most of the technology we use for NNLS can be applied directly to COLS, including use of active set methods, QR decomposition, Cholesky factorization, adding and deleting columns and re-factorization. Then, we highlight the difference between COLS and NNLS, their respective solution methods and the challenges of COLS solution methods. Finally, we present some important properties for the COLS method and numerical results.

4.1 Combined Objective Least Squares

The combined objective least squares problem (COLS) is an extension of the nonnegative least squares problems (NNLS).

Suppose we have a feasible solution to the NNLS problem and the feasible solution is not unique, we want to find a solution x^*, which satisfies the feasibility condition and minimizes $c^T x$, where $c \in \Re^n$, then this problem is equivalent to the linear programming formulation:

$$\min c^T x \tag{4.1.1}$$

Subject to:

$$Ax = b \tag{4.1.2}$$

$$x \geq 0 \tag{4.1.3}$$

where $A \in \Re^{m \times n}$, $b \in \Re^m$, $c \in \Re^n$ and $x \in \Re^n$ that are the decision variables.

We use the least squares approach for the above problem and we have the following Combined Objective Least Squares problem:

$$\min \frac{M}{2} \|b - Ax\|^2 + c^T x \tag{4.1.4}$$

Subject to:

$$x \geq 0 \tag{4.1.5}$$

where $A \in \Re^{m \times n}$, $b \in \Re^m$, $c \in \Re^n$, $M \in \Re$ and $x \in \Re^n$ which is the decision variables.

If $M < +\infty$ is sufficiently large and x^* is the optimal solution for the problem (4.1.4)-(4.1.5), then $\|b - Ax^*\|^2$ will be close to 0 and x^* is sufficiently close to the feasible region, thus this can be a feasible and optimal solution for the linear programming problem (4.1.1)-(4.1.3).

4.2 Solution Methods for COLS

In this section, we introduce the active set method for COLS. For the active set of columns in matrix A, we relax the nonnegative constraints and solve it as an unconstrained COLS problem. If the solution is nonnegative, we have the solution to the active set of columns. Otherwise, negative columns will be removed from the active set and the resulting unconstrained COLS problem will be solved. We repeat the above procedure until a solution for the active set columns is found, which is the optimal solution for the COLS problem (4.1.4)-(4.1.5). This procedure is very similar to the corresponding active set algorithm for NNLS introduced in the previous chapter as well as in an additional reference [21]. The only difference between two algorithms is that active set methods for COLS solves an unconstrained COLS problem as a subproblem instead of a least square problem in active set methods for NNLS.

4.2.1 Unconstrained COLS

For the active set of columns in matrix A, we need to solve an Unconstrained Combined Objective Least Squares problem, which is defined as follows:

$$\min \frac{M}{2}\|b - Ex_E\|^2 + c_E^T x_E \tag{4.2.1}$$

where $E \in \Re^{m\times k}$, $b \in \Re^m$, $c_E \in \Re^k$, $M \in \Re$ and $x_E \in \Re^k$ that are the decision variables. Matrix E is a submatrix of matrix A with k active independent columns in A. c_E and x_E are the corresponding sub-vectors of the vector c and x.

Let $f(x_E) = \frac{M}{2}\|b - Ex_E\|^2 + c_E^T x_E$, its derivatives are $\nabla f(x_E) = M \times E^T(Ex_E - b) + c_E^T$ and its second order derivatives are $\nabla^2 f(x_E) = M \times E^T E$.

$f(x_E)$ is a convex function, because the vector norm is convex, the linear function is convex and positive weighted sum of convex functions is convex.

For convex functions, we have the following **Necessary Conditions for Optimality**:

1. $\nabla f(x_E^*) = 0$.

2. $\nabla^2 f(x_E^*)$ positive semi-definite.

The second condition will always be satisfied as $\nabla^2 f(x_E) = M \times E^T E \succ 0$, because $M > 0$ and $E^T E$ will always be positive definite for any matrix E with a full column rank. For any vector $x_E \in \Re^k$, we have $\nabla^2 f(x_E)$ as positive definite.

For the first condition, we have

$$\nabla f(x_E) = c_E^T + ME^T(Ex_E - b) = 0 \tag{4.2.2}$$

Rearrange it and we have normal equations for COLS problems:

$$E^T E x_E = E^T b - \frac{c_E^T}{M} \tag{4.2.3}$$

Theorem 10. *Let matrix $A \in \Re^{m\times n}$ with full column rank and vector $b \in \Re^m$, a vector $x \in \Re^n$ solves the unconstrained combined objective least squares problem $\min \frac{M}{2}\|b - Ax\|^2 + c^T x$, if*

$$A^T A x = A^T b + \frac{c^T}{M} \tag{4.2.4}$$

Proof. As $A^T A$ is positive definite, $\nabla^2 f(x) \succ 0$. From $A^T Ax = A^T b + \frac{c^T}{M}$, we have $\nabla f(x) = 0$. Therefore, x is the unique optimal solution for $\min \frac{M}{2}\|b - Ax\|^2 + c^T x$

□

The normal equation for unconstrained COLS can be solved using Cholesky factorization, QR decomposition and singular value decomposition just as the case for least squares problems.

4.2.1.1 Unconstrained COLS via QR decomposition

Replace E with its QR decomposition, $E = QR$, where Q is orthogonal matrix, i.e. $QQ^T = I$ and R is upper-triangular matrix. We have:

$$c_E^T + MR^T Q^T Q R x_E - MR^T Q^T b = 0 \tag{4.2.5}$$

$$\Rightarrow \frac{1}{M} c_E^T = R^T Q^T b - R^T R x_E = R^T(Q^T b - R x_E) = R^T y \tag{4.2.6}$$

where $y = Q^T b - Rx_E$.

Solve the equations $\begin{cases} \frac{1}{M} c_E^T = R^T y \\ y = Q^T b - R x_E \end{cases}$ using back substitution, we find the optimal solution x_E^*.

The full algorithm for the unconstrained combined objective least squares problem is as follows:

Algorithm 4.1 Unconstrained COLS via QR decomposition

```
Require: $E$,$c_E$, $b$, $M$
Ensure: $x$
1: $QR \leftarrow qr(E)$
2: Solve $R^T y = \frac{1}{M} c_E^T$ for $y$ using back substitution.
3: Solve $Rx = Q^T b - y$ for $x$ using back substitution.
```

4.2.1.2 Unconstrained COLS via Cholesky factorization

Replace $E^T E$ with its Cholesky factorization $LL^T = E^T E$. We have:

$$LL^T x_E = Eb - \frac{c}{M} \Rightarrow Ly = E^T b - \frac{c}{M} \tag{4.2.7}$$

where $y = L^T x_E$.

Solve the equations $\begin{cases} Ly = E^T b - \frac{c}{M} \\ L^T x_E = y \end{cases}$ using back substitution, we find the optimal solution x_E^*.

The full algorithm for the unconstrained combined objective least squares problem is as follows:

Algorithm 4.2 Unconstrained COLS via Cholesky factorization

```
Require: $E$,$c_E$, $b$, $M$
Ensure: $x$
1: $LL^T \leftarrow chol(E^T E)$
2: Solve $Ly = E^T b - \frac{c}{M}$ for $y$ using back substitution.
3: Solve $L^T x = y$ for $x$ using back substitution.
```

4.2.2 KKT Conditions for COLS

Theorem 11 specifies the optimality solution for problem COLS.

Theorem 11. *A vector $\hat{x}$ is a solution for the COLS problem* $\min c'x + \frac{M}{2}\|Ax - f\|^2$ *subject to $x \geq 0$ if and only if there exists $\hat{w} \in \Re^m$ and a partitioning of the integers 1 through m into subsets P and Z such that*

1. *$\hat{w} = A^T(A\hat{x} - b) + \frac{c}{M}$*

2. *$\hat{x}_i = 0$, where $i \in Z$; $\hat{x}_i > 0$, where $i \in P$;*

3. *$\hat{w}_i \geq 0$, where $i \in Z$; $\hat{w}_i = 0$, where $i \in P$;*

Proof. See [15] and [18].

□

4.2.3 Active Set Methods for COLS

Similar to the NNLS problem, we also use active set methods to solve the COLS problem: partition the set of columns in matrix A into set P and set Z, where set P is the active set, $P \bigcup Z = A$ and $P \bigcap Z = \emptyset$. Form the unconstrained COLS problem by matrix A_p which consists of columns in set P. If some variables do not satisfy $x_p^* > 0$, move the columns with non-positive values from set P to set N.

Otherwise, pick a column satisfying conditions ($\hat{w}_i < 0$) from set Z and move it to P. If we cannot find such a column, then it is the optimal solution for the COLS problem.

We are given $A \in \Re^{m\times n}$, $c \in \Re^n$, $b \in \Re^m$, $M \in \Re^+$. P and Z are index sets. Variables indexed in the set Z will be set to value zero.

On termination x will be the solution vector, w will be the dual vector and r will be the residual multiplied by M. Note that r can be used in primal dual subproblem approaches.

Algorithm 4.3 Combined Objective Least Squares (COLS)

```
Require: $A$,$c$, $b$, $M$
Ensure: $x$, $w$, $r$
 1: $P = \emptyset$, $Z = \{1, 2, ..., n\}$, $x = 0$
 2: while TRUE do
 3:   $w \leftarrow A^T(b - Ax) - \frac{1}{M}c$
 4:   if $Z = \emptyset$ or $w_j \leq 0, \forall j \in Z$ then
 5:     $r = M(b - Ax)$, stop, return $x$ and $r$
 6:   end if
 7:   Find an index $t \in Z$ such that $w_t = \max\{w_j : j \in Z\}$.
 8:   Move the index $t$ from set $Z$ to set $P$.
 9:   while TRUE do
10:     Let $E$ denote the $m_2 \times n$ matrix defined by
11:     Column $j$ of $E \leftarrow \begin{cases} \text{column } j \text{ of } A, & \text{if } j \in P; \\ 0, & \text{if } j \in Z. \end{cases}$.
12:     Define $c_E$ as the elements in $c$ corresponds to $E$
13:     Compute the $n$-vector $z$ as a solution of the unconstrained least squares
        problem $\min c_E^T x + \frac{M}{2}\|Ex - b\|^2$. Note that only the components $z_j$, $j \in P$
        are determined. Define $z_j = 0$ for $j \in Z$.
14:     if $z_j > 0, \forall j \in P$ then
15:       $x \leftarrow z$, Break; //break out of the inner loop
16:     else
17:       Find an index $q \in P$ such that $\frac{x_q}{x_q - z_q} = \min\{\frac{x_j}{x_j - z_j} : z_j \leq 0, j \in P\}$
18:       $\alpha \leftarrow \frac{x_q}{x_q - z_q}$, $x \leftarrow x + \alpha(z - x)$
19:       Move from set $P$ to set $Z$ all indices $j \in P$ for which $x_j = 0$
20:     end if
21:   end while
22: end while
```

4.2.4 The Correctness of Algorithm 4.3

After algorithm 4.3 terminates, we have a partition of set P and set Z, and matrix E that consists of the columns in set P. $x_E \geq 0$ and $x_Z = 0$. Therefore, condition 2 in the KKT condition is satisfied.

On termination, we have $\frac{c}{M} - A^T r = A^T(A\hat{x} - b) + \frac{c}{M} \geq 0$ for columns in set Z. As for columns in set P, we have $E^T(Ex_E - b)M + c_E^T = 0$, which is the normal equation to calculate x_E in the last iteration. Conditions 1 and 3 in the KKT conditions are satisfied.

4.2.5 Dynamic M in COLS.

The value of M plays an important role in the convergence and quality of the solution: if M is relatively small, we can find the optimal solution in fewer number of iterations, but the residual norm will be relatively large, i.e. the solution is feasible for the original linear programming problem with relatively larger errors.

Here, we give a simple example. Suppose the linear program is as follows:

$$\min\{-100x_1 - 100x_2 : x_1 \leq 1, x_2 \leq 1, x_1 \geq 0, x_2 \geq 0\} \tag{4.2.8}$$

The optimal solution for the above problem is $x_1^* = x_2^* = 1$ and the objective value is -200.

If we use COLS and set $M = 1000$, the optimal solution will be $x_1 = x_2 = 1.01$, $\rho = \begin{pmatrix} 0.01 \\ 0.01 \end{pmatrix}$, $||b - Ax||^2 = 0.0002$ and the objective is $-202 + 1000 * 0.0002 = -201.8 < Obj_{LP}^*$.

If $M = 100$, the optimal solution will be $x_1 = x_2 = 1.1$, $\rho = \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}$, $||b - Ax||^2 = 0.02$ and the objective is $-220 + 1000/2 * 0.02 = -210 < Obj_{LP}^*$.

Clearly, if M is not large enough, the COLS solution is not a feasible solution for the corresponding linear programming problem. However, M cannot be too big as well. Suppose $M = +\infty$, the COLS solution will be the same as the NNLS solution, which is a feasible solution for the corresponding LP, but may not necessarily be optimal: $\lim_{M\to+\infty}\{\frac{1}{2}\|b - Ax\|^2 + \frac{c^T}{M}x\} = \frac{1}{2}\|b - Ax\|^2$.

Figure 4.1 shows the relationship between different values of the penalty M and its relationship to the simplex method when $M \to +\infty$.

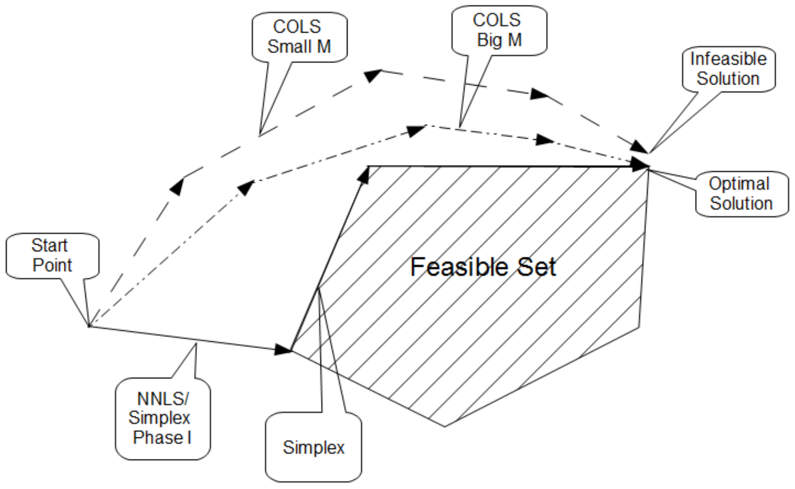


Figure 4.1: M values and the simplex method.

We found that the COLS methods with different penalty M always traversed outside the feasible region and the larger the M, the closer the trajectory was to the boundary of the feasible region.

4.3 An Augmented System Approach for COLS

In addition to QR decomposition and Cholesky factorization approaches, the unconstrained COLS min $\frac{M}{2}\|b - Bx\|^2 + c_B x$, where B is the basis in the active set method, can also be solved using the augmented system approach.

Let $W = \begin{pmatrix} I & B \\ B^T & 0 \end{pmatrix}$, which is nonsingular if matrix B has full column rank. The augmented system matrix W is a symmetric indefinite matrix, as introduced in the previous chapter.

Given matrix $B \in \Re^{m\times n}$, $b \in \Re^m$, we can use a LU decomposition approach to solve the unconstrained COLS problem $\min \frac{M}{2}\|b - Bx\|^2 + cx$.

Let $W \in \Re^{(m+n)\times(m+n)}$, where $W = \begin{pmatrix} I & B \\ B^T & 0 \end{pmatrix}$. The unconstrained COLS problem can be solved by

$$W\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} I & B \\ B^T & 0 \end{pmatrix}\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ \frac{c_B}{M} \end{pmatrix}$$

where $x \in \Re^m$ and $r \in \Re^n$ are the solution of the unconstrained COLS problem and $r = b - Bx$ is the residual.

Theorem 12. *The unconstrained COLS problem* $\min \frac{M}{2}\|b - Bx\|^2 + c_B x$ *is solved by*

$$\begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} I & B \\ B^T & 0 \end{pmatrix}^{-1}\begin{pmatrix} b \\ \frac{c_B}{M} \end{pmatrix}$$

Proof. We can rewrite the equation as $\begin{cases} Bx + r = b \\ B^T r = \frac{c_B}{M} \end{cases}$. When we rearrange it we have $r = b - Bx$. When we plug it into the second equation, we have $B^T(b - Bx) = \frac{c_B}{M}$, which is the normal equation for the unconstrained COLS problem.

□

As W is nonsingular, we have $W = LU$, where $L \in \Re^{(m+n)\times(m+n)}$ and lower triangular, and $U \in \Re^{(m+n)\times(m+n)}$ and upper triangular. We can solve for y using back substitution on $Ly = \begin{pmatrix} b \\ \frac{c_B}{M} \end{pmatrix}$, where $y = U\begin{pmatrix} r \\ x \end{pmatrix}$. Then, we can solve for $\begin{pmatrix} r \\ x \end{pmatrix}$ using back substitution.

4.4 Numerical Experiments

We implement a version of the COLS algorithm using Cholesky factorization and have the following computational results.

Table 4.1: Detailed computational results using COLS

Problem	set partitioning
Rows	350
Columns	3,302,596
Big M	$10^3 \sim 10^{10}$
Total COLS Iterations	11432
Total CPU Time	74.13
CPU Time breakdown	
Total Cholesky factorization time	59.225 seconds
Total back substitution time	1.417 seconds
Total Cholesky factorization time	59.225 seconds
Total time to calculate the reduced cost $w = r^T A - \frac{c}{M}$	5.979 seconds
Total time to find max w	1.16 seconds
Total time to check dependent	0.871 seconds

4.5 Numerical Results for Combined Objective Least Squares

We implemented three variants of Combined Objective Least Squares (COLS) in MATLAB, with Cholesky factorization and QR decomposition from the SuiteSparse package developed by Tim Davis. The variants are QR decomposition, normal equations via Cholesky factorization and the hybrid approach, in which the Cholesky factor is calculated by QR decomposition, column addition via Cholesky factorization's column addition algorithm and column deletion via QR decomposition. In all COLS variants, matrix decompositions are updated using column addition and column deletion approaches, with a partial pricing scheme based on the primal dual simplex method.

Table 4.2 provides the comparison between COLS Gurobi Version 4.6.0 simplex solvers. Note that Gurobi presolve took 67.38 seconds to reduce the problem size to 217 rows and 4,655,832 columns.

Table 4.3 provides computational details of three COLS variants. We found that the convergence rate and majority of computational efforts are similar, except that the QR decomposition approach spent significant time to multiply the orthogonal matrix Q to the right hand side and the incoming column when a column is added

Table 4.2: Computational results for set partitioning problem RJmax

RJmax with 219 rows 5,091,554 columns	Number of Iterations	Final Objective	Residual Norm	Run Time (sec)
COLS via QR	17898	2.2958e+04	3.0443e-010	323.8425
COLS via Cholesky	16428	2.2958e+04	3.0443e-010	207.3097
COLS hybrid	16950	2.2958e+04	3.0443e-010	216.7322
Gurobi with presolve	253442	2.2958e+04	N/A	307.56
Gurobi without presolve	60598	2.2958e+04	N/A	1710.00

into the basis. The Cholesky factorization approach and the hybrid approach do not need this computational burden, thus it is much faster overall. The hybrid approach is slightly slower than the Cholesky factorization approach, but it is more stable in theory.

Table 4.3: Computational results for COLS variants

Problem with 219 rows 5,091,554 columns	COLS via QR	COLS via Cholesky	COLS Hybrid
Total factorization time (sec)	15.8341	11.6377	13.4317
Total update time (sec)	87.2826	76.5341	79.5605
Total time to multiply Q^T (sec)	91.2606	N/A	N/A
Total back substitution time (sec)	54.9280	56.6284	55.6924
Total time to calculate reduced costs (sec)	31.0130	30.7634	32.0270
Total time to find max reduced cost (sec)	4.3212	4.7268	4.3056
Total time to check dependence (sec)	20.7637	23.3689	23.4782
Total outer iterations	17898	16428	16950
Total inner iterations	4801	4380	4538
Total run time (sec)	323.8425	207.3097	216.7322

Because the MATLAB implementations is slower than the corresponding C/C++ versions, especially when matrices are copying between the MATLAB environment and the SuiteSparse solvers, we also implement a variant of the Combined Objective Least Squares (COLS) in C++, with Cholesky factorization and QR decomposition from the SuiteSparse package developed by Tim Davis. The variant is the hybrid approach, in which the Cholesky factor is calculated by QR decomposition, column addition via Cholesky factorization's column addition algorithm and column deletion via QR decomposition. Similar to the MATLAB implementation, matrix decomposition is updated using column addition and column deletion approaches, with a partial pricing scheme based on the primal dual simplex method.

Additionally, we notice that dual simplex methods perform better than primal simplex methods in Gurobi; therefore, we also include computational results using different variants of Gurobi. Tables 4.4 and 4.5 compare the COLS performance with Gurobi variants and outperforms all. We notice that the dual simplex method without presolve performs better than the dual simplex method with presolve, but the reverse is true for Gurobi primal simplex methods.

Table 4.4: COLS vs Gurobi variants for problem RJMax

Problem RJMax with 219 rows 5,091,554 columns	Objective	Run Time (sec)
COLS	22958	12.983
Gurobi Primal with Presolve	22958	307.56
Gurobi Primal without Presolve	22958	1710.0
Gurobi Dual with Presolve	22958	105.43
Gurobi Dual without Presolve	22958	35.92

Table 4.5: COLS vs Gurobi variants for problem RJMod

Problem RJMod with 212 rows 5,052,622 columns	Objective	Run Time (sec)
COLS	21090.9	11.172
Gurobi Primal with Presolve	21090.9	1499.10
Gurobi Primal without Presolve	21090.9	821.18
Gurobi Dual with Presolve	21090.9	95.38
Gurobi Dual without Presolve	21090.9	28.00

CHAPTER V

THE LONG STEP PRIMAL DUAL SIMPLEX METHOD

In primal dual simplex methods, the step size is calculated deterministically as a function of the dual feasible solution and the dual solution from the restricted master problem to maintain dual feasibility. The resulting dual objective value is the previous dual objective values plus the step size times the objective value of the restricted master problem [23].

For linear programming problems with convexity constraints, it is possible to manipulate the dual values so that the step size θ can be pushed to maximize the dual objective value and still preserve dual feasibility. We can set the step size to an arbitrary value and still maintain dual feasibility by adjusting the dual variables associated with convexity constraints.

Convexity constraints occur very often in airline and transportation optimization problems. For example, in multicommodity flow problems, we have a convexity constraint for each commodity. In airline rostering problems, we have crew convexity constraints so that each crew will be assigned exactly one roster. In airline crew pairing problems, we need convexity constraints for each crew base so that the pairings assigned to each crew base do not exceed its flying capacity limit.

In this chapter, we introduce features essential to the long step primal dual simplex method and some technical details. In section 5.1, linear programming problems with convexity constraint will be defined. In section 5.2, the primal dual simplex method tailored to the linear programming problem with convexity constraints is reinstated and the formula to calculate the step size is presented. In section 5.3, we introduce the primal dual simplex method phase II to find global optimal solutions. In section 5.4, we introduce the long step primal dual simplex method. We study the dual objective value as a function of the step size and introduce a polynomial time algorithm to find the optimal step size.

5.1 Linear Programming Problems with Convexity Constraints

Considering the linear programming problem in standard form:

$$\min c^T x \tag{5.1.1}$$

Subject to:

$$Ax = b \tag{5.1.2}$$

$$x \geq 0 \tag{5.1.3}$$

where $A \in \Re^{m\times n}$, $c \in \Re^n$, $b \in \Re^m$ and $x \in \Re^n$.

If we add a set of convexity constraints $Dx = u$ to the standard form problem, we will get the linear programming problem with convexity constraints. $D \in \Re^{K\times n}$, $u \in \Re^K$ and $u > 0$. In each column of matrix D, there is exactly one nonnegative element.

We give an example of a set of convexity constraints as follows:

$$\begin{pmatrix} \times & 0 & 0 & \times & 0 \\ 0 & \times & \times & 0 & 0 \\ 0 & 0 & 0 & 0 & \times \end{pmatrix} x = \begin{pmatrix} \times \\ \times \\ \times \end{pmatrix} \tag{5.1.4}$$

Note that $\times$ represents nonzero elements. The convexity matrix in the example has three rows: the first has two nonzero elements, the second has two nonzero elements and the third has one nonzero element. Nonzero elements in each row are the same and each column has exactly one nonzero element. We do not allow negative right hand sides, because the dual objective value will not be a concave function if $u < 0$.

If a right hand side is negative, we multiple both sides with -1 to make it positive. If a right hand side is 0, then all columns in the corresponding block will not be allowed in the final solution. We can preprocess the data to remove those columns and the corresponding zero on the right hand side. For example, if the first element in the right hand side is 0, it will prevent selection of columns 1 and 4, which is equivalent to adding the constraints $x_1 = 0$ and $x_4 = 0$.

We rearrange the convexity matrix to make it a block-diagonal matrix with A, c, x changing accordingly:

$$\begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & 0 & \times \end{pmatrix} x = \begin{pmatrix} \times \\ \times \\ \times \end{pmatrix} \tag{5.1.5}$$

We have the standard form problem with convexity constraints as follows:

$$\min c^T x \tag{5.1.6}$$

Subject to:

$$Ax = b \tag{5.1.7}$$

$$Dx = u \tag{5.1.8}$$

$$x \geq 0 \tag{5.1.9}$$

where D is a $|K| \times n$ block diagonal matrix with exactly one nonzero element in each column, $u > 0$ and $u \in \Re^K$.

Convexity constraints divide the standard form problem into $|K|$ blocks. We can rewrite the standard form problem with convexity constraints as follows:

$$\min \sum_{k \in K} c^k x^k \tag{5.1.10}$$

Subject to:

$$\sum_{k \in K} A^k x^k = b \tag{5.1.11}$$

$$D^k x^k = u_k, \forall k \in K \tag{5.1.12}$$

$$x^k \geq 0, \forall k \in K \tag{5.1.13}$$

where $A^k \in \Re^{n_k \times m}$, $D^k \in \Re^{n_k}$ matrix with all entries being nonzero elements, $x^k \in \Re^{n_k}$, $b \in \Re^m$, $u > 0$ and $u \in \Re^K$.

The number of rows in Equation (5.1.11) is m, the number of rows in Equation (5.1.12) is $|K|$ and the number of columns for block k is n_k, where $k \in K$. The linear programming problem with convexity constraints has $m + |K|$ rows and $n = \sum_{k \in K} n_k$ columns.

5.1.1 The Dual Problem

The dual of the linear programming problem with convexity constraints (5.1.10) - (5.1.13) is:

$$\max \pi b + \sigma u \tag{5.1.14}$$

Subject to:

$$c - \pi A - \sigma D \geq 0 \tag{5.1.15}$$

where $\pi \in \Re^m$ is the dual variables corresponds to Equation (5.1.11) and $\sigma \in \Re^K$ is the dual variables corresponds to Equation (5.1.12).

5.1.2 Optimality Conditions

We extend the complementary slackness conditions to linear programming problems with convexity constraints.

Theorem 13. *(Complementary Slackness) If x is feasible in the primal problem and (π, σ) is feasible in the dual problem, a necessary and sufficient condition for both to be optimal is:*

$$\begin{aligned} \pi_i(a_i^T x - b_i) &= 0, \forall i && (5.1.16)\\ \sigma_k(d_k^T x - u_k) &= 0, \forall k && (5.1.17)\\ (c_j - \pi^T A_j - \sigma_k D_j)x_j &= 0, \forall j && (5.1.18) \end{aligned}$$

Proof. From the definition of the dual problem, π_i has the same sign of $a_i^T x - b_i$, σ_k has the same sign as $d_k^T x - u_k$ and $c_j - \pi^T A_j - \sigma_k D_j$ has the same sign of x_j, thus

$$\begin{aligned} \pi_i(a_i^T x - b_i) &\geq 0, \forall i && (5.1.19)\\ \sigma_k(d_k^T x - u_k) &\geq 0, \forall k && (5.1.20)\\ (c_j - \pi^T A_j - \sigma_k D_j)x_j &\geq 0, \forall j && (5.1.21) \end{aligned}$$

Adding the left hand side of all three equations together:

$$0 = \sum_{\forall i} \pi_i(a_i^T x - b_i) + \sum_{\forall k} \sigma_k(d_k^T x - u_k) + \sum_{\forall j}(c_j - \pi^T A_j - \sigma_k D_j)x_j \qquad (5.1.22)$$

$$= c^T x - \pi b - \sigma u \qquad (5.1.23)$$

Thus, $c^T x = \pi b + \sigma u$, where the left hand side $c^T x$ is the primal objective and the right hand side $\pi b + \sigma u$ is the dual objective. The dual objective equaling the primal objective is the necessary and sufficient condition for optimality.

□

5.2 The Primal Dual Simplex Method

The primal dual simplex method is a dual ascend method. We start with a dual feasible solution and maintain dual feasibility throughout the algorithm until a corresponding primal feasible solution is found. In this section, we reiterate the primal dual simplex method tailored to the linear programming problem with convexity constraints.

Given an initial dual feasible solution (π, σ), we can define the *admissible set* J, which is a set of all admissible columns. $J = \{i | c_i - \pi A_i - \sigma_k \leq \varepsilon, \forall A_i \in A^k, \forall k \in K\}$, where ε is the threshold for the admissible set J. If we set ε to 0, it is the classical primal dual simplex method. Otherwise, it is called the primal dual subproblem simplex method [17].

If we set all decision variables not in the admissible set to 0, we have a *Restricted Master Problem* (RMP):

$$\min \sum_{j=1}^{n} y_j + \sum_{k=1}^{K} z_k \tag{5.2.1}$$

Subject to:

$$\sum_{k \in K} A^k x^k + y = b \tag{5.2.2}$$

$$D^k x^k + z = u_k, \forall k \in K \tag{5.2.3}$$

$$x^k \geq 0, \forall k \in K \tag{5.2.4}$$

$$x_i = 0, \forall i \notin J \tag{5.2.5}$$

$$y \geq 0, z \geq 0 \tag{5.2.6}$$

where:

$y \in \Re^m$**:** the slack variables for constraints (5.2.2);

$z \in \Re^K$**:** the slack variables for constraints (5.2.3).

The RMP is a phase I type problem in linear programming with slack variables y and z. All columns $i \notin J$ are eliminated because their corresponding decision variables are zeros.

If the objective value of the RMP was 0, then we found a feasible primal solution using only the admissible set J. According to the complementary slackness conditions, a primal and dual feasible pair was found, thus it was optimal. This is applicable only if the threshold for the admissible set is 0. Otherwise, some complementary slackness conditions may still be violated if the threshold $\varepsilon > 0$.

If the objective of the RMP is positive, i.e. $Z^* > 0$, then we will get the dual values ρ and τ for constraints (5.2.2) and (5.2.3) in the restricted master problem. We have $Z^* = \sum_{i=1}^{m} b_i \rho_i + \sum_{k=1}^{K} u_k \tau_k > 0$. For a positive θ, the dual objective will be improved by $\theta \times Z^* > 0$.

Figure 5.1 illustrates the relationship between the current dual solution (π, σ), the dual solution (ρ, τ) from the restricted master problem and the updated dual feasible solution $(\pi, \sigma) + \theta(\rho, \tau)$.

We update the dual feasible solution by $(\pi, \sigma) = (\pi, \sigma) + \theta(\rho, \tau)$, where θ is the step size, which can be calculated as

$$\theta = \min_{j \notin J, \rho A_j + \tau D_j > 0} \frac{c_j - \pi A_j - \sigma D_j}{\rho A_j + \tau D_j} \tag{5.2.7}$$

This formula is derived to get the biggest possible θ while still maintaining dual feasibility. Given the original dual solution (π, σ) and the dual solution from the RMP (ρ, τ), for each column $i \notin J$, we have $c_i - \pi A_i - \sigma D_i > \varepsilon$. Clearly, if $\rho A_i + \tau D_i \leq 0$, then the step size θ is not constrained by column i, we can set θ arbitrarily large while still maintaining dual feasibility.

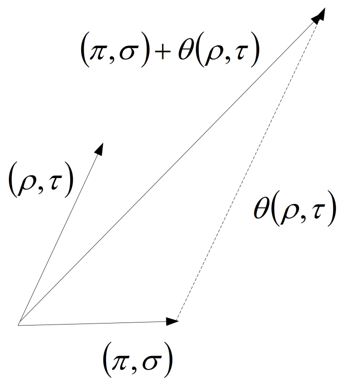


Figure 5.1: Primal Dual Simplex Method Step Size.

So we need only consider the case $\rho A_i + \tau D_i > 0$. In order to maintain dual feasibility, we must have $c_i - (\pi + \theta\rho)A_i - (\sigma + \theta\tau)D_i \geq 0$. Rearrange it, we have:

$$\theta \leq \frac{c_i - \pi A_i - \sigma D_i}{\rho A_i + \tau D_i} \tag{5.2.8}$$

Therefore, the step size θ can be derived by taking the minimum of all columns not in the admissible set J and $\rho A_j + \tau D_j > 0$. If all columns have $\rho A_j + \tau D_j \leq 0$, we can set θ to $+\infty$. As a result, the dual objective is $+\infty$ and the primal problem is infeasible.

Using the revised dual feasible solution, we update the admissible set J, construct a new restricted master problem and repeat this procedure until a primal and dual feasible pair is found. The full algorithm is illustrated in Algorithm 5.1.

Algorithm 5.1 The Primal Dual Simplex Method

```
Require: $A$,$D$, $b$, $u$, $\pi$, $\sigma$, $\varepsilon$
Ensure: $\pi^*, \sigma^*$
 1: while TRUE do
 2:     $J \leftarrow \{j : c_j - \pi A_j - \sigma D_j \leq \varepsilon\}$
 3:     Setup and solve the Restricted Master Problem, get the objective value $Z^*$ and
        dual solution $(\rho, \tau)$.
 4:     if $Z^* = 0$ then
 5:         Optimal, STOP;
 6:     else
 7:         if $\rho A + \tau D < 0$ then
 8:             Infeasible, STOP.
 9:         else
10:             $\theta = \min_{j \notin J, \rho A_j + \tau D_j > 0} \frac{c_j - \pi A_j - \sigma D_j}{\rho A_j + \tau D_j}$
11:             $\pi_\theta \leftarrow \pi + \theta \rho$
12:         end if
13:     end if
14: end while
```

5.3 The Primal Dual Subproblem Simplex Phase II

If the threshold ε to define the admissible set J in primal dual simplex methods is positive, the admissible set J may contain columns with positive reduced costs. If such columns remain in the optimal basis of the final restricted master problem on convergence, the resulting solution might not be global optimal, because some of the complementary slackness conditions may still be violated, i.e. $x_j > 0$, $c_j - \pi A_j - \sigma D_j > 0$ and $(c_j - \pi A_j - \sigma D_j)x_j \neq 0$.

Hu [16] proposed a phase II primal dual subproblem simplex method starting from feasible primal and dual solutions: given an initial dual feasible solution (π, σ), we can define the *admissible set* J, which is a set of all admissible columns. $J = \{i | c_i - \pi A_i - \sigma_k \leq \varepsilon, \forall A_i \in A^k, \forall k \in K\}$, where $\varepsilon > 0$ is the threshold for the admissible set J.

Set all decision variables not in the admissible set to 0, we have a *Restricted Master Problem* (RMP):

$$\min \sum_{i=1}^{n} c_i x_j \tag{5.3.1}$$

Subject to:

$$\sum_{k \in K} A^k x^k = b \tag{5.3.2}$$

$$D^k x^k = u_k, \forall k \in K \tag{5.3.3}$$

$$x^k \geq 0, \forall k \in K \tag{5.3.4}$$

$$x_i = 0, \forall i \notin J \tag{5.3.5}$$

The RMP is a phase II type problem in linear programming. All columns $i \notin J$ can be eliminated from the problem because the corresponding decision variables are zeros. Because no slack variables are presented in the formulation, we must start from a solution that is both primal and dual feasible. The primal dual subproblem simplex method introduced in the previous section provides such a solution on its convergence.

We solve the restricted master problem and get the optimal dual solution (ρ, τ). Because it is a phase II type problem, its objective might not be 0. To check global optimality, we calculate the reduced cost for all columns, i.e. $\bar{c}_{(\rho,\tau)} = c - \rho A - \tau D$. If $\bar{c}_{(\rho,\tau)} \geq 0$, we have achieved global optimality.

Otherwise we update the dual feasible solution by $(\pi, \sigma) = (1-\theta)(\pi, \sigma) + \theta(\rho, \tau)$, where $0 \leq \theta \leq 1$ is the step size, which can be calculated as

$$\theta = \min_{j \notin J, \bar{\rho} A_j + \bar{\tau} D_j > 0} \frac{c_j - \pi A_j - \sigma D_j}{\bar{\rho} A_j + \bar{\tau} D_j} = \min_{j \notin J, \rho A_j + \tau D_j - \pi A_j - \sigma D_j > 0} \frac{c_j - \pi A_j - \sigma D_j}{\rho A_j + \tau D_j - \pi A_j - \sigma D_j} \tag{5.3.6}$$

This formula is derived to get the biggest possible θ while still maintaining dual feasibility. Figure 5.2 and 5.3 illustrate a geometrical view of above equivalent equations, i.e. $\pi = (1-\theta)(\pi, \sigma) + \theta(\rho, \tau)$ and $\pi = (\pi, \sigma) + \theta((\rho, \tau) - (\pi, \sigma))$, which produces exactly the same step size, and updates the dual feasible solution $(1-\theta)(\pi, \sigma) + \theta(\rho, \tau)$.

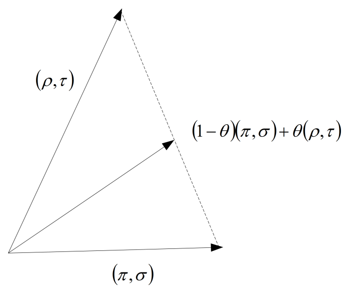


Figure 5.2: Primal Dual Simplex Method Phase II Step Size.

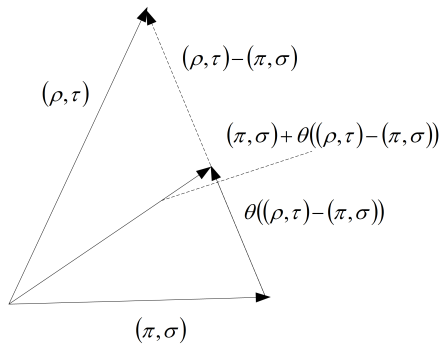

$(\rho,\tau)-(\pi,\sigma)$
(ρ,τ)
$(\pi,\sigma)+\theta((\rho,\tau)-(\pi,\sigma))$
$\theta((\rho,\tau)-(\pi,\sigma))$
(π,σ)

Figure 5.3: Rearranged Step Size.

Using the updated dual feasible solution, we update the admissible set J, construct a new restricted master problem and repeat this procedure until a dual solution for the RMP that is dual feasible for all columns is found. The full algorithm is illustrated in Algorithm 5.2. Note that this algorithm is slightly different from others that have been referenced[16] because of convexity constraints.

Algorithm 5.2 The Primal Dual Subproblem Simplex Method (Phase II)

```
Require: $A$,$D$, $b$, $u$, $\pi$, $\sigma$, $\varepsilon$
Ensure: $\pi^*, \sigma^*$
1: while TRUE do
2:     $J \leftarrow \{j : c_j - \pi A_j - \sigma D_j \le \varepsilon\}$
3:     Setup and solve the Restricted Master Problem, get the objective value $Z^*$ and
       dual solution $(\rho, \tau)$.
4:     if $\rho A + \tau D - \pi A - \sigma D \le 0$ then
5:         Infeasible, STOP.
6:     else
7:         $\theta = \min_{j \notin J, \rho A_j + \tau D_j - \pi A - \sigma D > 0} \frac{c_j - \pi A_j - \sigma D_j}{\rho A_j + \tau D_j - \pi A - \sigma D}$
8:         if $\theta = 1$ then
9:             Optimal, STOP.
10:        end if
11:        $\pi_\theta \leftarrow (1-\theta)\pi + \theta\rho$
12:        $\sigma_\theta \leftarrow (1-\theta)\sigma + \theta\tau$
13:    end if
14: end while
```

5.4 The Long Step Primal Dual Simplex Method

In traditional primal dual simplex methods, all dual prices are used together to calculate the step size as mentioned in the previous section. However, this approach can be improved for problems with convexity constraints so that larger dual objective improvements may be achieved.

The motivation to study the long step primal dual simplex method is to maximize the dual objective improvement given a RMP solution, because larger dual improvement in each iteration may lead to a fewer number of total iterations to find the optimal solution and thus improve the overall convergence rate.

Considering the block diagonal structure in the convexity constraints, for columns in the same block k, we only have one dual variable σ_k which is associated with the convexity constraint.

From the dual formulation (5.1.14)-(5.1.15), for any given π, we can find a corresponding σ_k for each block $k \in K$ to maintain dual feasibility. Unlike the primal dual simplex approach, we can set π and σ separately as long as dual feasibility constraint (5.1.15) is satisfied.

For example, we can pick any θ, the resulting dual is $\pi_\theta = \pi + \theta\rho$. For column i in block k, we have $\overline{c_i} = c_i - \pi_\theta A_i - \sigma_k \geq 0$, as long as $\sigma_k \leq c_i - \pi_\theta A_i = c_i - \pi A_i - \theta\rho A_i$.

For each block k, the dual variable for the convexity constraint can be determined by:

$$\sigma_k = \min_{i \in A^k} \{c_i - \pi A_i - \theta\rho A_i\} \tag{5.4.1}$$

Note that the updated dual values σ for the convexity constraints are independent of the original dual values σ and the dual variables τ from the restricted master problem.

Different pairs (π, σ) may result in different dual objectives. In the next section, we will investigate the relationship between the dual objective value and the step size θ, and we will find an optimal step size θ^* so that the dual objective is maximized.

5.4.1 The Dual Objective as a Function of the Step Size

Given a step size θ, we can find the corresponding dual solutions as follows: $\pi^\theta = \pi + \theta\rho$. For each block k,we set $\sigma_k^\theta = \min_{i \in A^k}\{c_i - \pi_\theta A_i\}$. The resulting dual solution $(\pi^\theta, \sigma^\theta)$ is still dual feasible.

The dual objective Z is a function of θ:

$$Z(\theta) = \pi_\theta b + \sum_{k \in K} \sigma_{\theta k} u_k \tag{5.4.2}$$

$$= \pi b + \theta \rho b + \sum_{k \in K} u_k \min(c_k - \pi A_k - \theta \rho A_k) \tag{5.4.3}$$

The first part of the function $\pi b + \theta \rho b$ is a linear function of θ, while the second part of the function $\sum_{k \in K} u_k \min(c_k - \pi A_k - \theta \rho A_k)$ may decrease as θ increases.

We study σ_k as a function of θ: it is a function of the form $\min_{i=n_k}(a_i - b_i\theta)$, where $a_i = c_i - \pi A_i$ and $b_i = \rho A_i$, which is a piecewise linear concave function, as illustrated in Figure 5.4. When the step size θ increases, $c_k - \pi A_k - \theta \rho A_k$ decreases and eventually becomes negative. In order to maintain dual feasibility, we need to set σ_k to a negative value to make the reduced cost be 0.

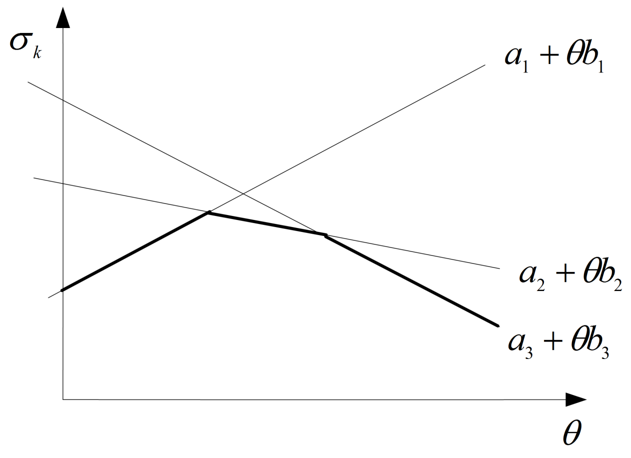


Figure 5.4: σ_k as a function of θ.

We have an alternative approach to derive the dual objective as a function of the step size. For all active columns in the admissible set J, we have $c_i - \rho A_i - \tau D_i \geq 0$, thus the step size can be an arbitrary value. Therefore, we only need to consider column j where $j \notin J$ and study the dual objective as a function of θ and j.

Suppose column j belongs to block k, we have $\sigma_k = c_j - \pi A_j - \theta \rho A_j$ and the dual objective $Z(\theta, j) = (\pi + \theta\rho)b + u_k\sigma_k = \pi b + \theta\rho b + c_j - \pi A_j - \theta\rho A_j = a_1 + a_2\theta$, where $a_1 = \pi b + c_j - \pi A_j = c_j + \pi(b - A_j)$ and $a_2 = \rho b - \rho A_j = \rho(b - A_j)$. Therefore, the dual objective is a linear function of θ. If $a_2 > 0$, the dual objective increases together with θ. If $a_2 < 0$, the dual objective decreases if θ increases.

The dual objective Z of the step size θ is the minimum of $Z(\theta, j)$ where $j \notin J$:

$$Z(\theta) = \min_{j \notin J} Z(\theta, j) \tag{5.4.4}$$

As each of $Z(\theta, j)$ is linear, $\min_{j \notin J} Z(\theta, j)$ is a piecewise linear and concave function.

Theorem 14. *The dual objective function $Z(\theta)$ is a piecewise concave linear function.*

Proof. The dual objective function $Z(\theta)$ is the minimum of a set of linear functions; therefore, it is piecewise linear.

Each linear function is both convex and concave and the minimum of a set of concave function is concave.

□

From the above study, we found that the objective value Z as a function of the step size θ is a 2-dimensional polyhedron, if one considers the shaded area under the piecewise linear function together with the constraints $\theta \geq 0$ and $Z \geq 0$, as illustrated in Figure 5.5.

$Z(0)$ is the dual objective value if $\theta = 0$ that corresponds to the dual objective value from the previous iteration. The optimal step size θ^*, as well as the corresponding Z^*, is achieved at one of the intersection points of the piecewise linear function as illustrated in Figure 5.5.

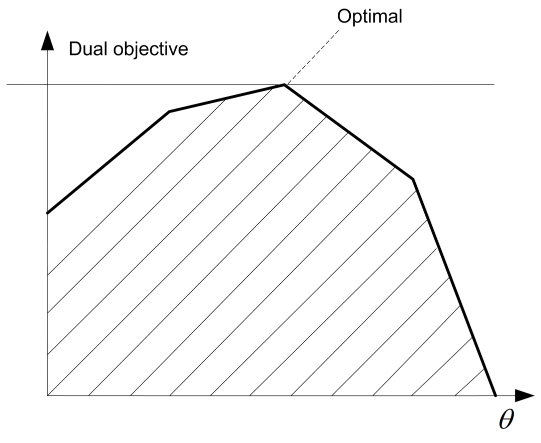


Figure 5.5: The dual objective value Z as a function of θ.

5.4.2 Find the Optimal Step Size

As we know from the previous section, given a step size θ, each column $j \notin J$ has a linear function associated with it and the dual objective is the minimum of all linear functions for $j \notin J$. We can find column i active at θ if $Z(\theta) = Z(\theta, i)$. For arbitrary θ, we have at least one active column. If the piecewise linear function has an intersection point at θ and then we have at least two active columns (see Figure 5.6).

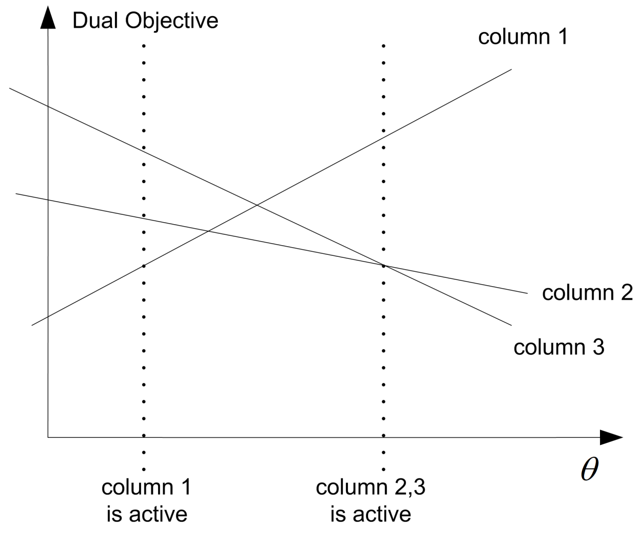


Figure 5.6: Active columns for step sizes.

The optimal step size can be found as follows: let $\theta_{\min} = 0$ and $\theta_{\max}$ be sufficiently large so that the slope of its active column is negative. If the slope is positive, double $\theta_{\max}$, and try it again, until a negative slope is found. When $\theta_{\min} = 0$, the slope of the active column is always positive, otherwise it is not possible to improve the dual feasible solution. Find the linear function $Z(\theta_{\min}, i) = a_i + \theta_{\min} b_i$ where column i is the active function for step size $\theta_{\min}$ and $Z(\theta_{\max}, j) = a_j + \theta_{\max} b_j$ where column j is the active function for step size $\theta_{\max}$.

Solve a new step size θ using the formula, $a_i + \theta b_i = a_j + \theta b_j$, where θ is the point when the above two linear functions intersect. We have

$$\theta = \frac{a_j - a_i}{b_i - b_j} \tag{5.4.5}$$

It is clear that $\theta_{\min} \leq \theta \leq \theta_{\max}$. Suppose column k is active at θ and its correspond linear function is $Z(\theta, k) = a_k + b_k\theta$; then $\theta_{\min} = \theta$ if $b_k > 0$ and $\theta_{\max} = \theta$ if $b_k < 0$. Repeat above steps to update θ until the optimal step size is found.

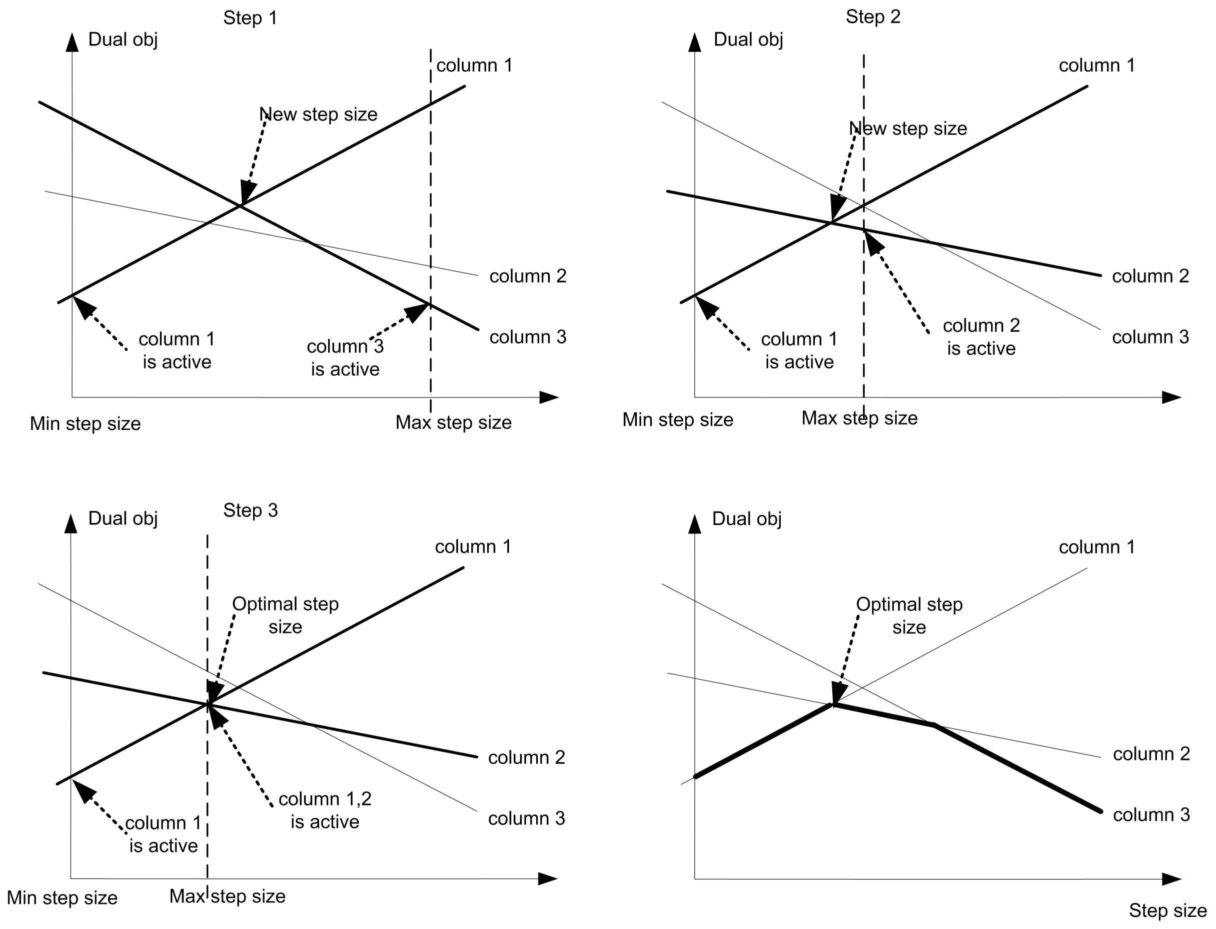


Figure 5.7: An illustrative example to find the optimal step size θ^*.

Algorithm 5.3 Find optimal step size on a piecewise linear concave function

```
Require: $\theta_{\min}$, $\theta_{\min}$
Ensure: $\theta$
 1: while TRUE do
 2:     Find column $i$ which is active at $\theta_{\min}$, its linear function $Z(\theta_{\min}, i) = a_i + \theta_{\min} b_i$, where $b_i > 0$.
 3:     Find column $j$ which is active at $\theta_{\max}$, its linear function $Z(\theta_{\max}, j) = a_j + \theta_{\max} b_j$, where $b_j < 0$.
 4:     $\theta \leftarrow \frac{a_j - a_i}{a_i - a_j}$
 5:     Find column $k$ which is active at $\theta$, its linear function $Z(\theta, k) = a_k + \theta b_k$.
 6:     if $\theta = \theta_{\min}$ OR $\theta = \theta_{\max}$ OR $b_k = 0$ then
 7:         STOP, OPTIMAL STEP SIZE IS FOUND
 8:     end if
 9:     if $b_k < 0$ then
10:         $\theta_{\max} \leftarrow \theta$
11:     end if
12:     if $b_k > 0$ then
13:         $\theta_{\min} \leftarrow \theta$
14:     end if
15: end while
```

We give an illustrative example from Figure 5.7: given $\theta_{\min} = 0$, we have $\theta_{\max}$, line 1 is active at $\theta_{\min}$, line 3 is active at $\theta_{\max}$. We set the intersection of lines 1 and 3 as θ, at which line 2 is active. As line 2 is decreasing, we set $\theta_{\min} = \theta$. Repeat the above step and set the intersection of lines 1 and 2 as θ, at which both lines 1 and 2 are active. We select line 2 again and set $\theta_{\max} = \theta$. The intersection of lines 1 and 2 is at $\theta_{\max}$ thus $\theta_{\max}$ is the optimal step size. It is also clear that the dual objective as a function of the step size is a piecewise linear concave function as illustrated.

5.4.3 The Algorithm

We put all the elements introduced in the previous sections into a full algorithm:

Algorithm 5.4 The Long Step Primal Dual Simplex Method

```
Require: $A$,$D$, $b$, $u$, $\pi$, $\sigma$, $\varepsilon$
Ensure: $\pi^*, \sigma^*$
 1: while TRUE do
 2:     $J \leftarrow \{j : c_j - \pi A_j - \sigma D_j \leq \varepsilon\}$
 3:     Setup and solve the Restricted Master Problem, get the objective value $Z^*$ and
        dual solution $(\rho, \tau)$.
 4:     if $Z^* = 0$ in phase I, or $c - \rho A - \tau D \geq 0$ in phase II then
 5:         Optimal, STOP;
 6:     else
 7:         Find the optimal step size $\theta^*$ using Algorithm 5.3
 8:         if $\theta^* = +\infty$ then
 9:             Infeasible, STOP;
10:         end if
11:         $\pi \leftarrow \pi + \theta^* \rho$
12:         for $k = 1$ to $K$ do
13:             $\sigma_k \leftarrow \min_{i \in A^k}(c_i - \pi A_i)$
14:         end for
15:     end if
16: end while
```

CHAPTER VI

LONG STEP PRIMAL DUAL SUBPROBLEM SIMPLEX METHODS FOR MULTICOMMODITY FLOW PROBLEMS

The Multicommodity Flow problem (MCF) simultaneously ships multiple commodities through a network so that the total amount of flow on each arc is no more than its capacity. It is an extension of the single commodity network flow problem, which can be solved polynomially. To share arcs among commodities makes the problem much more difficult to solve.

In this chapter, we will focus on the Minimum Cost Multicommodity Flow problem, which finds the flow assignment satisfying the supplies and demands of all commodities with minimum cost without violating the capacity constraints.

We have following assumptions for the Minimum Cost Multicommodity Flow Problem:

1. **Homogeneous goods:** each unit flow of each commodity uses 1 unit of capacity of the arc.

2. **No congestion:** the cost on each arc is linear on the flow.

3. **Rational arc costs:** the arc costs are rational number.

4. **Nonnegative arc costs:** no arc costs can be less than 0.

5. **Capacity on arcs:** we assume that all capacities upper bounds are on arcs. If there are capacity upper bounds on any node, we can make the network

transformation to change it to capacity on the arc as indicated in some references [1].

6. **Each commodity ships from one node to another:** we assume that each commodity has exactly one source node and one sink node. If it is not the case, we can apply the commodity split step to break down commodities into one source and one sink node.

7. **The Network is directed:** if the network is undirected, we can convert it into a directed one. [1]

The multicommodity flow problem has a wide variety of application areas, such as telecommunication, logistics, transportation, production planning and scheduling, VLSI design, graph partitioning and network design[28].

Many optimization approaches were developed for solving the multicommodity flow problem. Those approaches can be classified as follows[1]:

- Price-directive decomposition
  - Lagrangian relaxation: Lagrangian multipliers are placed on the bundle constraints to bring them to the objective function. The resulting formulation can be decomposed into a series of minimum cost flow problems for the commodities, which can be solved efficiently. The Lagrangian relaxation method adjusts the multipliers until an optimal solution is found.
  - Dantzig-Wolfe decomposition: by ignoring or imposing penalty on the bundle constraints, the problem is decomposed into a series of minimum cost flow problems. A subproblem and a price-setting linear program are solved iteratively until the optimal solution is found.
  - Column Generation: a path flow formulation is proposed. A subset of columns was chosen to form the restricted master problem and delayed

column generation approaches were used to obtain profitable columns.

- Resource-directive decomposition: arc capacities are allocated to each commodity k and a minimum cost flow problem is solved for each commodity. The information from the solutions are used to update the capacity allocation to improve the overall cost.

- Partitioning methods.

Barnhart [5] used the primal dual simplex method to solve the multicommodity flow problem. A Flow Adjustment Algorithm was proposed to solve the restricted master problem. Wang [28] compared the performance of the generic primal dual simplex method to a primal dual key path method. Several new multiple pairs shortest path algorithms were proposed to efficiently solve problems with many pairs.

In Lagrangian relaxation, Babonneau [2] modified and specialized an Analytic Center Cutting Plane Method. Cutting plane methods and interior point methods were used to solve the multicommodity flow problem.

The rest of the chapter will be organized as follows. Section 6.1 introduces the node arc formulation, path formation and sub-network formulation for the Minimum Cost Multicommodity Flow Problem. Section 6.2 introduces the pricing network and solution methods for the pricing subproblems. Section 6.3 introduces Dantzig-Wolfe decomposition, which is very efficient for MCF problems. Section 6.4 introduces row generation, which generates arc capacity constraints to make subproblems easier to solve. Section 6.5 introduces primal dual simplex methods for MCF and its major components such as ϵ-residual network and step size calculations. Section 6.6 introduces the long step primal dual simplex method and algorithms to calculate the optimal step size. Section 6.7 reports computational experiments we conducted using the long step primal dual subproblem method on minimum cost multicommodity flow problems.

6.1 Multicommodity Flow Formulations

Given a directed graph $G = (N, A)$, let N denote the set of all nodes, A denote the set of all arcs in G, where $(i, j) \in G, i \in N$ is the head node, $j \in N$ is the tail node, $i \neq j$. Let K denote the set of all commodities. For commodity $k \in K$ with origin $s_k \in N$, destination $t_k \in N$ and demand $b_k > 0$. The Minimum Cost Multicommodity Flow Problem can be modeled as following three formulations:

6.1.1 The Node Arc Formulation

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k \tag{6.1.1}$$

Subject to:

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}, \forall (i, j) \in A \tag{6.1.2}$$

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = b_i^k, \forall k \in K, \forall i \in N \tag{6.1.3}$$

$$x_{ij}^k \geq 0, \forall (i, j) \in A, \forall k = 1, 2, ..., K \tag{6.1.4}$$

where:

N**:** is the set of nodes, indexed by i;

A**:** is the set of arcs, indexed by (i, j), where i is the tail node and j is the head node;

K**:** is the set of commodities in the problem, indexed by k;

x_{ij}^k**:** is the flow variable on arc (i, j) for commodity k;

c_{ij}^k**:** is the cost of per unit flow of commodity k on arc (i, j);

u_{ij}**:** is the capacity on arc (i, j);

b_i^k: is the demand or supply on node i for commodity k, if $b_i^k > 0$, it is a supply; if $b_i^k < 0$, it is a demand. According to our assumptions, each commodity should have exactly two non-zero elements in $b_i^k, \forall i \in N$ and the summation should be zero.

Equation (6.1.1) is the objective function to minimize the total flow cost; equation (6.1.3) is the flow conservation constraints for each commodity; equation (6.1.4) ensure that the flow variables are nonnegative on each arc; equation (6.1.2) is the *bundle constraint* which make the Minimum Cost Mutlicommodity Problem difficult. Without equation (6.1.2), this problem can be decomposed into a series of $|K|$ Minimum Cost Flow Problems, which can be solved by polynomial algorithms.

6.1.2 The Path Formulation

The Multicommodity Flow Problem can be reformulated as a path formulation or tree/sub-network formulation. It is known from the theory of network flows that any network flow problems can be reformulated as a path or cycle flow formulation. Based on assumptions introduced at the beginning of this chapter, each arc cost is nonnegative, thus the cost of every cycle in the network is nonnegative. As a result, the flow on every cycle, if any, in the optimal shortest path is zero; therefore, we can eliminate the cycle flow variables and represent any shortest path optimal solutions as the sum of path flows[1].

We have the following equivalent path flow formulation of the MCF problem.

$$\min \sum_{k \in K} \sum_{p \in P^k} c_p f_p \tag{6.1.5}$$

Subject to:

$$\sum_{k \in K} \sum_{p \in P^k} b_k \delta_{ij}(p) f_p \leq u_{ij}, \forall (i,j) \in A \tag{6.1.6}$$

$$\sum_{p \in P^k} f_p = 1, \forall k \in K \tag{6.1.7}$$

$$f_p \geq 0, \forall k \in K, \forall p \in P^k \tag{6.1.8}$$

where:

A**:** is the set of arcs, indexed by (i, j), where i is the head node and j is the tail node;

K**:** is the set of commodities in the problem, indexed by k;

P^k**:** is the set of all paths starting from the source node and ending at the sink node for commodity k, indexed by p;

$\delta_{ij}(p)$: is the indicator whether arc $(i, j) \in p$, if $(i, j) \in p$, then $\delta_{ij}(p) = 1$, otherwise, $\delta_{ij}(p) = 0$;

f_p**:** is the flow variable on path p;

c_p**:** is the cost of the path p;

u_{ij}**:** is the capacity on arc (i, j);

d_k**:** is the demand/supply for commodity k.

Its dual formulation is:

$$\max \sum_{(i,j) \in A} \pi_{ij} u_{ij} + \sum_{1 \leq k \leq K} \sigma_k b_k \tag{6.1.9}$$

Subject to:

$$\sum_{(i,j) \in p} \pi_{ij} \delta_{ij}(p) + \sigma_k \leq c_p, \forall k \in K, \forall p \in P^k \tag{6.1.10}$$

$$\pi_{ij} \leq 0, \forall (i, j) \in A \tag{6.1.11}$$

where:

π_{ij}**:** is the dual for arc (i, j);

σ_k**:** is the dual for commodity k.

We introduce complementary slackness conditions for the Multicommodity Flow Problem:

Theorem 15. *(Multicommodity Flow Complementary Slackness) A pair of feasible primal and dual solutions f, (π, σ) is optimal if and only if*

$$\pi_{ij}(\sum_{k\in K}\sum_{p\in P^k}\delta_{ij}(p)f_p - u_{ij}) = 0, \forall(i,j) \in A \tag{6.1.12}$$

$$\sigma_k(\sum_{p\in P^k} f_p - d_k) = 0, \forall k \in K \tag{6.1.13}$$

$$c_p - \sum_{(i,j)\in p}\pi_{ij} - \sigma_k \geq 0, \forall k \in K, \forall p \in P^k \tag{6.1.14}$$

$$(c_p - \sum_{(i,j)\in p}\pi_{ij} - \sigma_k)f_p = 0, \forall k \in K, \forall p \in P^k \tag{6.1.15}$$

Rewrite the path flow formulation in a compact form and we then have a set packing problem with convexity constraints:

$$\min c^T x \tag{6.1.16}$$

Subject to:

$$Ax \leq b \tag{6.1.17}$$

$$Dx = u \tag{6.1.18}$$

$$x \geq 0 \tag{6.1.19}$$

where vector b is the capacity for each arc $(i, j) \in A$, vector u is the demand/supply of commodities, matrix A is the set of paths and matrix D is a block diagonal matrix with one non-zero entry in each column indicating a particular path's commodity.

6.1.3 The Sub-network Formulation

The path formulation can be extended to a sub-network formulation by defining super commodities. Let S denote the set of origin nodes or super-commodities, i.e. $S \subseteq N$. Super-commodity $s \in S$ is the set of all commodities with source node s. Let the set of commodities with the origin node s be K^s.

Sub-network g for super-commodity $s \in S$ is the combination of paths, where one path p_k for each commodity $k \in K^s$.

Let $\varphi_{ij}(t)$ denote the flow on arc (i, j) for sub-network g for super-commodity $s \in S$, we have $\varphi_{ij}(t) = \sum_{k \in K^s} \sum_{p \in P^k} \psi_p(g) b_{ij} \delta_{ij}(p)$, where $\psi_p(g)$ is the indicator that path p is in sub-network g.

The sub-network formulation is as follows:

$$\min \sum_{s \in S} \sum_{g \in G^s} c_g f_g \tag{6.1.20}$$

Subject to:

$$\sum_{s \in S} \sum_{g \in G^s} \varphi_{ij}(g) f_g \leq u_{ij}, \forall (i, j) \in A \tag{6.1.21}$$

$$\sum_{g \in S^s} f_g = 1, \forall s \in S \tag{6.1.22}$$

$$f_g \geq 0, \forall s \in S, \forall g \in G^s \tag{6.1.23}$$

where:

A**:** is the set of arcs, indexed by (i, j), where i is the tail node and j is the head node;

S**:** is the set of super-commodities, indexed by s;

G^s**:** is the set of all sub-networks with the origin node s, indexed by g;

f_g**:** is the flow variable on sub-network g;

c_g: is the cost of the sub-network g;

$\psi_{ij}(g)$: is the flow on arc $(i,j) \in g$, otherwise, $\psi_{ij}(g) = 0$;

u_{ij}: is the capacity on arc (i,j);

6.1.4 Comparison of Formulations

We compare the LP matrix sizes for different formulations:

Table 6.1: LP matrix sizes of optimization formulations

Formulation	Number of Constraints	Number of Variables
Node-Arc	$\|N\| \times \|K\| + \|A\|$	$\|A\| \times \|K\|$
Path	$\|A\| + \|K\|$	$\sum_{k \in K} \|P^k\|$
Sub-network	$\|A\| + \|S\|$	$\sum_{s \in S} \|G^s\|$

where:

N: is the set of nodes, indexed by i;

A: is the set of arcs, indexed by (i,j), where i is the tail node and j is the head node;

K: is the set of commodities in the problem, indexed by k;

S: is the set of super-commodities or origins in the problem, indexed by s;

P^k: is the set of all paths for commodity k, indexed by p;

G^s: is the set of all sub-networks for super-commodity s, indexed by g;

To compare the path and sub-network formulations, Barnhart [6] provided an example in which the sub-network formulation outperforms the path formulation, when the number of commodities is larger, i.e. in the order of $O(|N|^2)$.

Because the number of super-commodities $|S|$ is limited by the number of nodes $|N|$, while the number of commodities can be $O(|N|^2)$, the sub-network formulation often has less rows compared to the path formulation. However, the number of variables can be significantly larger. We can illustrate this through a simple example as follows:

Suppose a super-commodity s has 10 commodities and each commodity has 10 paths, i.e. $|K^s| = 10$ and $|P^k| = 10$ for $k \in K^s$. The number of variables in the path formulation is $\sum_{k\in K^s} |P^k| = 100$ and the number of variables in the sub-network formulation is $\prod_{k\in K} |P^k| = 10$ billion.

6.2 The Pricing Network

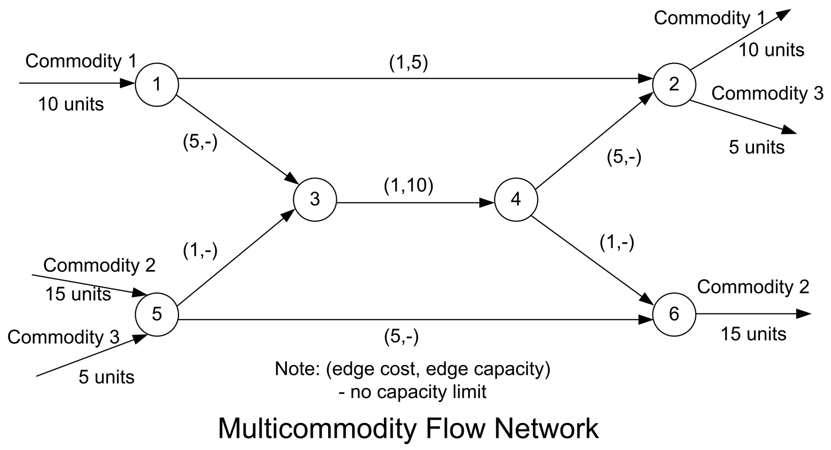
We set up a pricing network $\widetilde{G} = (N, \widetilde{A})$ with the same set of nodes, arcs and commodities as the original multicommodity network. For each arc $(i, j) \in A$, the arc capacity remains unchanged, but the arc cost is changed to $\widetilde{c}_{ij} = c_{ij} - \pi_{ij}$, where c_{ij} is the arc cost of the original multicommodity network and π_{ij} is the dual value associated with arc (i, j).

Figure 6.1 illustrates a multicommodity network and its corresponding pricing network. The node set and arc set are the same for two networks, and the cost of two capacitated arcs, $(1, 2)$, $(3, 4)$, are $c_{(1,2)} - \pi_{1,2}$ and $c_{(3,4)} - \pi_{3,4}$ for the pricing network, and $c_{(1,2)}$ and $c_{(3,4)}$ for the multicommodity network. There are only two capacitated arcs in the network, which have set packing type constraints in the optimization, thus have associated dual values $\pi_{(1,2)}$ and $\pi_{(3,4)}$.

As the dual values π are associated with set packing type constraints, we have $\pi \leq 0$. The arc cost of the multicommodity network is always nonnegative, i.e. $c_{(i,j)} \geq 0, \forall (i, j) \in N$. The arc costs in the pricing network are also nonnegative, i.e. $\bar{c}_{(i,j)} = c_{(i,j)} - \pi_{(i,j)} \geq 0$.

6.2.1 The Pricing Problem

Given a dual solution (π, σ), the reduced cost of a path p is $\bar{c_p} = \sum_{(i,j)\in p}(c_{(i,j)} - \pi_{(i,j)}) - \sigma_k$, if path p is for commodity k. The first part of the reduced cost $\sum_{(i,j)\in p}(c_{(i,j)} - \pi_{(i,j)})$ is the summation of arc costs in the pricing network. The second part is the dual



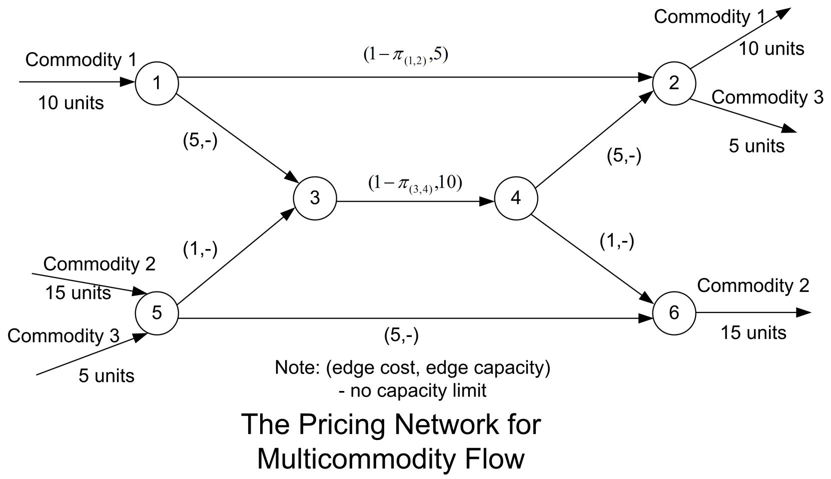


Figure 6.1: A multicommodity network and its corresponding pricing network.

value associated with commodity k, which is not part of the network. We have the shortest path formulation for subproblem k for commodity $k \in K$:

$$\min \sum_{(i,j)\in A} (c_{ij} - \pi_{ij})x_{ij} \tag{6.2.1}$$

Subject to:

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = \begin{cases} -1, & i = s; \\ 0, & i \neq s, i \neq t; \\ 1, & i = t. \end{cases} \quad \forall i \in N \tag{6.2.2}$$

$$x_{ij} \geq 0, \forall (i,j) \in A \tag{6.2.3}$$

where:

A**:** is the set of arcs, indexed by (i, j), where i is the head node and j is the tail node;

s_k**:** is the source node for commodity k;

t_k**:** is the sink node for commodity k;

x_{ij}**:** is the flow variable on arc (i, j);

c_{ij}**:** is the cost of arc (i, j);

π_{ij}**:** is the dual price of arc (i, j);

Because the arc cost is nonnegative $c_{ij} \geq 0$ and the dual price $\pi_{ij} \leq 0$, the updated cost for arc (i, j) is nonnegative, i.e. $c_{ij} - \pi_{ij} \geq 0$, we can use Dijkstra's algorithm for each origin node $s \in S$ to find the shortest path for all commodities with source node s. The time complexity of Dijkstra's algorithm is $O(|A|)$, where $|A|$ is the number of arcs in the network, the total time to price out all commodities is $O(|A||S|)$, which is consistent with existing algorithms for all pairs or multi-pair shortest path algorithms. Wang [28] provided a comprehensive review of related algorithms.

Figure 6.2 gives an example for shortest paths on the MCF network. The first network ship commodity 2,3 from node 5 to node 2 and node 6. We find the shortest paths from node 5 (or node s, which is equivalent) to nodes 2 and 6. The second network ship commodity 1 from node 1 to node 2.

For the shortest path $p_{5,2}$ from node 5 to node 2, we calculate its reduced cost by $\bar{c}_p - \sigma_1$, where $\bar{c}_p$ is the summation of all arc cost on the path and σ_1 is the dual value associated with commodity 1.

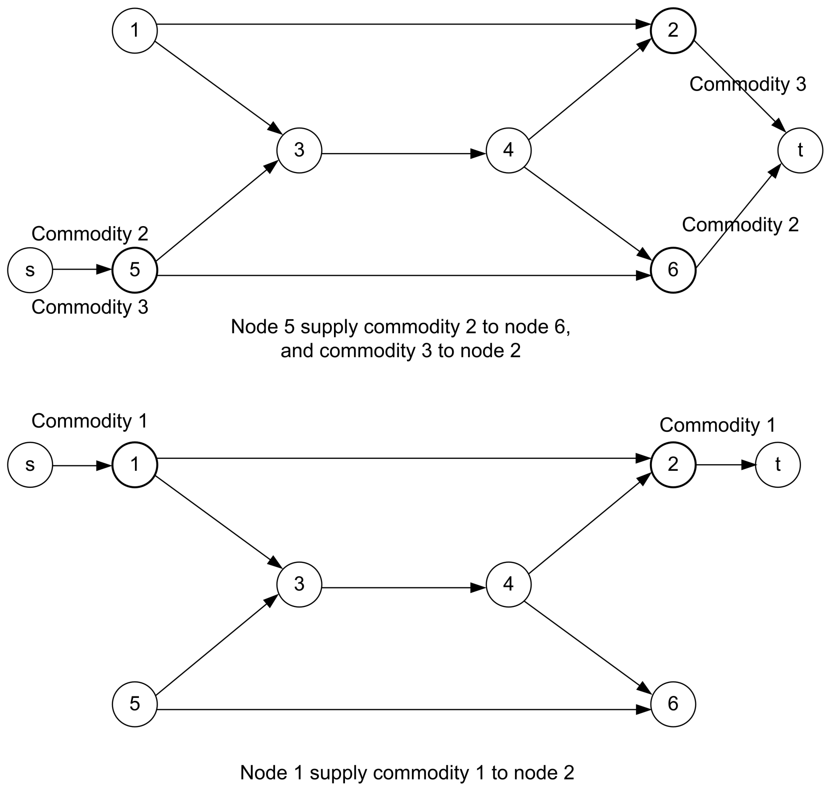

Commodity 3
Commodity 2
Commodity 2
Commodity 3
Node 5 supply commodity 2 to node 6, and commodity 3 to node 2
Commodity 1
Commodity 1
Node 1 supply commodity 1 to node 2

Figure 6.2: Forward Shortest Paths on Multicommodity Network.

6.3 Dantzig-Wolfe Decomposition

Dantzig-Wolfe (DW) decomposition is a common technique used to solve problems with block-angular structure. It is a generalization of Tucker's work on the multicommodity flow network and it is one of most efficient exact algorithms to solve MCF problems.

Dantzig-Wolfe decomposition can be applied using path or sub-network formulations. In the path formulation, the algorithm decomposes the problem into a RMP and $|K|$ subproblems, one for each commodity. The RMP is a path formulation of the multicommodity flow problem with a subset of paths generated by subproblems. Each subproblem is a shortest path problem on the pricing network as introduced in section 6.2, with additional arc cost of π_{ij} on arc $(i, j) \in A$, where π_{ij} is the optimal dual solution of RMP for arc (i, j).

In the sub-network formulation, the algorithm decomposes the problem into a RMP and $|S|$ subproblems, one for each super-commodity. The RMP is a sub-network formulation of the multicommodity flow problem with a subset of sub-networks generated by subproblems. Each subproblem is a shortest path problem on the original pricing network as introduced in section 6.2, with additional arc cost of π_{ij} on arc $(i, j) \in A$, where π_{ij} is the optimal dual solution of RMP for arc (i, j). For each super-commodity s, a shortest path from each commodity $s \in K^s$ is combined to form the sub-network.

After receiving the latest dual optimal solution from the RMP, each subproblem then either provides a new path/sub-network with negative reduced cost or reports no such paths/sub-networks exist. The paths/sub-networks provided by the subproblems are added to the RMP and then solved to optimality. The above steps will be repeated until no subproblem can provide negative reduced cost paths/sub-networks. Then the global optimal solution for the multicommodity flow problem is found.

The algorithm can discard any non-basic columns in each RMP iterations. However, it might be advantageous to keep old columns because their reduced costs may become negative in later iterations. The algorithm is finite if we keep all generated columns because we will eventually generate all columns in each subproblem that is finite.

Because signs of constraints in the RMP associated with arcs $(i, j) \in A$ are less than or equal to, we have $\pi_{ij} \leq 0$. As $c_{ij} \geq 0$, $c_{ij} - \pi_{ij} \geq 0$ for all $(i, j) \in A$, the shortest path problem can be solved efficiently using Dijkstra's algorithm.

6.4 Row Generation

For capacitated arcs in the multicommodity flow problems, the percentage of saturated arcs in the final optimal solution is typically low. Therefore, if an arc capacity is large enough, i.e. larger than its total flow upon the global optimal solution, it can be treated as an uncapacitated arc and eliminated from the LP formulation.

As it is impossible to know beforehand which arc will be saturated and the set of saturated arcs can change across iterations, a row generation approach is proposed to dynamically find saturated arcs and include them into the LP formulation. The overall process is as follows:

Given a RMP, we first assume all arcs are uncapacitated and solve it to optimality. Then we add total flow on each arc and compare it with arc capacity for violations. Add all rows that correspond to violated arcs into the LP formulation and resolve. Repeat the above process until no capacity constraints are violated.

6.5 The primal dual simplex method for MCF problems

The primal dual simplex method (phase I) is a dual ascent LP solution method that starts with a feasible dual solution and then iteratively constructs a primal feasible RMP based on the complementary slackness conditions. It uses the RMP dual solution to improve the current dual solution if primal infeasibility still exists in the RMP. The algorithm terminates when all primal infeasibility disappears.

For simplicity, we assume the sub-network formulation is used in the following discussions.

6.5.1 Construct the RMP problem for primal dual simplex methods

In primal dual subproblem simplex methods, it is critical to include all columns with reduced costs under threshold ϵ into the RMP, otherwise the algorithm may not be able to converge. This requirement makes constructing RMP for the primal dual simplex method more difficult than that for the Dantzig-Wolfe decomposition.

For smaller problems, we can enumerate all columns and calculate their reduced costs, form the RMP using eligible columns and exclude all ineligible columns.

For large cases, it may not be possible to check all columns to form the RMP. We use a filtering approach as follows to make sure all columns with reduced costs under threshold ε will be included in the RMP, but other columns with larger reduced costs can be included as well. When threshold ε is set properly, many arcs can be excluded and the resulting network will be much smaller.

6.5.1.1 ε-Residual Network

When the MCF problem is big and too time consuming to enumerate all columns, we can define admissible columns by forming a ε-Residual Network:

For each commodity $k \in K$, we denote $R_{ij}^k = SP_{s_k,i} + (c_{ij} - \pi_{ij}) + SP_{j,t_k} - SP_{s_k,t_k}$ for each arc $(i, j) \in A$, R_{ij}^k is the distance above the shortest path between s_k and t_k if arc (i, j) is included.

Denote $R_{ij} = \min_{k \in K} R_{ij}^k$, which is the distance above the shortest path for any commodity $k \in K$ and define the ε residual network G^ε as the sub-network consists of arcs with R_{ij}^k of at most ε, i.e. $G^\varepsilon \leq \varepsilon$.

The size of ε-Residual Network can be controlled by the ε value and the resulting network is typically much smaller than the original network.

Let $A^+ = A | R_{ij} \leq \varepsilon$ be the set of arcs included in the restricted master problem. We can eliminate arcs in $(i, j) \notin A^+$, because $R_{ij}^k \geq R_{ij} > \tau$ for all $(i, j) \notin A^+$ and any path with arc (i, j) will have path cost greater than the shortest path plus ε, thus should be excluded from the RMP problem.

As for columns remaining in A^+, it is still possible to have paths with costs greater than the threshold ε. If such columns are active in the final simplex basis upon convergence, then the problem is not optimal. Such phenomenon can often happen when ε is big, even if the RMP is exact.

For primal dual simplex method phase I, the RMP is still a multicommodity flow problem with all nodes and commodities as the original MCF problem and a subset of arcs with arc cost of 0. We denote it the RMP-MCF-I problem. It can be solved using traditional MCF algorithms, where the arc set is smaller and arc costs are 0.

The RMP-MCF-I problem is easier than the original problem.

If the optimal objective value of the RMP-MCF-I is positive, which means there must exist primal infeasibility when using only the current column set in the RMP-MCF-I, then the optimal dual solution of the RMP-MCF-I can be used as an improving direction for the current dual solution as we explained previously. On the other hand, if the optimal objective value of the RMP-MCF-I is zero, we have achieved primal feasibility while maintaining dual feasibility.

If all columns in the final basis have zero reduced costs, thus satisfied complementary slackness conditions, we have the optimal solution.

Otherwise, if some columns with positive reduced costs are active in the final phase I problem, the solution is only a near optimal solution for the MCF problem and a phase II approach is needed to make it optimal.

Many phase II approaches can be used, such as Dantzig-Wolfe decomposition/ columns generation. In this research work, we use primal dual subproblem simplex method phase II for MCF to find the global optimal solution, starting from the given near optimal solution from phase I.

For primal dual simplex method phase II, an initial near optimal solution is provided by RMP-MCF-I and the RMP is a multicommodity flow problem with all nodes and commodities as the original MCF problem with a subset of arcs. We denote it the RMP-MCF-II problem. Similarly, it can also be solved using traditional MCF algorithms. It is also easier than the original problem because of the initial feasible solution and a smaller arc set.

Upon the convergence of the primal dual subproblem simplex method phase II, the optimal solution of the final RMP-MCF-II problem is the global optimal solution for the original multicommodity flow problem.

6.5.2 Solve the RMP problem

For the RMP-MCF-II problem, the solution method is exactly the same as methods for the original MCF problem and will not be repeated here. In this section, we focus on methods for the RMP-MCF-I problem.

After the filtering step was introduced in the previous section, a restricted master multicommodity flow problem phase I(RMP-MCF-I) network resulted in $G^+ = (N, G^+)$. The node set is still the same, but the arc set is smaller. The RMP-MCF-I problem is a special MCF problem with zero arc costs.

For large problems, it is impossible to enumerate all columns in the ε residual network, thus we use column generation to solve the RMP-MCF-I, which is a phase I problem with all zero arc costs.

Given a dual solution (ρ, τ), the pricing network for the RMP has arc cost of $0 - \pi_{ij} \geq 0$, thus still can be solved using the Dijkstra's algorithms.

After the RMP problem with generated columns is solved, the new dual solution is used to solve subproblems for each super-commodity and sub-networks with negative reduced costs are added to the RMP. Repeat the above steps until an optimal solution is found and the optimal dual solution for RMP will be passed back to the primal dual simplex method.

6.5.3 Calculate the Step Size θ

It is known from the previous chapter that the optimal dual solution of the RMP problem, i.e. RMP-MCF-I or RMP-MCF-II, is an improving direction for the current feasible dual solution (ρ, τ).

Suppose (ρ, τ) is an optimal dual solution for the RMP. For primal dual simplex phase I, the feasible dual solution is updated using the formula $(\pi, \sigma) := (\pi, \sigma) + \theta(\rho, \tau)$.

For primal dual simplex phase II, the feasible dual solution is updated using the formula $(\pi, \sigma) := (1-\theta)(\pi, \sigma) + \theta(\rho, \tau)$.

Additionally, (ρ, τ) is a feasible direction, but the step size θ needs to be smaller than a specific value to maintain dual feasibility. The step size θ must be positive, otherwise the dual feasible solution will remain unchanged and the PD algorithm will not converge. If all columns with reduced costs less than or equal to $\varepsilon > 0$ are included in the RMP problem, we can find a positive step size θ.

For the primal dual simplex method phase I, if the step size θ can be increased indefinitely while dual feasibility is always preserved, then the dual problem is unbounded and the primal problem is infeasible.

For the primal dual simplex method phase II, the step size θ is limited between 0 and 1. If it can be increased to 1 while the dual feasibility is preserved, then the dual solution (ρ, τ) for the RMP is also a dual feasible solution for the original MCF problem. Therefore, a global optimal solution is found and the algorithm can be terminated.

For each capacitated arc, there is a corresponding constraint in equation 6.1.17 and an associated dual value $\pi_{i,j}$. The arc associated reduced cost is $\bar{c}_{(i,j)} = c_{(i,j)} - \pi_{(i,j)} > 0$, as $c_{(i,j)} > 0$ and $\pi_{(i,j)} \leq 0$. For each commodity, there is an arc from the node with demand to the super sink node, whose reduced cost is 0 for commodity k.

The reduced cost for a path p is the summation of reduced costs of arcs on the path minus the dual value for the corresponding commodity σ_k:

$$\bar{c}_p = \sum_{(i,j)\in p} \bar{c}_{(i,j)} - \sigma_k \tag{6.5.1}$$

To use the long step primal dual subproblem simplex method, we fix a step size θ, solve shortest path problems using Dijkstra's algorithm for each commodity $k \in K$, let $\bar{c}_p^k$ be the shortest path for commodity k and set $\sigma_k = \bar{c}_p^k$. From shortest paths for each commodity, we calculate the corresponding dual objectives and pick the path with the smallest dual objective. This path is the active column for the step size θ

and we calculate its slope.

With $\theta_{\min} = 0$ and $\theta_{\max}$, we calculate the optimal step size using an algorithm similar to Algorithm 5.3 as introduced in the previous chapter.

Algorithm 6.1 Find the optimal step size on Multicommodity Flow Network

```
Require: $\theta_{\min}$, $\theta_{\min}$
Ensure: $\theta$
 1: while TRUE do
 2:     Set dual value using $\pi_{\theta_{\min}} = \pi + \theta_{\min}\rho$, find shortest path for each commodity
        with $\bar{c}^k$. Set $\sigma_k = \bar{c}^k$ and calculate dual objectives for each commodity.
 3:     Find column $i$ which is active at $\theta_{\min}$, its linear function $Z(\theta_{\min}, i) = a_i + \theta_{\min} b_i$,
        where $b_i > 0$.
 4:     Set dual value using $\pi_{\theta_{\max}} = \pi + \theta_{\max}\rho$, find shortest path for each commodity
        with $\bar{c}^k$. Set $\sigma_k = \bar{c}^k$ and calculate dual objectives for each commodity.
 5:     Find column $j$ which is active at $\theta_{\max}$, its linear function $Z(\theta_{\max}, j) = a_j +$
        $\theta_{\max} b_j$, where $b_j < 0$.
 6:     $\theta \leftarrow \frac{a_j - a_i}{a_i - a_j}$
 7:     Set dual value using $\pi_\theta = \pi + \theta\rho$, find shortest path for each commodity with
        $\bar{c}^k$. Set $\sigma_k = \bar{c}^k$ and calculate dual objectives for each commodity.
 8:     Find column $k$ which is active at $\theta$, its linear function $Z(\theta, k) = a_k + \theta b_k$.
 9:     if $\theta = \theta_{\min}$ OR $\theta = \theta_{\max}$ OR $b_k = 0$ then
10:         STOP, OPTIMAL STEP SIZE IS FOUND
11:     end if
12:     if $b_k < 0$ then
13:         $\theta_{\max} \leftarrow \theta$
14:     end if
15:     if $b_k > 0$ then
16:         $\theta_{\min} \leftarrow \theta$
17:     end if
18: end while
```

6.6 The Long Step Primal Dual Simplex Method for Multicommodity Flow Problems

In the previous chapter, a long step primal dual simplex method for linear programming problems with convexity constraints is introduced. In the section, we apply it to solve multicommodity flow problems.

The long step primal dual simplex method is the same as the traditional primal dual simplex method except for how the step size θ is calculated. In traditional primal dual simplex methods, θ is calculated deterministically, to ensure a dual feasible solution. In long step primal dual simplex methods, the step size θ is calculated to maximize the dual objective. Typically, we have a longer step size for the long step primal dual simplex methods, i.e. $\theta_{LPD} \geq \theta_{PD}$, as introduced in the previous chapter.

For multicommodity flow problems, columns are not explicitly listed due to the problem size. Therefore, we introduce a column generation approach to find the optimal step size θ^*.

6.6.1 Find the Optimal Step Size θ^* to Maximize the Dual Objective

Given the initial step size θ_{PD} calculated by the traditional PD method, we denote $\theta_{\min} = \theta_{PD}$ and a maximum step size $\theta_{\max} >> \theta_{\min}$.

It is known from the previous chapter that each step size θ has a corresponding line associated with it, i.e. $a \times \theta + b$. It is descending, if $a < 0$, or ascending if $a > 0$. The optimal step size is derived by combining ascending and descending lines.

We use long step primal dual simplex phase I to illustrate the steps to find the optimal step size. Given a step size θ, a shortest path problem is solved for each $k \in K$. Let SP_k denote the shortest path for $k \in K$, its cost $\hat{c}_{sp} = \sum_{(i,j) \in SP_k} \hat{c}_{ij}$ and the dual value corresponding to commodity k is set to $\sigma_k = \sum_{(i,j) \in SP_k} \hat{c}_{ij}$, so that the reduced cost become zero.

The dual objective is:

$$\begin{aligned} Z_\theta &= \sum_{(i,j) \in A} \pi'_{ij} u_{ij} + \sum_{k \in K} \sigma'_k \\ &= \sum_{(i,j) \in A} \pi'_{ij} u_{ij} + \sum_{k \in K} \sum_{(i,j) \in SP_k} \hat{c}_{ij} \\ &= \sum_{(i,j) \in A} (\pi_{ij} + \theta \rho_{ij}) u_{ij} + \sum_{k \in K} \sum_{(i,j) \in SP_k} (c_{ij} - (\pi_{ij} + \theta \rho_{ij})) \\ &= \sum_{(i,j) \in A} \pi_{ij} u_{ij} + \sum_{k \in K} \sum_{(i,j) \in SP_k} (c_{ij} - \pi_{ij}) + \theta \Big(\sum_{(i,j) \in A} \rho_{ij} u_{ij} - \sum_{k \in K} \sum_{(i,j) \in SP_k} \rho_{ij} \Big) \end{aligned}$$

Therefore, the linear function for θ is $Z_\theta = a\theta + b$, where $a = \sum_{(i,j)\in A} \rho_{ij}u_{ij} - \sum_{k\in K}\sum_{(i,j)\in SP_k} \rho_{ij}$ and $b = \sum_{(i,j)\in A} \pi_{ij}u_{ij} + \sum_{k\in K}\sum_{(i,j)\in SP_k}(c_{ij} - \pi_{ij})$.

Algorithm 6.2 Find the optimal step size for Long Step primal dual subproblem simplex method

```
Require: $\theta_{\min}, N, A, K, \pi, \sigma, \rho$
Ensure: $\theta^*$
 1: Let $a = \sum_{(i,j)\in A} \rho_{ij}u_{ij} - \sum_{k\in K}\sum_{(i,j)\in SP_k} \rho_{ij}$
 2: Let $b = \sum_{(i,j)\in A} \pi_{ij}u_{ij} + \sum_{k\in K}\sum_{(i,j)\in SP_k}(c_{ij} - \pi_{ij})$
 3: Let $\theta = \theta_{\min}$, solve shortest path problems to get $LINE_{\min}$, $a_{\min}$, $b_{\min}$
 4: if $a < 0$ then
 5:     $\theta^* = \theta$, STOP, OPTIMAL STEP SIZE IS FOUND
 6: end if
 7: Find a big enough $\theta_{\max}$ such that $a_{\max} > 0$. Keep $LINE_{\max}$, $b_{\max}$
 8: while TRUE do
 9:     Set dual value using $\pi_{\theta_{\min}} = \pi + \theta_{\min}\rho$, find shortest path for each commodity
        with $\bar{c}^k$. Set $\sigma_k = \bar{c}^k$ and calculate dual objectives for each commodity.
10:     Find column $i$ which is active at $\theta_{\min}$, its linear function $Z(\theta_{\min}, i) = a_i + \theta_{\min}b_i$,
        where $b_i > 0$.
11:     Set dual value using $\pi_{\theta_{\max}} = \pi + \theta_{\max}\rho$, find shortest path for each commodity
        with $\bar{c}^k$. Set $\sigma_k = \bar{c}^k$ and calculate dual objectives for each commodity.
12:     Find column $j$ which is active at $\theta_{\max}$, its linear function $Z(\theta_{\max}, j) = a_j +$
        $\theta_{\max}b_j$, where $b_j < 0$.
13:     $\theta \leftarrow \frac{a_j - a_i}{a_i - a_j}$
14:     Set dual value using $\pi_\theta = \pi + \theta\rho$, find shortest path for each commodity with
        $\bar{c}^k$. Set $\sigma_k = \bar{c}^k$ and calculate dual objectives for each commodity.
15:     Find column $k$ which is active at $\theta$, its linear function $Z(\theta, k) = a_k + \theta b_k$.
16:     if $\theta = \theta_{\min}$ OR $\theta = \theta_{\max}$ OR $b_k = 0$ then
17:         STOP, OPTIMAL STEP SIZE IS FOUND
18:     end if
19:     if $b_k < 0$ then
20:         $\theta_{\max} \leftarrow \theta$
21:     end if
22:     if $b_k > 0$ then
23:         $\theta_{\min} \leftarrow \theta$
24:     end if
25: end while
```

Table 6.2: Three methods for solving Multicommodity Flow Problems

Algorithm	Solution Method
PD	primal dual simplex method phase I and phase II
LPD	long step primal dual simplex method phase I and phase II
DW	Dantzig-Wolfe decomposition

6.7 Computational Experiments for Multicommodity Flow Problems

In this section, we share data from the computational experiments conducted that used a collection of publicly available multicommodity flow problems. We implement the primal dual subproblem simplex method (PD), the long step primal dual subproblem simplex method (LPD) and Dantzig-Wolfe decomposition (DW).

In order to solve large-scale problems with millions of commodities, the subnetwork formulation is selected and row generation is used whenever possible to solve the resulting subproblems, before passing them to the simplex solver (Gurobi). All algorithms are coded in Java and all experiments are conducted using Gurobi 5.6 on an OpenSUSE Linux 13.2 desktop with Intel Core i7-3770 @ 3.40GHz CPU and 16 GB memory. Table 6.2 provides a list of algorithms used.

6.7.1 Test Problems

We conduct computational experiments using the Planar data set for multicommodity flows, which are artificial problems that mimic telecommunication networks, generated by Larsson and Yuan [20]. Nodes are randomly chosen as points in the plane and arcs link neighbor nodes in such a way that the resulting graph is planar. Arc costs are Euclidean distances and arc capacities are uniformly distributed. Commodities are pairs of random origin and destination nodes, with uniformly distributed demands.

Larsson and Yuan [20] solved problems up to Planar1000. Bompadre and Orlin [8] solved the multicommodity flow problem as a sequence of subproblems, on very sparse network and solved problems up to Planar800.

Table 6.3: Problem characteristics for test problems

Problem	$\|N\|$	$\|A\|$	$\|K\|$	$\|N\|\|K\| + \|A\|$
Planar300	300	1680	3584	1,076,880
Planar500	500	2,842	3,525	1,765,342
Planar800	800	4,388	12,756	10,209,188
Planar1000	1,000	5,200	20,026	20,031,200
Planar2500	2,500	12,990	81,430	203,587,990
Chicago-Region	12,982	39,018	2,297,945	29,831,961,008

The other test problem is a transportation problem, i.e. Chicago-Region that can be downloaded from *http://www.bgu.ac.il/bargera/tntp/*. Just like Babonneau [2], we divided all demands by a same coefficient until the problem became feasible. We found a feasible solution using coefficient 4 that was used in our computational experiments. In literature, Babonneau [2] used coefficient 6 to have lower demands. The characteristics of the test problems are listed in Table 6.3.

6.7.2 Computational Experiments

6.7.2.1 Impact of the Optimal Step Size

In our computational experiments, we study the performance of the long step primal dual subproblem method and its improvement over the traditional primal dual subproblem simplex method.

In Tables 6.4 and 6.5, we compared step sizes and dual objectives for LPD and PD at iteration 1. Because LPD and PD started from the same initial point, the feasible dual solution π and the RMP dual solution ρ were the same for both approaches and LPD consistently achieved longer step sizes and better dual objectives compared to PD.

As a result, the number of iterations is much smaller for the LPD approach; for example, for the largest case Planar2500 LPD took 9 iterations for Phase I, PD took 35 iterations for Phase I and the total run time is 6 times more for PD than LPD. See details in Tables 6.6 and 6.7.

Table 6.4: Step Sizes for LPD and PD algorithms at Iteration 1 on MCF problems

Problem	LPD Phase I Step Size	PD Phase I Step Size
Planar300	550.0	88.3
Planar500	900.0	450.0
Planar800	254.0	34.3
Planar1000	248.0	18.1
Planar2500	132.5	9.3
Chicago-Region	0.0689	0.0081

Table 6.5: Dual Objectives for LPD and PD algorithms at Iteration 1 on MCF problems

Problem	LPD Phase I Dual Objective	PD Phase I Dual Objective
Planar300	9,685,080.0	2,382,403.3
Planar500	1,313,268.0	945,724.0
Planar800	8,412,130.0	1,703,882.0
Planar1000	103,752,182.0	13,782,620.9
Planar2500	637,432,132.4	78,984,440.2
Chicago-Region	4,707,771.7	1,170,742.9

Table 6.6: Number of iterations of LPD and PD algorithms on MCF problems

Problem	LPD Phase I Iterations	PD Phase I Iterations
Planar300	4	14
Planar500	2	2
Planar800	4	12
Planar1000	6	25
Planar2500	9	35
Chicago-Region	7	20

Table 6.7: Total time (seconds/minutes) of LPD and PD algorithms on MCF problems

Problem	LPD Phase I	LPD Phase I/II	PD Phase I	PD Phase I/II
Planar300	7.721(s)	16.757(s)	21.889(s)	29.526(s)
Planar500	5.325(s)	13.198(s)	4.168(s)	11.695(s)
Planar800	50.328(s)	98.073(s)	101.438(s)	144.116(s)
Planar1000	253.172(s)	717.466(s)	1063.532(s)	1481.518(s)
Planar2500	128.4(m)	746.0(m)	890.9(m)	1551.1(m)
Chicago-Region	921.1(m)	4143(m)	3074.2(m)	7179.6(m)

Table 6.8: Objective values and optimality gaps of LPD and PD algorithms on MCF problems

Problem	LPD Phase I (optimality gap)	PD Phase I (optimality gap)
Planar300	705,664,745.0727462 (2.27%)	701,520.057.9439197 (1.67%)
Planar500	521,276,544.3932601 (8.15%)	521,322,075.0894 (8.16%)
Planar800	1,217,239,781.8000221 (4.27%)	1,210,163,565.91488 (3.67%)
Planar1000	3,597,762,275.6686816 (4.29%)	3,536,266,451.57915 (2.51%)
Planar2500	13,371,649,768.60141 (5.6%)	13,013,849,521.820404 (2.776%)
Chicago-Region	1,850,103,425.34 (0.67%)	1,847,077,784.38 (0.51%)

Table 6.9: Total time (seconds/minutes) of LPD and DW algorithms on MCF problems

Problem	LPD Phase I	LPD Phase I and II	DW
Planar300	7.721(s)	16.757(s)	8.178(s)
Planar500	5.325(s)	13.198(s)	5.519(s)
Planar800	50.328(s)	98.073(s)	41.123(s)
Planar1000	253.172(s)	717.466(s)	599.104(s)
Planar2500	128.4(m)	746(m)	1075.5(m)
Chicago-Region	921.1(m)	4143(m)	4449(m)

6.7.2.2 *Computational Performance: Primal Dual Subproblem Simplex Methods vs Dantzig-Wolfe Decomposition*

In the literature, Barnhart [5] reported that Dantzig-Wolfe decomposition method was about 1.5 to 2 times faster than the PDN (Primal Dual Network) implementations and about 3 times faster than the Primal Dual implementation. Wang [28] reported similar results for multicommodity flow problems with 49 to 300 nodes and 323 to 811 commodities.

Table 6.9 provides computational experiments to compare the long step primal dual subproblem simplex (LPD) with Dantzig-Wolfe decomposition (DW). For all cases, LPD performs very similar to DW and the gap becomes smaller for larger cases, i.e. for Planar1000, LPD phase I is 42% of DW run time and LPD is only 20% more than DW in total run time; for Planar2500, LPD phase I is 23% of DW run time and LPD is faster than DW in total run time.

Table 6.10 provides detailed major and minor iterations for PD, LPD and DW.

Table 6.10: Comparisons on number of iterations

Problem	PD major iterations	PD minor iterations	LPD major iterations	LPD minor iterations	DW iterations
Planar300 Phase I	13	60	4	15	-
Planar300 Phase II	1	15	1	15	19
Planar500 Phase I	2	6	2	6	-
Planar500 Phase II	1	13	1	12	13
Planar800 Phase I	12	49	4	16	-
Planar800 Phase II	1	19	1	19	19
Planar1000 Phase I	24	169	6	36	-
Planar1000 Phase II	1	33	1	34	41

PD and LPD algorithms solve many more subproblems than DW, but each of the subproblems are easier than DW's.

CHAPTER VII

CONCLUSIONS AND FUTURE RESEARCH

This chapter concludes this thesis by highlighting this research work's contributions in Section 7.1 and proposing some potential directions for future research in Combined Objective Least Squares, the long step primal dual simplex method and Multicommodity Flow problems in Section 7.2.

7.1 Conclusions

The first part of this research work focuses on COLS and various matrix factorization methods associated with it.

Simplex methods sometimes encounter the degeneracy problem when the objective value remains unchanged across iterations. Various methods have been introduced to avoid it, such as Bland's rule or interior point methods. COLS methods guarantee strict improvement between iterations, thus eliminate the possibility of degeneracy. In each COLS iteration, a series of least squares problems are solved efficiently to ensure the overall effectiveness of the COLS method. In previous research work, QR decomposition approaches were used to solve least squares problems because of their numerical stability and ability to be used as a black box solver for any least squares problems.

However, QR decomposition approaches are not as efficient as LU decomposition approaches used in simplex methods, especially when solving a series of least squares problems. As a result, the COLS approach solves the linear program in fewer iterations compared to the simplex method, but spends more time in each iteration thus influences COLSs overall performance.

In this research work, we evaluated alternative approaches to solve a series of least squares problems, such as augmented matrix approaches, normal equations approaches, etc., to find a balance between numerical stability and computational performance. In augmented matrix approaches, we formed the augmented matrix and solved it using LU decomposition. While this approach can successfully solve individual least squares problems efficiently, there is no efficient way to solve a series of least squares problems when consecutive problems differ by only one or two columns.

The normal equations approach with Chelosky factorization solved a series of least squares problems efficiently with computational performance comparable to LU decomposition approaches in simplex methods. Although this approach has been used successfully in interior point methods to solve linear programming problems for years and has never encountered numerical difficulty, the normal equations approach has only been stable under certain conditions according to numerical analysis.

After carefully analyzing the QR decompositions computational performance in COLS, we identified the computational bottleneck as the step to add a new column in solving a series of least squares problems, which may be avoided if we use Chelosky factorization instead. Based on these observations, we proposed a hybrid approach to combine QR decomposition and Chelosky factorization to solve a series of least squares problems. We used QR decomposition in all steps except the one to add a column when Chelosky factorization was in use. This hybrid approach can have the theoretical background of a semi-normal equation whose numerical stability can be further improved through iterative refinement if needed.

We conducted numerical experiments on different variants of COLS, such as COLS with QR decomposition, COLS with Chelosky factorization and hybrid COLS. COLS with Chelosky factorization and hybrid COLS achieved similar computational performance and they are about 30% faster than COLS with QR decomposition. We also compared the COLS with Gurobi Simplex solvers: COLS performs much better in comparision with the Gurobi primal simplex solver and outperforms the Gurobi dual simplex solver.

The second part of this research work made contributions to primal dual simplex methods in solving multicommodity flow problems.

Primal dual simplex methods are very successful in solving large-scale set partitioning problems, especially in the airline industry. Barnhart [5] used it to solve multicommodity flow problems and Wang [28] explored variants of multi-pair shortest path algorithms to improve Barnhart's approach.

In primal dual simplex methods, a feasible dual solution is maintained throughout iterations and a restricted master problem is formed in each iteration. A RMP is solved to optimality and its dual optimal solution is used together with the feasible dual solution to find a step size so that a new feasible dual solution can be calculated. These steps are repeated in each iteration until a global optimal solution is found. When we were solving the multicommodity flow problem, which was formulated as a set packing problem with convexity constraints, we noticed that the dual variables associated with the set packing constraints π and the dual variables associated with the convexity constraints σ can be processed separately: given any $\pi \leq 0$, we can find a dual feasible solution (π, σ) if a corresponding σ is selected. For different pairs of π and σ, we have different dual objective values.

Based on these observations, we proposed a long step primal dual simplex method, in which the step size was determined as a function of the feasible dual solution and the RMP optimal dual solution. We discovered that it was a piecewise linear concave function and proposed an efficient way to find the optimal step size that maximizes dual objective.

Because larger dual objective improvements may lead to less iterations in the primal dual simplex method, we expect the long step primal dual simplex method will converge faster than the corresponding primal dual simplex method, even though we spent a little more time in calculating the optimal step size in each iteration.

We conducted extensive computational experiments on long step primal dual simplex methods and achieved significant improvement over primal dual simplex methods.

We used the long step primal dual simplex method to solve large-scale multicommodity flow problems and solved two previous unsolved largest cases, Planar2500 and a Chicago-Region problem, to global optimality for the first time. The Chicago-Region problem has over two million commodities and we used a tree/sub-network based formulation that reduced the number of rows but significantly increased the number of columns in the formulation and this was solved successfully. In order to improve computational performance, we used row generation to include only active arcs in the multicommodity flow network and column generation in various steps of the long step primal dual simplex method, such as optimal step size calculation.

Dantzig-Wolfe decomposition approaches are very efficient in solving multicommodity problems and consistently outperform primal dual simplex methods by a factor of 1.5 to 3 in computational performance based on past studies. Our long step primal dual simplex method, which is significantly faster than the corresponding primal dual simplex method, outperforms Dantzig-Wolfe decomposition on large-scale cases.

7.2 Future Research

In this section, we propose some possible directions for future research.

In solving integer programming problems, cutting planes may be generated as new constraints are added after the optimal solution is found. Dual simplex methods are very suitable in such cases, but primal simplex methods have troubles. COLS approaches resemble primal simplex methods with no straightforward ways to make good use of the existing optimal solution. However, COLS approaches move outside the feasible region until the last step and the new cutting plane will move the current solution back into the infeasible region. It will be very interesting to explore an efficient way for COLS to return to the optimal solution so that it can be used efficiently in cutting plane approaches.

In our proposed hybrid approach, iterative refinement steps can be used to improve numerical stability. However, we did not use this approach because our matrices were not ill-conditioned. Further research on iterative refinement steps for semi-normal equations can help us understand better its impact on computational performance and stability.

We applied the long step primal dual simplex method to solve large-scale multicommodity flow problems that have been formulated as a set packing problem with convexity constraints. This approach can also be applied to other formations such as set partitioning/covering problems with convexity constraints. It is possible to apply long step primal dual simplex methods to solve other challenging problems with convexity constraints. Additionally, for problems with features similar to convexity constraints, we may also be able to discover a variant of the long step primal dual simplex approach to find optimal dual steps efficiently.

Currently, we set a positive threshold for primal dual simplex methods. The resulting solution may not be optimal if some active columns have positive reduced costs thus a phase II approach is necessary to bring it to global optimality. If we set the threshold to 0, the phase I solution will be optimal and further research can explore the impact of the optimal step size on overall convergence rates.

The MCF case Chicago-Region with 2 million commodities is the largest multicommodity flow problem we can find in the literature. In future, we may find or create larger multicommodity flow problems with more commodities to solve so we can further explore this work area. For larger cases, we can apply parallel and distributed computation to solve them more efficiently. All pair shortest path problems are suitable candidates to be parallelized.

REFERENCES

[1] Ahuja, R., Magnanti, T., and Orlin, J., *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall.

[2] Babonneau, F., *Solving the multicommodity flow problem with the analytic center cutting plane method.* PhD thesis, University of Geneva, 2006.

[3] Babonneau, F., du Merle, O., and Vial, J.-P., "Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-accpm," *Operations Research*, vol. 54, no. 1, pp. 184–197, 2006.

[4] Barnes, E., Chen, V., Gopalakrishnan, B., and Johnson, E., "A least square primal-dual algorithm for solving linear programming problems," *Operations Research Letters*, vol. 30, 2002.

[5] Barnhart, C., *A Network-based Primal-Dual Solution Methodology for the Multi-Commodity Network Flow Problem.* PhD thesis, Massachusetts Institute of Technology, Department of Civil Engineering, 1988.

[6] Barnhart, C., *Airline Schedule Planning.* MIT OpenCourseWare 1.206J/16.77J/ESD.215J, Spring 2003.

[7] Bjorck, A., *Numerical Methods for Least Squares Problems.* Philadelphia, PA: SIAM, 1996.

[8] Bompadre, A. and Orlin, J. B., "A simple method for improving the primal simplex method for the multicommodity flow problem," *Networks - Special Issue In Memory of Stefano Pallottino*, vol. 51, no. 1, pp. 63–77, 2008.

[9] Davis, T. A. and Hager, W. W., "Row modifications of a sparse cholesky factorization," *SIAM. J. Matrix Anal. Appl.*, vol. 26, pp. 621–639, 2005.

[10] Davis, T. A., *Direct Methods for Sparse Linear Systems.* Philadelphia PA: SIAM, 2006.

[11] Golub, G., "Numerical methods for solving linear least squares problems," *Numerische Mathematik*, vol. 7, no. 3, pp. 206–216, 1965.

[12] Golub, G. and Kahan, W., "Calculating the singular values and pseudo-inverse of a matrix," *Journal of SIAM: Series B, Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[13] Golub, G. H. and Loan, C. F. V., *Matrix Computations.* Baltimore and London: The Johns Hopkins University Press, 1996. Third Edition.

[14] Gondzio, J. and Terlaky, T., *A computational view of interior-point methods for linear programming.* University Press, 1994.

[15] Gopalakrishnan, B., *Least-square methods for Solving Linear Programming Problems.* PhD thesis, Georgia Institute of Technology, June 2002.

[16] Hu, J., *Solving Linear Programs Using Primal-dual Subproblem Simplex Method and Quasi-explicit Matrices.* PhD thesis, Georgia Institute of Technology, June 1996.

[17] Hu, J. and Johnson, E., "Computational results with a primal-dual subproblem simplex method," *Operations Research Letters*, vol. 25, 1999.

[18] Kong, S., *Linear Programming Algorithms using Least Squares Method.* PhD thesis, Georgia Institute of Technology, May 2007.

[19] Kuhn, H., "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, 1955.

[20] Larsson, T. and Yuan, D., "An augmented lagrangian algorithm for large scale multicommodity routing," *Computational Optimization and Applications*, vol. 27, no. 2, pp. 187–215, 2004.

[21] Lawson, C. L. and Hanson, R. J., *Solving Least Squares Problems.* Philadelphia, Pennsovinia: SIAM, 1995. This SIAM edition is an unabridged, revised republication of the work first published by Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1974.

[22] Leichner, S., Dantzig, S. A., and Davis, J., "A strictly improving linear programming phase 1 algorithm," *Annals of Operations Research*, vol. 47, 1993.

[23] Papadimitriou, C. H. and Steiglitz, K., *Combinatorial Optimization Algorithms and Complexity.* Mineola, New York: Dover Publications, Inc., 1998.

[24] Rothberg, E. and Hendrickson, B., "Sparse matrix ordering methods for interior point linear programming," *INFORMS Journal on Computing*, vol. 10, no. 1, pp. 107–113, 1998.

[25] Stewart, G., *Matrix Algorithms Volume I: Basic Decompositions.* Philadelphia PA: SIAM, 1998.

[26] Stewart, G., *Matrix Algorithms Volume II: Eigensystems.* Philadelphia PA: SIAM, 2001.

[27] Trefethen, L. N. and David Bau, I., *Numerical Linear Algebra.* Philadelphia PA: SIAM, 1997.

[28] Wang, I., *Shortest Paths and Multicommodity Network Flows.* PhD thesis, Georgia Institute of Technology, April 2003.