DYNAMIC CONSTRAINT OPTIMAL SELECTION TECHNIQUES FOR LINEAR PROGRAMMING

by

ALIREZA NOROZIROSHAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

April 2016

To my beloved ones

Acknowledgements

I take this opportunity to sincerely express my gratitude to professor Dr. H.W. Corley for chairing my committee and supervising my dissertation. I am indebted to him for his whole-hearted support, enthusiasm, and inspiration throughout my graduate studies. I am grateful to Dr. Jay M. Rosenberger for his valuable advice and suggestions during my studies and for his participation in my committee. His courses have benefited me greatly during my doctoral studies. I also sincerely appreciate Dr. Victoria Chen's support as Interim Chair of IMSE and for her interest and time on the committee. In addition, I am very thankful to Richard Zercher for his support and guidance and for constantly running CPLEX on the workstations. Finally I am grateful to the students of COSMOS lab for creating a positive learning environment.

April 15, 2016

Abstract

DYNAMIC CONSTRAINT OPTIMAL SELECTION TECHNIQUES FOR LINEAR PROGRAMMING

Alireza Noroziroshan, PhD

The University of Texas at Arlington, 2016

Supervising Professor: H.W. Corley

Linear programming has been studied for over 60 years. It has been considered as one of the most valuable optimization tool for many industrial problems. The simplex algorithm remains the predominant approach to solving linear programming problems. Here we use the simplex method in an active-set frame work to improve it substantially. In general an active-set method obtains solutions by adding one or more problem constraints at a time to solve smaller problems iteratively. In particular, some of these methods have proven to perform significantly faster than the simplex method. In this dissertation we proposed an efficient constraint selection metric for NNLPs called NVRAD to add constraints recursively in two ways; using posterior method and dynamic active-set approach for both nonnegative linear programming and general linear programming. In general linear programming we improve on past prior active-set methods by using dynamic constraint selection technique. These innovations improved the solver's performance and reduced the computation time needed to solve large-scale linear programming problems.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

1.1 The Linear Programming Problem

Linear programming algorithms have been studied for over sixty years and it has been considered as an optimization tool for several problems. Consider the following linear programming problem P:

$$\text{(P)} \qquad Max \;\; z = \boldsymbol{c}^T \boldsymbol{x} \tag{1.1}$$

$$s.t.$$

$$\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \tag{1.2}$$

$$\boldsymbol{x} \geq \boldsymbol{0}, \tag{1.3}$$

where $\boldsymbol{x}$ and $\boldsymbol{c}$ are the n-dimensional column vectors of variables and objective coefficients, respectively, and z represents the objective function. The matrix $\boldsymbol{A}$ is an $m \times n$ matrix $[a_{ij}]$ with row vectors $\boldsymbol{a}_1, \dots, \boldsymbol{a}_m$; $\boldsymbol{b}$ is an m-dimensional column vector; and $\boldsymbol{0}$ is an n-dimensional column vector of zeros. The non-polynomial simplex methods and polynomial interior-point barrier-function algorithms illustrate the two different approaches to solve problem P. There is no single best algorithm [1]. For any existing approach, there is a problem instance for which the developed method performs poorly [2], [3]. However, interior point methods do not provide efficient post-optimality analysis, so the simplex algorithm is the most frequently used approach [2], even for sparse large scale linear programming problems where barrier methods perform extremely well. In fact, the simplex method has been called “the algorithm that runs the world” [4], yet it often cannot efficiently solve the large scale LPs required in many applications.

We consider both the general linear program (LP) and the special case with $\boldsymbol{a_i} \geq \boldsymbol{0}$ and $\boldsymbol{a_i} \neq \boldsymbol{0}, \forall i = 1, \dots, m; \boldsymbol{b} > \boldsymbol{0}$; and $\boldsymbol{c} > \boldsymbol{0}$, which is called a nonnegative linear program (NNLP). NNLPs have some useful properties that simplify their solution, and they model

various practical applications such as determining an optimal driving route using global positioning data [5] and updating airline schedules [6], for example.

After introducing the simplex method by George B.Dantzig in 1949, many individuals have conducted theoretical and computational research in the field of linear programming. For a wide range of problems, the simplex algorithm is capable of producing solution in a reasonable amount of time and it allows an efficient post optimality analysis. Simplex is known as a combinatorial algorithm in which the problem complexity increases by the order of the problem size. However, in practice, simplex is able to solve problems with complexity proportion to $n + m$.

Since the principle use of LP in industrial applications is in binary and integer programming algorithms, however, pivoting algorithms with efficient post-optimality analysis are frequently preferable to interior-point methods. On the other hand, simplex methods often cannot solve large-scale LPs at a speed required by many current applications.

In many linear programming problems, the majority of constraints will be redundant and they won't bind at optimality. Active-set method is a method in which solution is achieved by adding one or more constraints at a time to solve small subset of problems iteratively. Our approach divides the constraints of problem P into operative and inoperative constraints at each iteration. Operative constraints are those active in a current relaxed subproblem $P_r,\ r = 1,2, \dots, \text{of } P$, while the inoperative ones are constraints of the problem P not active in P_r. In our active-set method we iteratively solve $P_r,\ r = 1,2, \dots, \text{of } P$ after adding one or more violated inoperative constraints from (1.2) to P_{r-1} until the solution $\boldsymbol{x}_r^*$ to P_r is a solution to P.

1.2 Objectives of the Work

This research focuses on developing new active-set methods as well as new constraint selection metric that outperform the current existing active-set methods in solving both non-negative linear programming and general linear programming problems. Each new developed method is termed a Constraint Optimal Selection Technique (COST). COSTs can be categorized in two main classes, prior and posterior. Prior COSTs only utilize the global information of a subset of constraints (relaxed LP problems) while posterior COSTs use current optimal solutions $\boldsymbol{x}_r^*$ of the relaxed problem to measure the likelihood of binding constraint at optimality. A constraint selection metric and two dynamic active-set approaches are developed, implemented and tested. The efficiency of the proposed algorithms is tested by several test problems. For the majority of the problems, the developed methods reveal superior performance than the existing methods in large-scale optimization experiments.

1.3 Brief Description of COSTs

All constraints are sorted with respect to their probability of binding at optimal solution by using a certain constraint selection metric. Our approach divides the constraints of problem P into operative and inoperative constraints at each iteration. Operative constraints are those active in a current relaxed sub-problem $P_r,\ r = 1,2,\dots,\text{of } P$, while the inoperative ones are constraints of the problem P not active in P_r. In our active-set method we iteratively solve $P_r,\ r = 1,2,\dots,\text{of } P$ after adding one or more violated inoperative constraints from (1.2) to P_{r-1} until the solution $\boldsymbol{x}_r^*$ to P_r is a solution to P.

In order to maintain appropriate level of progress we propose a dynamic method that adds a varying number of constraints to P_r that depends on the progress made at P_{r-1}. No equality constraints are considered here, but any equality constraints can be included in P_0. An active-set function is defined to compensate for the lack of progress in P_{r-1} by

adding more violated constraints at P_r. The algorithm stops when the solution $\boldsymbol{x}_r^*$ to P_r is the optimal solution to P.

1.4 Contribution

In this research, five different methods are developed to enhance the efficiency of the current linear programming algorithms. The developed techniques are as follows.

I. Dynamic active-set method for NNLPs

II. $NVRAD(\boldsymbol{a_i}, b_i, \boldsymbol{c}) = \frac{\boldsymbol{a_i}^T \boldsymbol{c}}{b_i^2}(\boldsymbol{a_i x}^* - b_i)$

III. Dynamic active-set method for GLPs

IV. Dynamic column generation using $\text{DRAD}\left(\boldsymbol{a}^{j^T}, \boldsymbol{b}, c_j\right) = \left\{\frac{\boldsymbol{a}^{j^T}\boldsymbol{b}}{c_j} \mid \boldsymbol{a}^{j^T}\boldsymbol{y}_r^* < c_j\right\}$

V. Dynamic COST $\left(\boldsymbol{a}^{j^T}, \boldsymbol{b}, c_j\right) = \left\{\frac{\boldsymbol{a}^{j^T}\boldsymbol{b}}{c_j}\right\}$ for dual of P

The main contributions of this research are

i) A new technique of dynamically adding multiple cuts at P_r based on the obtained progress at P_{r-1} is incorporated into the NVRAD and GRAD for NNLPs and GLPs respectively.

ii) Introducing a posterior constraint selection metric $\text{NVRAD}(\mathbf{a}_i, \text{b}_i, \mathbf{c}, \boldsymbol{x}^*) = \frac{\boldsymbol{a}_i^T \boldsymbol{c}}{b_i^2}(\boldsymbol{a}_i \boldsymbol{x}^* - b_i)$.

iii) A new dynamic active-set method is incorporated into the NVRAD for NNLPs.

iv) Implementing a dynamic active-set method to increase the efficiency of the GLPs.

v) Creating sets of randomly generated non-negative linear programming and general linear programming problems and evaluating the performance of the developed prior and posterior COST methods on these problems.

vi) Using DRAD as a Column generation method

vii) Using DRAD COST as a constraint selection metric in Dual of P.

viii) Experimenting CPLEX and our method for equality constraints and comparing the results.

1.5 Overview of the Dissertation

This thesis is organized into seven chapters. Chapter 2 includes some essential definitions, keywords and basic concepts of optimization techniques as well as brief explanation on theoretical background, related works and historical perspective. A dynamic active set method for general linear programming is discussed in Chapter 3 and the results are compared with the existing methods for GLPs. In Chapter 4, NVRAD is introduced as a constraint selection metric for NNLPs along with its geometric interpretation. In the same chapter, the proposed method is applied on randomly generated NNLP problems and the importance of posterior methods is discussed in detail. Chapter 5 contains discussions on variable entering rules and column generation techniques in linear program problems. A new column generation method is developed and the results are compared with the Sifting method on test problems. Chapter 6 discusses a method for solving large-scale linear programing problems with equality constraints. The conclusion and the possible future works of this study are discussed in Chapter 7. Chapter 3, 4, and 5 each represents journal papers either published or submitted.

CHAPTER 2

Background

2.1 Introduction

The preliminary definitions and general aspects of linear programming are briefly summarized in this chapter.

2.2 Non-negative Linear Programming

Nonnegative linear programing problem (NNLP), which is the special case of P with $\boldsymbol{a_i} \geq \mathbf{0}$ but $\boldsymbol{a_i} \neq \mathbf{0}, \forall i = 1, \dots, m$; $\boldsymbol{b} > \mathbf{0}$; and $\boldsymbol{c} > \mathbf{0}$. NNLPs model a large portion of linear programming applications such as determining the optimal driving path for navigation systems using traffic data [5], updating flight status due to the variations occurring in passenger loads or weather conditions [2], and detecting common errors in DNA sequences [7]. NNLPs have the following two important properties.

- The origin $\boldsymbol{x} = \mathbf{0}$ is feasible,
- $x_j \leq \min_{i=1,\dots m} \left\{ \frac{b_i}{a_{ij}} : a_{ij} > 0 \right\}, \forall j = 1, \dots, n$.

Thus NNLPs have both a bounded feasible region and a bounded objective function if and only if no column of $\boldsymbol{A}$ is a zero vector, and so their boundedness is easily verifiable without computation.

Active-set methods have been studied by Stone [8], Thompson et al. [9], Adler et al. [10], Zeleny [11], Myers and Shih [12], Curet [13], and Bixby et al. [1], among others. The term ‘‘constraint selection technique’’ was introduced in [12], while the approaches of [10] and [8] illustrate two distinct classes of active-set methods. When the constraint selection metric for choosing violated inoperative constraints to be added to P_r does not depend on the solution $\boldsymbol{x}_r^*$, the associated active-set method is called a prior method. On the other hand, if the constraint selection at P_r does depend on $\boldsymbol{x}_r^*$, it is called a posterior

method. Adler et al. [10] developed a prior method in which a violated inoperative constraint was chosen randomly at each iteration. Zeleny [11] proposed a posterior method in which the inoperative constraint most violated by $\boldsymbol{x}_r^*$ was added. This posterior method is called VIOL here. VIOL is also used as a pricing rule in delayed column generation [14], for adding multiple constraints in the interior point cutting plane method of [15], and in the sifting algorithm of [1] for column generation.

2.3 Constraint Selection Metric

An efficient constraint selection metric plays a pivotal rule in selecting the potential rows that are more likely to be active at optimality. Constraint selection metric can be categorized as prior or posterior methods. Some of the constraint selection criterions are as follows:

- SUB (Prior COST)
- COS (Prior COST)
- VIOL (Posterior COST)
- NVIOL (Posterior COST)
- RAD (Prior COST)
- VRAD (Posterior COST)
- GRAD (Prior COST)

In prior method, all the constraints are sorted only once before the solver starts but unlike prior methods, posterior methods depend on $\boldsymbol{x}_r^*$ and require extra processing for every iteration [16]. Different constraint selection metrics place a different priority on each constraint and consequently each method leads to a different performance. Hence, there is an interest in development of a more efficient constraint selection metric to measure the likelihood of the binding rows at optimality. Based on different criteria, constraints can be ordered either preliminary or postliminary by a certain amount of likelihood of being binding

at optimal solution. In the following, some of the constraint selection rules are described in more detail.

2.3.1 SUB

In SUB method, no constraint selection rule is used and the violated constraints are added in the same order they appeared in the problem. Therefore, it does not require any preprocessing [10].

2.3.2 COS

Cosine (COS) constraint selection metric focuses on computing the angle between normal vector $\boldsymbol{c}$ and normal vector of $\boldsymbol{a}_i$ which can be obtained by $cos\ (\theta_i) = \frac{\boldsymbol{a}_i{}^T \boldsymbol{c}}{||\boldsymbol{a}_i|| * ||\boldsymbol{c}||}$ [17]. The constraint that creates smaller θ has a higher chance to be binding at optimality. The efficiency of COS reduces in problem with low density by not getting the total impact of the $\boldsymbol{c}$ vector in every dimension. Therefore, in low density problems, the value of $cos\ (\theta)$ approaches to a small number which implies that the formed angle between normal vector c and normal vector of $\boldsymbol{a}_i$ is within the range of $80'$ to $90'$, therefore COS metric may not provide that much information. Figure2.1 illustrates the constraint selection rule based on maximum cosine value.

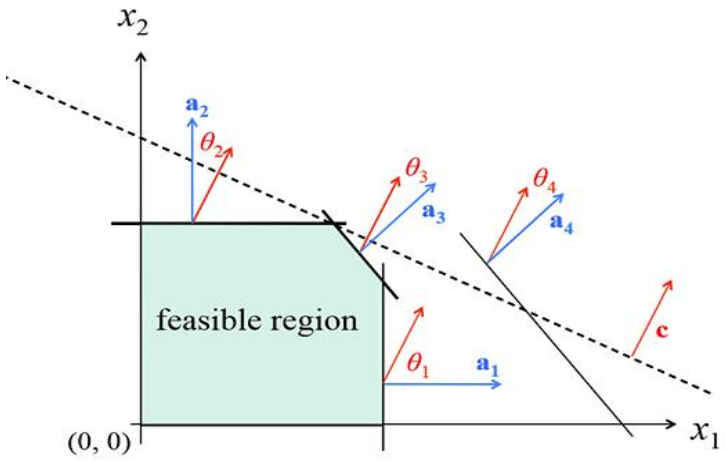

Figure 2.1 Constraint selection technique based on maximum cosine value

In addition, COS does not consider the depth of the feasible region removed by constraints. So two hyper-plans forming a same angle with normal vector of $\boldsymbol{c}$ might be located in completely different positions have the same priority. Problem density can be obtained by (2.1).

$$\text{Problem density} = \frac{\text{Number of non zero elements in matrix } \mathbf{A}}{\text{Number of Rows} \ * \ \text{Number of Columns}} \tag{2.1}$$

2.3.3 VIOL

Zeleny [11] used a constraint selection rule which added the constraint that most violated at each iteration – the method we called VIOL. Let's define $\frac{b_i \boldsymbol{a_i}}{\boldsymbol{a_i}^T \boldsymbol{a_i}}$ an "a-point" on constraint i. $\boldsymbol{x_r^*} - \frac{b_i}{\boldsymbol{a_i}^T \boldsymbol{a_i}} \boldsymbol{a_i}$ is a vector from "a-point" to $\boldsymbol{x_r^*}$ and the scalar product of normal vector of $\boldsymbol{a_i}$ with vector $\boldsymbol{x_r^*} - \frac{b_i}{\boldsymbol{a_i}^T \boldsymbol{a_i}} \boldsymbol{a_i}$ can provide a geometric interpretation of violation. Violation can be described as the projection of a vector from "a- point" to current optimal solution $\boldsymbol{x_r^*}$ on the constraint normal vectors. The ranking of the constraints are from maximum to minimum value for VIOL which as follows:

$$VIOL(\boldsymbol{a_i}, b_i, \boldsymbol{x_r^*}) = \boldsymbol{a_i}^T \boldsymbol{x_r^*} - b > 0,$$

where $i^* \in argmax(VIOL\ (\boldsymbol{a_i}, b_i, \boldsymbol{x_r^*})$: $\boldsymbol{a_i}^T \boldsymbol{x_r^*} > b_i$; $i \notin OPERATIVE)$. Figure 2.2 depicts the geometric description of VIOL criteria.

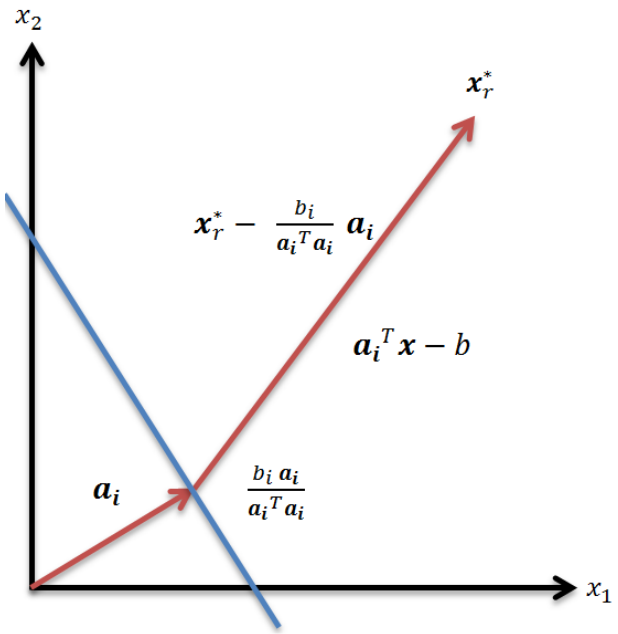

Figure 2.2 Geometric description of VIOL

2.3.4 RAD

If the feasible space looks like a sphere to an n-dimension space and objective function vector moves toward objective gradient until hits the constraints. By moving in $\boldsymbol{c}$ direction toward constraint we have $\boldsymbol{a}_i{}^T(k\boldsymbol{c}) = b$. Then, scalar k is obtained by $\frac{b_i}{\boldsymbol{a}_i{}^T\boldsymbol{c}}$. As can be seen from (2.5), RAD measures the likelihood of active constraints at optimal point as a combination of two factors.

$$RAD = \frac{\boldsymbol{a}_i{}^T\boldsymbol{c}}{b_i} = \frac{||\boldsymbol{a}_i||}{b_i}\frac{\boldsymbol{a}_i{}^T\boldsymbol{c}}{||\boldsymbol{a}_i|| * ||\boldsymbol{c}||}||\boldsymbol{c}|| \propto \frac{||\boldsymbol{a}_i||}{b_i}\cos(\boldsymbol{a}_i, \boldsymbol{c}) \tag{2.2}$$

The first factor is the depth of the feasible region that removes and the second factor is the angle of its normal vector $\boldsymbol{a}_i$ with objective vector $\boldsymbol{c}$. All constraints are sorted in a descending order of RAD. Figure 2.3 presents the geometric interpretation of RAD constraint selection criteria.

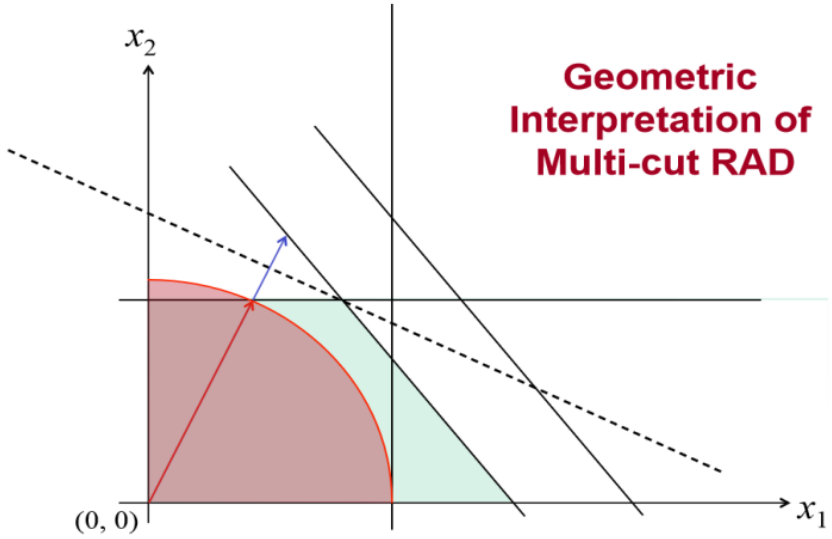

Figure 2.3 Geometric interpretation of RAD

However, for problems with a low density, which $\boldsymbol{a_i}$ contains some zeros in certain variables, adding a single constraint $\boldsymbol{Ax} \leq b$ may not be efficient enough. In the problem with high density RAD is able to effectively prioritize the constraints since it captures more information from every dimension. However, in a problem with a low density because of not getting the total impact of the objective vector the RAD's efficiency reduces. Adding multiple cutting planes in which forming a dome-shape polytope are more efficient than adding a single plane [6]. The efficient active-set approaches developed by using RAD metric are:

- NRAD: Non-negative linear programming [6]
- GRAD: General linear programming [18]

Adding multiple constraints to p_r to bound all the variables forms a geodesic-like dome which leads to cutting off the $\boldsymbol{x}_r^*$ more effectively, but in a high density problems, P_r is bounded at each iteration by adding few cuts. Therefore, the multi cut approach might perform somehow similar to a single cut method on high density problems.

2.3.5 VRAD

The VRAD is a posterior constraint selection metric which considers the distance of $\boldsymbol{x}_r^*$ to the violated $\boldsymbol{a}_i{}^T\boldsymbol{x}_r^* - b$ along with RAD [16]. The constraint selection metric

$$VRAD(\mathbf{a_i}, \mathrm{b}_i, \mathbf{c}, \boldsymbol{x_r^*}) = \frac{\boldsymbol{a_i}{}^T\boldsymbol{c}}{b_i} * \frac{(\boldsymbol{a_i}{}^T\boldsymbol{x}_r^* - b)}{||\boldsymbol{a}_i||} \tag{2.3}$$

Where $i^* \in argmax(VRAD\ (\boldsymbol{a_i}, b_i, \mathbf{c}, \boldsymbol{x}_r^*): \boldsymbol{a_i}{}^T\boldsymbol{x}_r^* > b_i\ ; i \notin OPERATIVE)$

2.3.6 GRAD

Saito et al. [18] developed a constraint selection metric for general linear programming problem which as follows:

$$GRAD\ (\boldsymbol{a}_i, b_i, \boldsymbol{c}) = \sum_{j=1, c_j>0}^{n} \frac{a_{ij}\ c_j}{b_i{}^+} - \sum_{j=1, c_j<0}^{n} \frac{-a_{ij}}{b_i{}^+}, \tag{2.4}$$

$$where\ b_i{}^+ = \begin{cases} b_i - \min\limits_{k=1,\dots,m}[b_k] + \varepsilon & if \min\limits_{k=1,\dots,m}[b_k] < 0 \\ b_i & Otherwise \end{cases} \tag{2.5}$$

for a small positive constant ε, and $i^* \in argmax(GRAD\ (\boldsymbol{a}_i, b_i, \boldsymbol{c}): \boldsymbol{a}_i{}^T\boldsymbol{x}_r^* > b_i\ , i \notin OPERATIVE\ )$. The first term invokes the shortest distance of constraint in positive c_j directions to $\boldsymbol{a}_i^T\ \boldsymbol{x}\ =\ b_i$ and second term represents the general closeness rate of a constraint to the origin for all negative c_j directions. For a_j with $c_j < 0$, it is hard to measure the likelihood of the binding constraint at optimality since the intersection of multiple c vector, and $\boldsymbol{a}_i^T\ \boldsymbol{x} =\ b_i$ may not lie in the feasible region. Therefore, all c_j is set to be -1.

2.4 General Approach to Active-Set

Active-set method divides the problem constraints ($\boldsymbol{A}$ matrix) into two groups of operative and inoperative sets and problem is solved iteratively by ignoring the inoperative constraints and updating the operative set at each iteration. The main purpose of using any such active-set method is that a solution to the main problem P can be obtained by solving a sequence of relatively small sub problems P_r. Saito et al. [18] developed an active-set

approach for NNLPs. To construct P_0, we choose constraints from (1.2) in descending order of RAD until each variables x_j has an $a_{ij} > 0$ in the coefficient matrix of P_0. We say the variables are covered by the constraints of the initial problem P_0. It significantly increases the efficiency of constraint selection criteria RAD for non-negative linear programming. The main drawback with covering technique is the method might add only few constraints at each iteration to cover all columns in P_r for a problem with relatively high density.

In order to implement a new active-set approach for solving general linear programming problems Saito et al. [18] proposed to add an artificial bounding constraint $\boldsymbol{c}^T\boldsymbol{x} \leq M$ for the first step. Then P_0 is formed by adding multiple constraints from (1.2) sorted by descending order of GRAD until all the columns of the new $\boldsymbol{A}$ matrix have at least one negative and one positive coefficient. The primal simplex method is used to solve P_0 and dual simplex is employed to solve next active set iterations P_r.

2.5 Historical Perspective

The history of the development and research on linear programming is shown in the following.

The idea of LP goes back into 1824, Leonid Kantorovich, a Soviet mathematician formulate LP but his work remain unknown until 1950s [19]. There was an interest in applying the optimization techniques for resource allocation and operation planning during wartime [20]. G.B. Dantzig developed the simplex method on 1947 and the following year he published his work “Programming in a Linear Structure “. New topics such as linear programming under uncertainty, network optimization and mathematical programming and began to emerge on 1950 [21].

Dantzig decomposition method for solving large-scale problem was introduced by Dantzig in the late 1950. The idea is inspired from Minkowski and Weyl’s work on convex polyhedral sets. On 1955 Dantzig, pointed a way to find a solution to a large size integer

programming problems (TSP) and several combinatorial optimization problems. Dantzig and Beale independently worked on stochastic programming [20]. Gomory developed his cutting planes method for solving integer programming problems on 1958. The dual version of Dantzig-Wolfe decomposition which is Benders decomposition was proposed and published by Benders on 1962. Interior point method using ellipsoids developed by Khachiyan It was the first polynomial-time algorithm for solving LP [22]. In 1984 Karmarkar developed an interior point method with polynomial time complexity for solving linear programming.

The simplex method searching for the optimal solution by traversing a series of basic feasible point and changing a basic variable at each iteration. Simplex method is considered as a combinatorial algorithm since there are finite set of extreme points in each linear programming problem. Interior point method was developed by John von Neumann [23]. It is a certain class of optimization method that achieves optimality by moving through the feasible direction which maximize the objective function per unit distance, rather than moving around the boundary. There are three categories of interior point method algorithms:

1) Potential reduction algorithm
2) Affine scaling algorithm
3) Path following algorithm

The primal-dual algorithm interior point method is one of the most efficient path following algorithms that simultaneously operates on primal and dual space [24].

2.6 Large-Scale Linear Programming

Linear programming has been widely applied to several scientific and industrial problems. Many such problems have a particular form and structure in which the positive elements of $\boldsymbol{A}$ matrix representing block angular structure. By utilizing some of the

decomposition techniques we are able to convert large size problems in to the smaller sub-problems. The decomposition technique can improve the computational time of the linear programing problems, which their $\boldsymbol{A}$ matrix has a certain structure. The master problem contains general constraints and sub-problem includes special structured constraints and then the strategy operates on both master and sub-problem by passing the information back and forth until the optimal solution to the main problem is achieved. Dantzig-Wolfe and Benders decomposition are two types of these decomposition techniques [1].

The idea of partitioning methods and decomposition techniques for solving large-scale optimization problems can be described as follows:

1. In a problem with $(m >> n)$ excessive number of row, a constraint is only created and then added if it is violated by the current optimal solution. The constraint selection techniques are used to determine the priority of the violated constraints.
2. In a problem with short and wide structure $(n >> m)$ a column is only generated if its reduced cost is negative. The generated column is then added.

2.6.1 Delayed Column Generation

Delayed column generation is one of the most efficient methods for solving large scale linear programs. The main idea of column generation is that many linear programming problems have short and wide structure and the numbers of variables are too large to consider all the variables in $\boldsymbol{A}$ matrix. Meanwhile only a sub set of variables need to be considered when solving the problem. It considers generating a column if only it has been discovered that it can profitably enter the basis. In delayed column generation the linear programming problem is divided into two problems:

1. Master problem
2. Sub-problem

The master problem contains subset of variables from original problem that created to discover the favorable columns to enter the basis. Delayed constraint generation is the dual of delayed column generation [25].

2.6.2 Sifting (Sprint)

Sifting algorithm is a column generation technique for linear programming problems and it was developed by John Forrest and it was applied on airline crew scheduling problem. The problem under study was a set partitioning problem with 837 rows and 12,750,000 columns [1].

2.7 Pivoting Rule

A pivoting rule plays an important role in the simplex method. It guaranties the improvement of the objective function in each iteration. Dantzig's pivoting rule is commonly used in simplex method. Let N be the non-basis and B the associated basis at the current iteration of the simplex algorithm. The non-basic variable's reduced costs can be achieved by a pricing operation

$$\bar{\boldsymbol{c}}_N = \boldsymbol{c}_N - N^T \boldsymbol{\pi}, \qquad \boldsymbol{\pi} = \boldsymbol{c}_B B^{-1}. \tag{2.6}$$

Optimality is obtained if the index set is empty for $\forall\, j = 1,..,n$

$$J = \{j \mid \bar{\boldsymbol{c}}_j < 0, j \in N\} \tag{2.7}$$

otherwise, any index in J (has negative reduced cost) can improve the objective function.

2.7.1 Full Pricing

The full pricing rule considers all non-basic variables' reduced cost at each iteration and selects the variable in which decreases or increases the objective value the most. There exist some other standard frequently used pricing rules developed and tested and the computational results were encouraging in several problem samples compared to original Dantzig's rule which are as follow:

1. Largest decrees rule

2. Dantzig's rule
3. Steepest-edge rule
4. Devex-rule
5. Largest distance
6. Nested pricing

The nested pricing and largest distance rule yield run times reduced by a factor of 5.73 and 3.24 respectively, compared to the Devex rule [26]. The direct cosine simplex algorithm (DCA) is developed by [27] to improve the simplex algorithm. The proposed algorithm defines a set of variables with negative reduced cost and among those DCA pivots on the variable that has lower cosine value. All candidate variables with negative reduce cost, are required to calculate the corresponding $cos^*(\theta_i)$ value by (2.8).

$$cos^*(\theta_i) = \left[\frac{(\boldsymbol{A}^T)_i \,.\, \boldsymbol{B}^T}{||(\boldsymbol{A}^T)_i||}\right] = \frac{\sum_{j=1}^{m} aji}{\sum_{j=1}^{m}(aji)^2} \tag{2.8}$$

where $\boldsymbol{B} = [b_1\,, b_2, \ldots, b_m]^T$ and $\boldsymbol{A_i} = [a_{i1}, a_{i2}, \ldots, a_{in}]$, is the ith row of matrix $\boldsymbol{A}$ in the dual problem. A variable with the minimal $cos^*(\theta_i)$ is chosen as entering variable. DCA represents the superior performance over Dantzig's rule in most problem samples in computational experiment [27].

2.7.2 Partial Pricing

Partial pricing is a method that only a small proportion of the non-basic variables' are considered as for pricing out process. The potential variable from this set is selected by some metric and enter to the basis and keep the other variables as potential candidates for the next iterations. The process continues until all the reduced costs associated with all variables in the set are nonnegative [26].

CHAPTER 3

Prior Cost Improvement

3.1 Introduction

In this chapter, a dynamic active-set approach for NNLPs is developed and its geometric interpretation is described in detail. Also the performance of GRAD is tested in the dynamic active-set frame work to show the efficiency of utilizing the dynamic active-set method for GLPs.

3.2 Dynamic Active-Set Approach for NNLP

The dynamic active-set approach developed for solving NNLPs is as follows. Constraints are initially ordered by the RAD constraint selection metric (2.2). To construct P_0, we choose constraints from (1.2) in descending order of RAD until each variables x_j has an $a_{ij} > 0$ in the coefficient matrix of P_0. We say the variables are covered by the constraints of the initial problem P_0. P_0 is then solved by the primal simplex to achieve an initial solution $\boldsymbol{x}_0^*$. Now let γ_0 be the number of constraints in the original problem P, and in general let γ_r be the number of constraints of problem P violated by $\boldsymbol{x}_r^*$. At each iteration r, the total number of violated constraints γ_r is computed, and the improvement percentage is calculated by

$$\omega_r = max\left\{0, \left(\frac{\gamma_{r-1} - \gamma_r}{\gamma_{r-1}}\right)\right\} * 100\ , \forall\ r = 1,2, \ldots, \tag{3.1}$$

where $\omega_r > 0$ represents the percent of improvement made in reducing the total number of violated constraints at iteration r. Next, with **[.]** denoting the greatest integer function, let

$$\varphi_{r+1} = [\varphi_r * (1 + \log(101 - \omega_r))]\ ,\ 0 \le \omega_r \le 100\ , \forall\ r = 1,2, \ldots\ , \tag{3.2}$$

where φ_1 = 100. The value φ_r is an upper bound on the possible number of non-operative violated constraints that can be added at active-set iteration $r = 1,2, \ldots$. The actual number

added is $min\{\varphi_{r+1}, \gamma_r\}$. The active-set iterations terminate when $\gamma_r = 0$ and therefore $\omega_r = 100$. Equation (3.2) was developed from the results of computational experiments.

Pseudocode for the dynamic active-set NNLPs is as follows.
Step 1 — Identify constraints to initially bound the problem.
1: $\boldsymbol{a}^* \leftarrow \mathbf{0}$
2: $\boldsymbol{while}\ \boldsymbol{a}^* \not> \mathbf{0}\ \boldsymbol{do}$
3: $Let\ i^* \in \underset{i \notin BOUNDING}{\arg\max}\ RAD(\boldsymbol{a}_i, b_i, \boldsymbol{c})$
4: $\boldsymbol{if}\ \exists j\ |a_j^* = 0\ \boldsymbol{and}\ a_{i*j} > 0\ \boldsymbol{then}$
5: $BOUNDING \leftarrow BOUNDING \cup \{i^*\}$
6: $\boldsymbol{end\ if}$
7: $\boldsymbol{a}^* \leftarrow \boldsymbol{a}^* + \boldsymbol{a}_{i*}$
8: $Optimized \leftarrow \boldsymbol{false}$
9: $\boldsymbol{end\ while}$
Step 2 — Using the primal simplex method, obtain an optimal solution $\boldsymbol{x}_0^*$ for the initial bounded problem P_0

$$maximize\ z = \boldsymbol{c}^T\boldsymbol{x}$$
$$\text{subject to}\ \boldsymbol{a}_i^T\boldsymbol{x} \leq b_i \quad \forall i \in BOUNDING$$
$$\boldsymbol{x} \geq \mathbf{0}$$

Step 3 — Perform the following iterations until an optimal solution to problem P is found.
1: $\varphi_1 \leftarrow \#100$
2: $r \leftarrow 0$
3: $\gamma_0 \leftarrow \#\ rows$
4: $\boldsymbol{while}\ (Optimized = \boldsymbol{false}\ )\ \boldsymbol{do}$
5: $r \leftarrow r + 1$
6: $\boldsymbol{if}\ \{\boldsymbol{a}_i^T\boldsymbol{x}_r^* > b_i\ ,\ \forall\ i = 0, \dots, rows\}\ \boldsymbol{then}$ calculate γ_r
7: Calculate $\omega_r = max\left\{0, \left(\frac{\gamma_{r-1} - \gamma_r}{\gamma_{r-1}}\right)\right\} * 100$
8: $\boldsymbol{if}\ 0 \leq \omega_r < 100\ \ \boldsymbol{then}\ \ \varphi_{r+1} = [\varphi_r * (1 + \log(101 - \omega_r))]$
9: $Let\ i^* \in \underset{i \notin OPERATIVE}{\arg\max}\ \{\mathrm{RAD}(\boldsymbol{a}_i, b_i, \boldsymbol{c}) : \boldsymbol{a}_i{}^T\boldsymbol{x}_r^* > b_i\ \}$
10: $\boldsymbol{for}$ (i=0 to $min\{\varphi_{r+1}, \gamma_r\}$) $OPERATIVE \leftarrow OPERATIVE \cup \{i^*\}\ \boldsymbol{end}$
11: Solve the following P_r by the dual simplex method to obtain $\boldsymbol{x}_r^*$
12: $\boldsymbol{else\ if}(\omega_r = 100)\ \boldsymbol{then}\ \ Optimized \leftarrow \boldsymbol{true}$ // $\boldsymbol{x}_r^*$ is an optimal solution to P.
13: $\boldsymbol{end\ if}$
14: $\boldsymbol{else}\ Optimized \leftarrow \boldsymbol{true}$ // $\boldsymbol{x}_r^*$ is an optimal solution to P.
15: $\boldsymbol{end\ if}$
16: $\boldsymbol{end\ while}$

3.3 Dynamic Active-Set Approach for LP

The dynamic active-set approach for solving LPs is similar to the one for NNLPs. We construct P_0 by choosing a number of constraints ρ_1 from (1.2) in descending order of

GRAD from (2.4) until no variable x_j is left without at least either a positive or a negative coefficient. In addition, we include an artificial bounding constraint $\mathbf{1}^T\boldsymbol{x} \leq M$. If $\rho_1 < 100$, then set $\rho_1 = 100$. Then P_0 is solved to obtain an initial solution $\boldsymbol{x}_0^*$. It is initially assumed that all constraints are violated ($\gamma_0 = m$). Then the relative improvement percent ω_r is calculated by (3.1) for P_r and P_{r+1}. Now let

$$\rho_{r+1} = [\rho_r * \log(101 - \omega_r)], \quad 0 \leq \omega_r \leq 100, \forall r = 1,2,\dots, \tag{3.3}$$

where the value ρ_r is an upper bound on the possible number of non-operative violated constraints that can be added at active-set iteration $r = 1,2,\dots$. The actual number added is $min\{\rho_{r+1}, \gamma_r\}$. As ω decreases, ρ_{r+1} increases in (3.3) to add more violated constraints to P_{r+1}. The algorithm stops at 100% reduction in the number of violated constraints.

Pseudocode for the dynamic active-set for LPs is as follows.

Step 1 — Identify constraints to initially bound the problem.

1: $\boldsymbol{a}^* \leftarrow \mathbf{0}$

2: $\boldsymbol{while}\ \boldsymbol{a}^* \not> 0\ \boldsymbol{do}$

3: $Let\ i^* \in \underset{i \notin BOUNDING}{\arg\max}\ GRAD(\boldsymbol{a}_i, b_i, \boldsymbol{c})$

4: $\boldsymbol{if}\ \exists j\ |a_j^* = 0\ \boldsymbol{and}\ a_{i*j} > 0\ \ or\ \ a_{i*j} < 0\ \boldsymbol{then}$

5: $BOUNDING \leftarrow BOUNDING \cup \{i^*\}$

6: $\boldsymbol{end\ if}$

7: $\boldsymbol{a}^* \leftarrow \boldsymbol{a}^* + \boldsymbol{a}_{i*}$

8: $Optimized \leftarrow \boldsymbol{false}$

9: $\boldsymbol{end\ while}$

Step 2 — Using the primal simplex method, obtain an optimal solution $\boldsymbol{x}_0^*$ for the initial bounded problem P_0 given by

$$maximize\ z\ =\ \boldsymbol{c}^T\boldsymbol{x}$$
$$\text{subject to } \boldsymbol{a}_i^T\boldsymbol{x} \leq\ b_i \quad \forall i\ \in\ BOUNDING$$
$$\mathbf{1}^T\boldsymbol{x} \leq\ M$$
$$\boldsymbol{x}\ \geq\ \mathbf{0}$$

Step 3 — Perform the following iterations until an optimal solution to problem P is found.

1: $\rho_1 \leftarrow \text{Max}\{\#BOUNDING, 100\}$

2: $r\ \leftarrow 1$

3: $\gamma_0 \leftarrow \#\ rows$

4: $\boldsymbol{while}\ (Optimized = \boldsymbol{false})\ \boldsymbol{do}$

5: $r\ \leftarrow r + 1$

6: $\boldsymbol{if}\ \{\boldsymbol{a}_i^T\boldsymbol{x}_r^* > b_i\ ,\ \forall\ i = 0, \ldots, rows\}\ \boldsymbol{then}$ calculate γ_r

7: Calculate $\omega_r =\ max\left\{0, \left(\frac{\gamma_{r-1}-\gamma_r}{\gamma_{r-1}}\right)\right\} * 100$

8: $\boldsymbol{if}\ 0 \leq \omega_r < 100\quad \boldsymbol{then}\quad \rho_{r+1} = [\rho_r * \log(101 - \omega_r)]$

9: $Let\ i^* \in \underset{i \notin OPERATIVE}{\arg\max}\ \{\text{GRAD}(\boldsymbol{a}_i, b_i, \boldsymbol{c}) : \boldsymbol{a}_i{}^T\boldsymbol{x}_r^* > b_i\}$

10: $\boldsymbol{for}$ (i=0 to $min\{\rho_{r+1}, \gamma_r\}$) $OPERATIVE \leftarrow OPERATIVE\ \cup \left\{i^*\right\}\ \boldsymbol{end}$

11: Solve the following P_r by the dual simplex method to obtain $\boldsymbol{x}_r^*$

12: $\boldsymbol{end\ if}$

13: $\boldsymbol{else}\ \ Optimized\ \leftarrow \boldsymbol{true}$ // $\boldsymbol{x}_r^*$ is an optimal solution to P.

14: $\boldsymbol{end\ if}$

15: $\boldsymbol{end\ while}$

3.4 Problem Instances and CPLEX Preprocessing

Four sets of NNLPs used in [6] are considered to evaluate the performance of the developed algorithm. Each problem set contains five problem instances for 21 different density levels and for varying ratios of $(m\ \text{constraints})/(n\ \text{variables})$ from 200 to 1. Each

set contains 105 randomly generated NNLPs with various densities p ranging from 0.005 to 1. Randomly generated real numbers between 1 and 5, 1 and 10, 1 and 10 were assigned to the elements of $\boldsymbol{A}, \boldsymbol{b}$, and $\boldsymbol{c}$ respectively. To avoid having a constraint in the form of an upper bound on a variable, each constraint is required to have at least two non-zero a_{ij}. For general LP, a problem set containing 105 randomly generated by Saito et al. [18] is compared with the dynamic approach of this paper. These LP problems contain 1000 variables (n) and 200,000 constraints (m), with various densities ranging from 0.005 to 1 and the randomly generated a_{ij} ranging between -1 and -5 or between 1 and 5.

Two parameters that CPLEX uses for solving linear programming are PREIND (preprocessing pre-solve indicator) and PREDUAL (preprocessing dual). As described in [18] and [6], when parameter setting PREIND = 1 (ON), the preprocessing pre-solver is enabled and both the number of variables and the number of constraints is reduced before any type of algorithm is used. By setting PREIND = 0 (OFF) the pre-solver routine in CPLEX is disabled. PREDUAL is the second preprocessing parameter in CPLEX. By setting parameter PREDUAL = 0 (ON) or -1 (OFF), CPLEX automatically selects whether to solve the dual of the original LP or not. Both are used with the default settings for the CPLEX primal simplex method, the CPLEX dual simplex method, and the CPLEX barrier method. Neither CPLEX pre-solver nor PREDUAL parameters were used in any part of the developed dynamic active-set methods for NNLPs and LPs.

3.5 Computational Results for NNLP

Table 3.1 illustrates the performance comparison between dynamic RAD method and the previously defined constraint selection technique COST RAD on Set 1 to Set 4 for various dimensions of the matrix $\boldsymbol{A}$ used in [6]. Both methods are compared with the CPLEX barrier method (interior point), the CPLEX primal simplex method, and the CPLEX dual

simplex method. The worst performance occurs at m/n ratio of 200, where on average, dynamic RAD is 8% faster than COST RAD for densities less than 0.2 and 18% slower for densities above 0.2. When the density increases, dynamic RAD shows an increase in computation time more than that of COST RAD. On the other hand, for an m/n ratio of 20 the CPU times decrease with an increase in density. For higher densities above 0.01, dynamic RAD is more efficient and takes less computation times than COST RAD. On average, dynamic RAD is 10% more efficient than COST RAD. For an m/n ratio of 2 at densities higher than 0.009, the data show that COST RAD starts taking significantly more time than dynamic RAD. Dynamic RAD was 5.5% faster than COST RAD over all densities and 21% faster on average for densities above 0.5. For an m/n ratio of 1 with densities greater than 0.01, dynamic RAD is about 8% more efficient than COST RAD. On average, dynamic RAD is superior performance to COST RAD for problem sets 2, 3, and 4.

Table 3.2 from [6] is presented to provide an immediate comparison of the developed dynamic RAD method with the standard CPLEX solvers. A reporting limit of 3000 seconds was used. On average, the CPU times for dynamic RAD were faster than any of the CPLEX solvers across all densities and ratios. However, CPLEX barrier methods show smaller CPU times when ratio m/n = 20 and the density is less than or equal to 0.01.

3.6 Computational Results for LP

Table 3.3 shows computational results for the CPLEX primal simplex method, the dual simplex method, and the interior point barrier method for the general LP problem set used in [18]. CPU times for COST GRAD and VIOL using both the multi-cut technique and dynamic approaches are presented for comparison. Dynamic GRAD is stable over the range of densities. In addition, its performance is superior to multi-cut GRAD for every problem instance. Average CPU times for GRAD using multi-cut method and dynamic

approach are 43.87 and 24.57 seconds, respectively, a 42% improvement in computation time. Average computation times for GRAD and VIOL using dynamic approach are 24.57 seconds vs. 33.82 seconds, respectively.

It should be noted that GRAD captures more information than VIOL in higher densities to discriminate between constraints. Interestingly, when the dynamic active-set is used for both GRAD and VIOL, their CPU times are significantly faster than the same metrics with the multi-cut method. GRAD using the multi-cut technique takes the longest computation time in comparison to others at higher densities. Unlike the proposed dynamic approach, the LP algorithm COST GRAD requires checking the signs of the nonzero a_{ij} and therefore more computation time for higher densities.

The efficiency of VIOL decreases significantly with increasing density. On average, dynamic GRAD is approximately 35 times faster than the CPLEX primal simplex, 21 times faster than the CPLEX dual simplex, and 17 times faster the CPLEX barrier linear programming solvers without preprocessing. The superior overall performance of GRAD using dynamic approach is apparent across all densities in general LP set.

Table 3.1 Results from dynamic RAD and COST RAD for set 1- set 4, (Random, NNLP a_{ij} =1-5, b_i =1-10, c_j = 1-10)

		Dynamic RAD [+]				COST RAD [+]			
	n	1000	3163	10000	14143	1000	3163	10000	14143
	m	200000	63246	20000	14143	200000	63246	20000	14143
	m/n	200	20	2	1	200	20	2	1
Density	No	CPU TIME, sec [++]							
0.005	1	2.02	30.88	108.52	126.93	2.10	30.82	108.70	127.55
0.006	2	2.47	32.19	106.48	113.68	2.42	31.48	104.87	114.03
0.007	3	2.63	30.39	96.61	104.34	2.65	29.41	92.45	104.18
0.008	4	2.46	31.03	90.14	89.24	2.54	30.63	88.20	90.73
0.009	5	2.67	28.89	82.66	86.57	2.78	30.10	83.53	85.21
0.01	6	2.73	28.22	75.46	83.66	2.79	27.81	77.90	80.43
0.02	7	2.88	23.17	45.55	49.82	3.09	24.69	47.63	49.95
0.03	8	2.83	17.97	33.85	37.35	3.22	20.49	36.68	38.33
0.04	9	2.92	15.24	29.23	28.98	3.33	19.06	32.74	32.53
0.05	10	2.97	14.10	24.83	26.37	3.34	16.97	28.23	28.59
0.06	11	2.86	11.93	23.38	24.45	3.20	14.94	27.58	27.27
0.07	12	2.94	11.21	20.38	21.08	3.41	14.88	23.59	23.79
0.08	13	2.87	10.25	19.47	21.43	3.32	13.57	23.44	24.19
0.09	14	3.05	9.33	19.43	20.49	3.38	12.67	23.09	23.80
0.1	15	3.20	9.33	18.03	18.78	3.39	12.92	22.93	20.85
0.2	16	4.39	8.07	14.86	16.50	4.30	11.09	18.87	20.31
0.3	17	5.26	8.19	13.77	15.27	4.97	10.58	18.11	19.46
0.4	18	6.40	9.19	14.32	15.60	5.76	12.31	18.55	18.88
0.5	19	7.80	9.84	14.33	15.97	6.98	11.92	18.00	19.89
0.75	20	10.86	11.91	14.55	16.26	8.26	12.01	17.19	18.06
1	21	12.93	12.01	12.61	14.58	8.39	12.20	17.71	18.50
Average		4.24	17.30	41.83	45.11	3.98	19.07	44.28	46.98

[+]Used CPLEX preprocessing parameters of presolve = off and predual = off.
[++]Average of 5 instances of LPs at each density.

Table 4.4 Result obtained from primal, dual simplex and barrier for set1-set4, (Random, NNLP a_{ij} =1-5, b_i = 1-10, c_j = 1-

		Primal[--]				Dual[--]				Barrier [--]			
	n	1000	3163	10000	14143	1000	3163	10000	14143	1000	3163	10000	14143
	m	200000	63246	20000	14143	200000	63246	20000	14143	200000	63246	20000	14143
	m/n	200	20	2	1	200	20	2	1	200	20	2	1
Density	No	CPU Time (Sec) [++]											
0.005	1	7.01	71.02	228.51	309.83	54.84	762.62	1597.24	1169.04	2.36	14.52	240.17	650.83
0.006	2	10.36	77.28	245.60	291.07	60.29	803.97	1607.16	2413.42	2.39	16.30	224.08	666.54
0.007	3	12.98	75.84	219.72	265.09	91.39	876.85	1483.20	1702.47	3.04	18.34	233.55	671.56
0.008	4	15.72	82.01	206.45	239.30	100.06	912.75	1445.54	1236.76	3.90	20.70	232.38	668.82
0.009	5	19.25	80.35	196.72	216.23	114.95	898.99	1375.73	427.95	4.76	22.66	232.23	649.26
0.01	6	21.92	78.50	182.47	216.60	123.49	912.63	1252.05	436.31	5.53	24.29	228.76	650.30
0.02	7	39.90	78.80	118.28	127.59	203.08	963.66	807.29	362.34	17.13	32.08	242.54	711.26
0.03	8	45.42	79.75	98.02	108.60	217.18	1207.76	545.91	723.98	28.79	45.03	266.90	727.61
0.04	9	50.30	78.78	89.75	88.32	248.75	1489.40	450.08	539.92	41.50	62.28	292.15	806.80
0.05	10	55.16	78.92	81.09	82.14	256.49	1746.46	418.69	519.50	53.72	81.32	327.01	837.67
0.06	11	60.34	77.49	77.28	78.27	251.39	2124.31	378.71	409.47	67.58	100.48	359.53	897.58
0.07	12	62.07	78.93	70.44	70.37	251.74	2446.69	310.89	544.15	84.70	125.49	401.72	948.01
0.08	13	62.92	76.96	70.21	69.81	264.48	2799.62	307.25	388.94	99.51	149.37	454.01	1038.86
0.09	14	66.57	79.07	71.46	72.37	258.14	2523.03	718.04	427.95	119.26	186.06	495.28	1153.31
0.1	15	71.00	74.57	67.43	62.64	287.36	2251.10	267.14	436.31	138.67	207.54	539.64	1194.56
0.2	16	87.49	83.12	64.38	62.99	294.39	1450.82	201.73	362.34	379.68	691.77	1298.76	2529.97
0.3	17	94.57	77.91	67.14	66.61	341.44	1280.71	175.16	267.16	657.45	1333.29	2418.75	[b]
0.4	18	99.33	78.46	73.58	71.48	384.10	1236.30	146.09	233.39	985.86	2076.09	[b]	[b]
0.5	19	111.30	84.30	86.50	75.62	427.16	1173.49	133.49	208.65	1350.82	[b]	[b]	[b]
0.75	20	128.26	99.35	115.00	102.51	410.98	1056.18	132.25	181.95	[b]	[b]	[b]	[b]
1	21	207.55	94.09	393.54	145.96	375.89	411.19	148.90	165.45	[b]	[b]	[b]	[b]
Average		63.30	80.26	134.46	134.45	238.93	1396.60	662.03	626.55	n/a	n/a	n/a	n/a

[--]Used CPLEX preprocessing parameters of presolve = ON and predual = Auto.

[++]Average of 5 instances of LPs at each density.

[b] Runs with CPU times > 3000s are not reported.

Table 3.3 Comparison of computation times of CPLEX solvers, GRAD, and VIOL using both dynamic active-set and multi-cut method on general LP problem set (Random LP with 1000 variables and 200,000 constraints [18])

		Constraint selection metric[+]				CPLEX[--]			
No	Density	GRAD	VIOL	GRAD	VIOL	Primal	Dual	Barrier	Auto[1]
		Multi-cut method		Dynamic active-					
		CPU TIME, sec[++]							
1	0.005	9.85	12.31	7.96	9.26	40.99	23.05	2.39	1.79
2	0.006	11.48	14.50	9.44	11.11	84.56	35.52	2.62	3
3	0.007	13.36	14.21	10.60	12.58	128.65	48.62	3.79	4.07
4	0.008	14.24	14.67	12.09	13.00	183.70	61.56	4.93	5.71
5	0.009	15.41	15.32	12.25	14.57	212.79	75.34	6.06	7.19
6	0.01	16.55	17.09	14.10	15.33	256.70	92.11	7.33	8.18
7	0.02	24.74	22.24	20.93	21.79	396.55	205.25	15.86	22.21
8	0.03	27.84	24.30	22.91	26.21	460.01	295.18	26.63	37.3
9	0.04	30.55	24.47	23.87	29.52	602.73	350.86	35.26	51.03
10	0.05	37.59	28.72	28.52	33.57	617.29	396.10	46.76	65.6
11	0.06	34.29	26.58	26.86	33.80	656.22	438.92	59.55	81.7
12	0.07	37.46	28.05	26.91	34.34	729.43	465.61	71.65	104.35
13	0.08	36.28	26.29	25.54	33.46	739.21	510.10	82.98	130.13
14	0.09	37.97	27.74	24.60	33.21	823.11	521.89	94.01	140.32
15	0.10	39.50	28.30	25.99	35.61	956.17	554.29	108.03	178.63
16	0.20	56.26	36.64	27.97	41.28	1456.41	759.66	280.09	432.58
17	0.30	60.93	42.40	28.41	40.68	1664.83	900.12	527.05	904.53
18	0.40	74.58	56.97	33.39	52.19	2033.10	1057.27	760.07	1325.8
19	0.50	85.02	71.35	36.85	54.68	1925.32	1334.80	1076.40	2253.9
20	0.75	113.02	116.78	39.44	59.53	2232.88	1571.28	2132.53	NA
21	1.00	144.35	173.02	57.22	104.5	2301.76	1717.25	3267.10	NA
Avg		43.87	39.14	24.57	33.82	881.07	543.56	410.05	NA

[+] Used CPLEX preprocessing parameters of presolve = off and predual = off.
$\mathbf{1}^T \boldsymbol{x} \leq M = 10^{10}$ was used as the bounding constraint.
[++] Average of 5 instances of LPs at each density.
[--] Used CPLEX preprocessing parameters of presolve = ON and predual = Auto.
[1] Computation time for one problem instance.

For comparison purposes, Table 3.4 shows GRAD and VIOL computation times when a fixed number of violated constraints is added at each iteration. Adding a fixed number of constraints is examined for both GRAD and VIOL. At densities below 0.03,

dynamic GRAD takes less CPU time than the fixed-cut approach. GRAD with 500 cuts per iteration shows a faster solution time than 100 or 1000 cuts. VIOL performs best for a 100-constraint cut. On the other hand, GRAD performs best for a 500-constraint cut. In fact, the 500-constraint cut for GRAD performs as well as the GRAD dynamic active-set approach. However, determining an optimum number of cuts for a given problem is not possible.

Table 3.4 Comparison of computation times of GRAD using dynamic active-set and fixed cut method on general LP problem set (Random LP with 1000 variables and 200,000 constraints [18])

No	Density	Constraint selection metric⁺							
		GRAD	VIOL	GRAD			VIOL		
		Dynamic active-set		Fixed number of constraints					
				100	500	1000	100	500	1000
		CPU TIMES, sec⁺⁺							
1	0.005	7.96	9.26	14.58	10.04	8.05	10.49	8.82	11.56
2	0.006	9.44	11.11	18.75	13.14	9.48	12.61	10.57	14.45
3	0.007	10.60	12.58	20.32	13.24	10.67	13.87	12.22	15.01
4	0.008	12.09	13.00	23.57	12.63	12.05	14.97	12.82	16.22
5	0.009	12.25	14.57	23.16	13.60	12.32	15.88	14.00	18.35
6	0.01	14.10	15.33	26.04	14.83	13.59	17.72	15.35	19.26
7	0.02	20.93	21.79	36.49	21.27	20.38	23.55	22.70	28.35
8	0.03	22.91	26.21	38.40	22.33	22.30	25.40	26.07	34.22
9	0.04	23.87	29.52	38.48	22.68	23.19	25.63	27.51	36.21
10	0.05	28.52	33.57	46.34	27.77	28.69	29.67	32.25	41.66
11	0.06	26.86	33.80	40.35	24.47	26.26	27.12	30.36	40.53
12	0.07	26.91	34.34	41.91	26.05	27.92	28.88	32.69	42.09
13	0.08	25.54	33.46	37.80	24.61	26.36	26.64	31.62	42.58
14	0.09	24.60	33.21	37.71	25.01	28.18	27.38	32.19	43.69
15	0.1	25.99	35.61	39.30	25.54	28.00	29.12	33.81	46.08
16	0.2	27.97	41.28	41.66	29.36	33.48	33.54	40.20	57.02
17	0.3	28.41	40.68	38.05	28.25	34.05	32.88	41.53	59.39
18	0.4	33.39	52.19	41.45	33.58	41.14	39.98	50.68	74.25
19	0.5	36.85	54.68	42.40	36.86	46.14	44.68	56.76	81.71
20	0.75	39.44	59.53	45.88	40.36	50.28	52.67	69.07	101.71
21	1	57.22	104.58	48.44	46.14	57.55	61.59	78.23	114.15
Average		24.57	33.82	35.29	24.37	26.67	28.30	32.36	44.69

⁺Used CPLEX preprocessing parameters of presolve = off and predual = off. $\mathbf{1}^T\boldsymbol{x} \leq M = 10^{10}$ was used as the bounding constraint.

⁺⁺Average of 5 instances of LPs at each density.

3.7 Conclusion

In this chapter, dynamic active-set methods have been proposed for both NNLPs and LPs. In particular, these new approaches were compared to existing methods for problems with various sizes and densities. On average, dynamic RAD shows superior performance over COST RAD for the NNLP problem sets 2, 3, and 4. In the LP problem set, dynamic GRAD significantly outperformed the COST GRAD as well as the CPLEX primal simplex and the dual simplex. In this LP problem set, however, the barrier solver did outperform all methods for densities up to 0.03. In addition, dynamic GRAD outperformed a dynamic version of VIOL, which is a standard method in column generation and decomposition methods.

CHAPTER 4

Posterior Cost

4.1 Introduction

In this chapter, we propose an active-set method to solve nonnegative linear programming problems more efficiently. Our approach divides the constraints of problem P into operative and inoperative constraints at each iteration. Operative constraints are those active in the current relaxed sub-problem $P_r,\ r = 1,2,\dots,\text{of } P$, while the inoperative ones are constraints of the problem P not active in P_r. In our active-set method we iteratively solve $P_r,\ r = 1,2,\dots,\text{of } P$ after adding one or more violated inoperative constraints from (1.2) to P_{r-1} until the solution $\boldsymbol{x}_r^*$ to P_r is a solution to P.

RAD is a geometric constraint selection criterion for determining the constraints most likely to be binding at optimality. In the associated active-set COST algorithm of [19], all constraints of (1.2) are pre-ordered by decreasing value of RAD prior to solving an initial bounded problem P_0 by the primal simplex. The dual simplex is then used when violated inoperative constraints are added according to their RAD value. In computational experiments, RAD proved superior to existing linear programming methods for NNLPs. A similar constraint selection metric GRAD was developed in [18] to solve general linear programs (LPs). It should be noted that both a constraint selection metric and the associated COST active-set method are identified by the same name – for example, RAD.

4.2 Overview

In this chapter a posterior constraint selection metric NVRAD, as distinguished from the prior constraint metric RAD, is developed for NNLPs and utilized in a dynamic active-set framework. The main contributions are:

(i) a geometric interpretation is presented for NVRAD,

(ii) a dynamic active-set approach is developed by adding a varying number of violated constraints at P_r based on the progress at P_{r-1},

(iii) extensive computational experiments that (a) confirm the efficacy of the dynamic active-set approach for the posterior method as compared to multi-cut and multi-bound technique proposed in Saito et al. [6]; (b) indicate the ability of NVRAD to determine constraints likely to be binding at optimality for NNLPs; and (c) show that NVRAD solves NNLPs faster than other computational methods, including RAD and various versions of VIOL.

4.3 Explanation of NVRAD

Let $\boldsymbol{x}_r^*$ be the current optimal solution for some P_r with a perpendicular distance $d = \frac{\boldsymbol{a_i}^T \boldsymbol{x}_r^* - b_i}{\| \boldsymbol{a}_i \|}$ to a violated hyperplane $\boldsymbol{a}_i{}^T \boldsymbol{x} = b_i$. It follows that

$$\frac{d}{\frac{b_i}{\| \boldsymbol{a}_i \|}} = \frac{\boldsymbol{a}_i{}^T \boldsymbol{x}_r^* - b_i}{b_i}. \tag{4.1}$$

Note that $\frac{b_i}{\| \boldsymbol{a}_i \|}$ is the perpendicular distance of hyperplane $\boldsymbol{a}_i{}^T \boldsymbol{x} = b_i$ to the origin. Thus choosing a violated hyperplane $\boldsymbol{a}_i{}^T \boldsymbol{x} = b_i$ with a maximum value $\frac{\boldsymbol{a_i}^T \boldsymbol{x}_r^* - b_i}{b_i}$ on the right side of (6) can be interpreted from the left side of (6) as selecting a violated constraint giving the deepest cut based on information derived from $\boldsymbol{x}_r^*$. But the information from the prior constraint selection metric $RAD(\boldsymbol{a}_i, b_i, \boldsymbol{c})$ of (4) is also valuable. From [20], the expression $\frac{\boldsymbol{a_i}^T \boldsymbol{c}}{b_i}$ on the right side of (4) is the distance from the origin to the hyperplane $\boldsymbol{a}_i^T \boldsymbol{x} \leq b_i$ along the vector $\boldsymbol{c}$, i.e., the direction of steepest ascent for the objective function of (P). We update the prior information obtained from RAD with subsequent information obtained from $\boldsymbol{x}_r^*$. In a manner reminiscent of using Bayes' Theorem to get a posterior probability

distribution from a prior distribution and a random sample [28], we multiply RAD by the right side of (4.1) to obtain

$$NVRAD(\boldsymbol{a}_i, b_i, \boldsymbol{c}, \boldsymbol{x}_r^*) = \frac{\boldsymbol{a}_i{}^T \boldsymbol{c}}{b_i{}^2} (\boldsymbol{a}_i{}^T \boldsymbol{x}_r^* - b_i). \tag{4.2}$$

Equation (4.2) incorporates the global prior information from RAD with local information from $\boldsymbol{x}_r^*$. NVRAD thus seeks

$$i^* \in \underset{i \notin OPERATIVE}{argmax} \left(\frac{\boldsymbol{a}_i{}^T \boldsymbol{c}}{b_i{}^2} (\boldsymbol{a}_i{}^T \boldsymbol{x}_r^* - b_i) : \boldsymbol{a}_i{}^T \boldsymbol{x}_r^* > b_i \right) \tag{4.3}$$

for each P_r. We mention that the $b_i{}^2$ term in the denominator of (4.2) works better in the constraint selection metric NVRAD than would simply b_i. This fact was established in computational results not reported here and therefore supports derivation.

4.4 Dynamic COST NVRAD

Let $\boldsymbol{x}_r^*$ be the optimal extreme point for P_r. The cosine of the angle θ_r (in radians) between $\boldsymbol{x}_r^*$ and $\boldsymbol{c}$ is given by

$$cos(\theta_r) = \frac{\boldsymbol{c}^T \boldsymbol{x}_r^*}{\|\boldsymbol{x}_r^*\| \, \|\boldsymbol{c}\|}, \tag{4.4}$$

which is nonnegative since P_r is also an NNLP. Ideally we want to decrease θ_r at each iteration so that $\boldsymbol{x}_r^*$ is more in line with the gradient of the objective function z of (1.1) as illustrated in Figure 4.1 for two dimensions. Toward that end, we develop a dynamic heuristic that adds a varying number of violated inoperative constraints to P_r according to the progress made at P_{r-1}in reducing the angle between vectors $\boldsymbol{x}_{r-1}^*$ and $\boldsymbol{c}$**.** As our goal, let $\theta_r = 0$ in (4.4) to give

$$\boldsymbol{c}^T \boldsymbol{x}_r^* = \|\boldsymbol{x}_r^*\| \, \|\boldsymbol{c}\|. \tag{4.5}$$

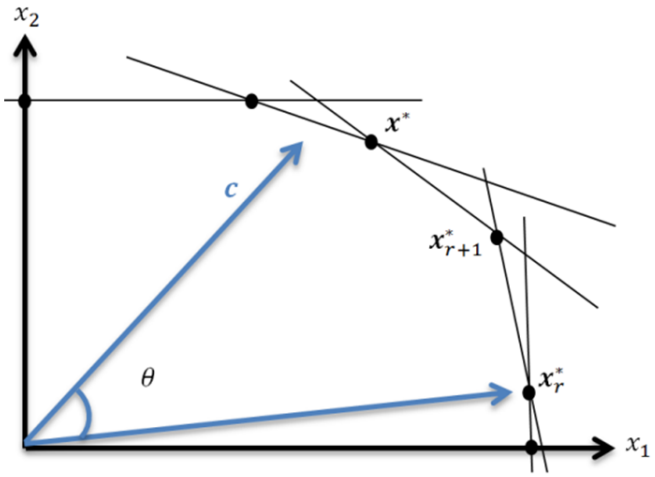

Figure 4.1 An ideal changing angle between $\boldsymbol{x}_r^*$ and $\boldsymbol{c}$

When $\theta_r = 0$, it thus follows from (4.5) that

$$\frac{\sum_{j=1}^{n} c_j x_{rj}^*}{\sqrt{\sum_{j=1}^{n} c_j{}^2}} = \sqrt{\sum_{j=1}^{n} (x_{rj}^*)^2}. \tag{4.6}$$

Letting $|\,.\,|$ denote absolute value, define

$$\delta_r(\boldsymbol{x}_r^*) = \left| \frac{\sum_{j=1}^{n} c_j x_{rj}^*}{\sqrt{\sum_{j=1}^{n} c_j{}^2}} - \sqrt{\sum_{j=1}^{n} (x_{rj}^*)^2} \right| \tag{4.7}$$

as a measure of the performance of our active-set method at iteration r. The value of $\delta_r(\boldsymbol{x}_r^*)$ decreases as $\boldsymbol{x}_r^*$ as θ_r decreases. Such a decrease usually occurs as $\boldsymbol{x}_r^*$ moves toward an optimal extreme point of P.

The dynamic COST NVRAD for solving NNLPs is described as follows. Constraints are initially ordered by the RAD constraint selection metric (2.2). To construct P_0, we choose constraints from (1.2) in descending order of RAD until each variables x_j has an $a_{ij} > 0$ in the coefficient matrix of P_0. We say the variables are covered by the constraints of the initial problem P_0. No equality constraints are considered here, but any equality constraints can be included in P_0. P_0 is then solved by the primal simplex to achieve an

initial solution $\boldsymbol{x}_0^*$ and $\delta_0(\boldsymbol{x}_0^*)$ is calculated accordingly. Let γ_r be the number of constraints of problem P violated by $\boldsymbol{x}_r^*$. At every iteration $r-1$ and r, the values of $\delta_{r-1}(\boldsymbol{x}_{r-1}^*)$ and $\delta_r(\boldsymbol{x}_r^*)$ are calculated, respectively, and the percentage of improvement is calculated by

$$\omega_r = max\left\{0, \left(\frac{\delta_{r-1}(\boldsymbol{x}_{r-1}^*) - \delta_r(\boldsymbol{x}_r^*)}{\delta_{r-1}(\boldsymbol{x}_{r-1}^*)}\right)\right\} * 100\ , \forall\ r = 1,2, \ldots, \tag{4.8}$$

where $\omega_r > 0$ represents the percent of improvement made in reducing the angle between vectors $\boldsymbol{x}_r^*$ and $\boldsymbol{c}$ at iteration r. With **[.]** denoting the greatest integer function, let

$$\begin{cases} \varphi_{r+1} = \varphi_r * (1 + [(\ln \omega_r)^{-1}]), \forall\ r = 1,2, \ldots & \text{if} \quad \omega_r > 1 \\ \varphi_{r+1} = \gamma_r, \quad \forall\ r = 1,2, \ldots & \text{if} \quad \omega_r \leq 1 \end{cases} \tag{4.9}$$

where $\varphi_1 = 200$. The value of φ_r is an upper bound on the possible number of non-operative violated constraints that can be added at active-set iteration r. The actual number added is $min\{\varphi_{r+1}, \gamma_r\}$. The active-set function is defined to compensate for the lack of progress in P_{r-1} by adding more violated constraints at P_r. The algorithm stops when $\gamma_r = 0$.

The pseudocode for dynamic NVRAD is as follows.

Step 1 — Identify constraints to initially bound the problem.

1: $a^* \leftarrow 0$

2: $\boldsymbol{while}\ a^* \not> 0\ \boldsymbol{do}$

3: $Let\ i^* \in \underset{i \notin BOUNDING}{argmax}\ RAD(\boldsymbol{a}_i, b_i, \boldsymbol{c})$

4: $\boldsymbol{if}\ \ \exists j\ |a_j^* = 0\ \boldsymbol{and}\ a_{i*j} > 0\ \boldsymbol{then}$

5: $BOUNDING \leftarrow BOUNDING \cup \{i^*\}$

6: $\boldsymbol{end\ if}$

7: $\boldsymbol{a}^* \leftarrow \boldsymbol{a}^* + \boldsymbol{a}_{i*}$

8: $Optimized \leftarrow \boldsymbol{false}$

9: $\boldsymbol{end\ while}$

Step 2 — Using the primal simplex method, obtain an optimal solution x_0^* for the initial bounded problem P_0

$$\begin{gathered} maximize\ z\ =\ \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to}\ \boldsymbol{a}_i^T\boldsymbol{x} \leq\ b_i \quad \forall i\ \in\ BOUNDING \\ \boldsymbol{x}\ \geq\ \boldsymbol{0} \end{gathered}$$

Step 3 — Perform the following iterations until an optimal solution to problem P is found.

1: $r\ \leftarrow 0$

2: $\boldsymbol{While}\ (Optimized = \textbf{false})\ \boldsymbol{do}$

3: Calculate $\delta_r(\boldsymbol{x}_r^*)$

4: $\boldsymbol{if}\ r > 1\ \ \boldsymbol{then}\ \omega_r = max\left\{0, \left(\frac{\delta_{r-1}(\boldsymbol{x}_{r-1}^*) - \delta_r(\boldsymbol{x}_r^*)}{\delta_{r-1}(\boldsymbol{x}_{r-1}^*)}\right)\right\}$

5: $\boldsymbol{if}\ \omega_r > 1\ \boldsymbol{then}\ \varphi_{r+1} =\ \varphi_r * (1\ + [(\ln \omega_r)^{-1}])$

6: $\boldsymbol{else\ if}\ \omega_r \leq 1\ \boldsymbol{then}\ \varphi_{r+1} = \gamma_r$

7: $\boldsymbol{end\ if}$

8: $\boldsymbol{else}\ \varphi_r \leftarrow 200$

9: $\boldsymbol{end\ if}$

10: $\boldsymbol{if}\ \{\boldsymbol{a}_i^T\boldsymbol{x}_r^* > b_i\ ,\ \forall\ i = 0, \dots, rows\ \}\ \boldsymbol{then}$

11: $\gamma_r \leftarrow \#\{\boldsymbol{a}_i^T\boldsymbol{x}_r^* > b_i\ ,\ \forall\ i = 0, \dots, rows\ \}$

12: $Let\ i^* \in \underset{i \notin OPERATIVE}{argmax}\ \{NVRAD(\boldsymbol{a}_i, b_i, \boldsymbol{c}, \boldsymbol{x}_r^*) = \frac{\boldsymbol{a}_i{}^T\boldsymbol{c}}{b_i{}^2}(\boldsymbol{a}_i{}^T\boldsymbol{x}_r^* - b_i) : \boldsymbol{a}_i{}^T\boldsymbol{x}_r^* > b_i\ \}$

13: $\boldsymbol{for}\ (\text{i} = 0\ \text{to}\ min\{\varphi_{r+1}, \gamma_r\})\ \ OPERATIVE \leftarrow OPERATIVE\ \cup \{i^*\}\ \boldsymbol{end}$

14: Solve the following P_r by the dual simplex method to obtain $\boldsymbol{x}_r^*$

15: $r\ \leftarrow\ r\ + 1$

16: $\boldsymbol{Go\ to\ 3}$

17: $\boldsymbol{else}\ \ Optimized\ \leftarrow \boldsymbol{true}$ // $\boldsymbol{x}_r^*$ is an optimal solution to P.

18: ***end if***

19: ***end while***

4.5 Computational Experiments

Dynamic NVRAD is now compared with the CPLEX primal simplex, dual simplex, and barrier methods. NVRAD is further compared to RAD, VIOL, and a normalized version of VIOL called NVIOL, which is usually superior to VIOL. Since, it's preferable to apply a combined procedure to measure the effect of the both global and local information, the COSTs RAD and NVRAD are used at even and odd iterations, respectively which is called Hybrid here.

4.6 Problem Instances

Five sets of NNLPs in [6] are used to evaluate the performance of the dynamic COST NVRAD. Each set contains 105 randomly generated NNLPs at 21 different density levels ranging from 0.005 to 1, and four ratios of $(m \text{ constraints})/(n \text{ variables})$ ranging from 200 to 1. There are five problem instances per combination of density level and ratio. In these problem sets, randomly generated real numbers between 1 and 5, 1 and 10, 1 and 10 were assigned to the elements of $\boldsymbol{A}, \boldsymbol{b},$ and $\boldsymbol{c},$ respectively. To avoid having a constraint in the form of an upper bound on a variable, each constraint is required to have at least two non-zero a_{ij}. Problem Set 5 of NNLPs is a set of large-scale problems with 5000 variables and 1,000,000 constraints. In this set, real numbers between 1 and 100 were assigned to the elements of $\boldsymbol{b}$ and $\boldsymbol{c}$ with densities p ranging from 0.0004 to 0.06. Again, to avoid having a constraint in the form of a upper bound on a variable, each constraint is required to have at least two non-zero a_{ij}.

4.7 CPLEX Processing

Two CPLEX parameters for solving linear programming are discussed here. Preprocessing pre-solve indicator (PREIND) and preprocessing dual (PREDUAL) are the two parameters that CPLEX uses for solving linear programming. Preprocessing pre-solver is enabled with the parameter setting PREIND = 1 (ON), which reduces both the number of variables and the constraints before any type of algorithm is used. Pre-solver routine in CPLEX is disabled by setting PREIND = 0 (OFF). The second preprocessing parameter in CPLEX, which affects the computational speed, is PREDUAL. By setting parameter PREDUAL = 0 (ON) or PREDUAL = -1 (OFF), CPLEX automatically selects whether to solve the dual of the original LP or not, respectively. Both CPLEX pre-solver and preprocessing dual were disabled in the developed methods.

4.8 NVRAD Computation Results

The experiments are performed on an Intel Core (TM) 2 Duo X9650 3.00GHz with a Linux 64-bit operating system and 8 GB of RAM. The developed method uses IBM CPLEX 12.5 callable library to solve nonnegative linear programming problems. The CPU times represent the average computation time of five problem samples in each density level. In Table 4.1, the performance of the non-dynamic COST active-set approach of [19] was compared to the dynamic COST active-set approach for Set 1. Dynamic NVRAD significantly reduces computational time compared to the multi-cuts and multi-bounds technique of [6]. Moreover, in higher density problems the dynamic version of NVIOL is up to 21 times faster than when the same metric incorporated the multi-cuts and multi-bounds technique. Dynamic NVRAD performs better than VIOL and NVIOL on every problem instance.

Table 4.1 Comparison of CPU times to illustrate the effect of multi-cuts and multi-bounds and dynamic active-set approach on problem Set1 (Random NNLP with 1000 variables and 200,000 constraints, a_{ij}=1-5, b_i =1-10, c_j =1-10)

		VIOL$^+$	NVIOL$^+$	NVRAD$^+$	NVIOL$^+$	NVRAD$^+$
		Multi-Cut & Multi-Bound			Dynamic Active-Set	
Density	No	CPU Time (sec)$^{++}$				
0.005	1	6.54	4.56	2.51	2.49	2.26
0.006	2	6.84	5.06	2.92	2.96	2.62
0.007	3	7.15	5.34	3.03	3.03	2.75
0.008	4	6.61	4.96	3.02	3.13	2.83
0.009	5	7.02	5.16	3.11	3.39	3.09
0.01	6	6.83	5.14	3.41	3.51	3.12
0.02	7	6.11	4.81	3.36	3.79	3.44
0.03	8	5.79	4.79	3.33	3.99	3.52
0.04	9	5.71	4.45	3.31	3.99	3.71
0.05	10	5.41	4.62	3.49	4.10	3.64
0.06	11	5.32	4.3	3.52	3.91	3.63
0.07	12	5.87	4.73	3.79	3.93	3.73
0.08	13	5.53	4.68	3.68	3.86	3.61
0.09	14	5.76	4.89	3.99	4.06	3.65
0.1	15	6.04	5.07	4.31	4.08	3.89
0.2	16	10.9	9.64	8.28	4.96	4.82
0.3	17	17.3	15.15	13.05	6.03	5.68
0.4	18	24.64	22.12	20.53	7.28	6.56
0.5	19	32.93	29.85	27.67	7.63	7.34
0.75	20	62.21	57.36	54.97	10.53	10.50
1	21	261.23	251.65	245.5	11.43	11.13
Average		23.89	21.82	20.04	4.86	4.55

$^{++}$Average of 5 instances of LP at each density
$^+$ Used CPLEX preprocessing parameters of presolve = off and predual = off.

In Table 4.2, CPU times of the test problems by dynamic NVRAD are compared with RAD. In problem Set 1, RAD is 12.5% faster than NVRAD over all densities. In problem Set 2, the average computation times for RAD and dynamic NVRAD are 19.07 and 16.86 seconds respectively. On average, dynamic NVRAD is superior to RAD with 38.91 seconds and 41.87 seconds on Sets 3 and 4 respectively. At lower densities and higher dimensions, where there may not be much prior global information the results from Table 2 affirm the ability of NVRAD to add appropriate constraints at each iteration. For m/n ratio of 20,

NVRAD with 16.86 seconds is faster than the 19.07 seconds of COST RAD. For m/n ratio of 2, for densities less than 0.01, dynamic NVRAD is up to 18% more efficient than COST RAD. NVRAD performed well for both very low and very high density problems.

Table 4.2 CPU Times from RAD (multi-cuts and multi-bounds), and NVRAD using dynamic active-set method for set1-set 4 (Random, NNLP a_{ij} =1-5, b_i =1-10, c_j = 1-

		NVRAD$^+$				RAD$^+$			
	n	1000	3163	1000	1414	1000	3163	1000	1414
	m	20000	63246	2000	1414	20000	6324	2000	1414
	m/	200	20	2	1	200	20	2	1
		Dynamic Active-Set				Multi-Cut & Multi Bound			
Density	No	CPU Time (sec)$^{++}$				CPU Time (sec)$^{++}$			
0.005	1	2.26	25.00	88.36	106.2	2.10	30.82	108.7	127.5
0.006	2	2.62	27.23	88.73	97.31	2.42	31.48	104.8	114.0
0.007	3	2.75	25.79	82.04	90.65	2.65	29.41	92.45	104.1
0.008	4	2.83	27.49	78.04	78.92	2.54	30.63	88.20	90.73
0.009	5	3.09	27.43	74.65	75.18	2.78	30.10	83.53	85.21
0.01	6	3.12	26.07	68.23	73.29	2.79	27.81	77.90	80.43
0.02	7	3.44	22.48	45.06	46.24	3.09	24.69	47.63	49.95
0.03	8	3.52	18.59	34.78	38.25	3.22	20.49	36.68	38.33
0.04	9	3.71	16.92	29.96	30.75	3.33	19.06	32.74	32.53
0.05	10	3.64	15.47	26.31	28.01	3.34	16.97	28.23	28.59
0.06	11	3.63	13.62	24.62	25.62	3.20	14.94	27.58	27.27
0.07	12	3.73	12.93	22.24	23.19	3.41	14.88	23.59	23.79
0.08	13	3.61	11.99	20.74	22.30	3.32	13.57	23.44	24.19
0.09	14	3.65	11.39	20.47	21.64	3.38	12.67	23.09	23.80
0.1	15	3.89	10.81	19.65	20.18	3.39	12.92	22.93	20.85
0.2	16	4.82	8.98	16.31	19.44	4.30	11.09	18.87	20.31
0.3	17	5.68	8.84	15.66	19.09	4.97	10.58	18.11	19.46
0.4	18	6.56	9.77	15.76	17.23	5.76	12.31	18.55	18.88
0.5	19	7.34	10.60	15.82	17.89	6.98	11.92	18.00	19.89
0.75	20	10.50	11.24	15.80	16.73	8.26	12.01	17.19	18.06
1	21	11.13	11.34	13.85	11.12	8.39	12.20	17.71	18.50
Averag		4.55	16.86	38.91	41.87	3.98	19.07	44.28	46.98

$^{++}$ Average of 5 instances of LP at each density.
$^{+}$ Used CPLEX preprocessing parameters of presolve = off and predual = off.

In Table 4.3, CPU times of the test problems by RAD and hybrid method of COSTs (RAD and NVRAD) are compared. The hybrid version of COSTs represents a superior

performance than RAD on problem Sets 2, 3, and 4 in dynamic active-set frame work. It confirms the power of using both global and local information. In problem Set 1, dynamic RAD is 1.4% faster than hybrid method over all densities. In problem Set 2, the average computation times for dynamic RAD and hybrid are 16.84 and 16.03 seconds respectively. On average, hybrid method is superior to RAD in a dynamic framework with 38.40 seconds and 41.76 seconds on Sets 3 and 4 respectively.

In both Tables 4.2 and Table 4.3, HYBR is better than RAD, dynamic RAD, and dynamic NVRAD, though only marginally was better that dynamic NVRAD. HYBR can probably be improved. However, it is not our goal to determine the optimal ratio of RAD and NVRAD in HYBR since this ratio might differ depending on various factors such as density and m/n.

Table 4.3 Result obtained from dynamic RAD and hybrid method for Set1-Set4

		RAD+				Hybrid+ (RAD,NVRAD)			
	n	1000	3163	10000	14143	1000	3163	10000	14143
	m	200000	63246	20000	14143	200000	63246	20000	14143
	m/n	200	20	2	1	200	20	2	1
		Dynamic Active-Set				Dynamic Active-Set			
Density	No	CPU Time (sec)++							
0.005	1	2.02	29.51	106.42	127.70	2.19	27.14	94.26	113.63
0.006	2	2.32	30.20	107.61	116.08	2.53	28.51	95.25	103.78
0.007	3	2.49	29.07	95.87	104.79	2.78	26.14	84.13	95.47
0.008	4	2.47	29.80	89.37	93.12	2.76	27.63	78.88	83.75
0.009	5	2.67	28.59	80.46	86.46	2.93	27.21	77.09	81.11
0.01	6	2.65	27.09	75.60	81.28	3.02	25.68	70.59	75.25
0.02	7	2.85	22.01	45.39	48.01	3.28	20.90	44.45	46.91
0.03	8	2.83	17.30	33.40	36.20	3.19	17.46	34.44	36.18
0.04	9	2.82	14.97	29.29	27.47	3.18	15.08	27.58	29.31
0.05	10	2.97	13.91	24.38	24.70	3.04	13.68	24.61	24.94
0.06	11	2.85	11.43	22.31	22.82	3.19	11.81	23.03	23.32
0.07	12	2.93	11.04	19.40	20.48	3.31	11.55	19.99	20.88
0.08	13	2.91	10.37	18.75	19.87	3.30	10.36	19.19	20.72
0.09	14	3.16	9.16	18.11	18.93	3.34	9.37	18.52	19.54
0.1	15	3.06	9.54	17.35	17.34	3.51	9.18	17.74	17.89
0.2	16	4.34	8.12	14.39	15.99	4.36	7.92	14.01	15.55
0.3	17	5.70	8.57	13.31	15.32	5.40	7.86	12.92	14.66
0.4	18	7.00	9.28	13.55	14.58	6.50	8.95	12.65	13.93
0.5	19	7.95	9.85	13.60	16.35	7.84	9.60	13.38	14.52
0.75	20	10.64	12.05	14.43	15.56	9.81	10.77	12.38	14.13
1	21	12.66	11.71	12.60	14.72	11.00	9.76	11.39	11.59
Average		4.25	16.84	41.22	44.66	4.31	16.03	38.40	41.76

++ Average of 5 instances of LP at each density.

+ Used CPLEX preprocessing parameters of presolve = off and predual = off.

Table 4.4 from [6] is presented to provide an immediate comparison of the developed dynamic NVRAD with the standard CPLEX solvers. Dynamic NVRAD was superior across all ratios m/n and all densities except few problem instances with densities less than 0.02 and a ratio $\mathrm{m}/\mathrm{n} = 20$ in which barrier method was faster.

For a larger test problems (n = 5000; m = 1,000,000), Table 4.5 illustrates the effects of the dynamic active-set method along with the multi-cuts and multi-bounds technique over several constraint selection metrics. In a multi-cut and multi-bound framework, NVIOL shows 43% improvement in computation time compared to VIOL, while NVRAD reduces the solution time around 53% using the same technique. Among all posterior methods, NVRAD represents a superior performance compared to other constraint selection metrics using both dynamic and multi-cut and multi-bound approach. The average improvement made by using NVRAD in a dynamic framework is about 47% reduction in CPU time compared to NVRAD using multi-cut and multi-bound method. The average CPU times are not calculated for some of the CPLEX solvers since the CPU times more than 3000 seconds are not reported.

Table 4.4 Result obtained from primal, dual simplex and barrier for set1-set4, (Random, NNLP a_{ij} =1-5, b_i = 1-10, c_j = 1-10) [6]

		Primal[--]				Dual[--]				Barrier [--]			
	n	1000	3163	10000	14143	1000	3163	10000	14143	1000	3163	10000	14143
	m	200000	63246	20000	14143	200000	63246	20000	14143	200000	63246	20000	14143
	m/n	200	20	2	1	200	20	2	1	200	20	2	1
Density	No	CPU Time (Sec) [++]											
0.005	1	7.01	71.02	228.51	309.83	54.84	762.62	1597.24	1169.04	2.36	14.52	240.17	650.83
0.006	2	10.36	77.28	245.60	291.07	60.29	803.97	1607.16	2413.42	2.39	16.30	224.08	666.54
0.007	3	12.98	75.84	219.72	265.09	91.39	876.85	1483.20	1702.47	3.04	18.34	233.55	671.56
0.008	4	15.72	82.01	206.45	239.30	100.06	912.75	1445.54	1236.76	3.90	20.70	232.38	668.82
0.009	5	19.25	80.35	196.72	216.23	114.95	898.99	1375.73	427.95	4.76	22.66	232.23	649.26
0.01	6	21.92	78.50	182.47	216.60	123.49	912.63	1252.05	436.31	5.53	24.29	228.76	650.30
0.02	7	39.90	78.80	118.28	127.59	203.08	963.66	807.29	362.34	17.13	32.08	242.54	711.26
0.03	8	45.42	79.75	98.02	108.60	217.18	1207.76	545.91	723.98	28.79	45.03	266.90	727.61
0.04	9	50.30	78.78	89.75	88.32	248.75	1489.40	450.08	539.92	41.50	62.28	292.15	806.80
0.05	10	55.16	78.92	81.09	82.14	256.49	1746.46	418.69	519.50	53.72	81.32	327.01	837.67
0.06	11	60.34	77.49	77.28	78.27	251.39	2124.31	378.71	409.47	67.58	100.48	359.53	897.58
0.07	12	62.07	78.93	70.44	70.37	251.74	2446.69	310.89	544.15	84.70	125.49	401.72	948.01
0.08	13	62.92	76.96	70.21	69.81	264.48	2799.62	307.25	388.94	99.51	149.37	454.01	1038.86
0.09	14	66.57	79.07	71.46	72.37	258.14	2523.03	718.04	427.95	119.26	186.06	495.28	1153.31
0.1	15	71.00	74.57	67.43	62.64	287.36	2251.10	267.14	436.31	138.67	207.54	539.64	1194.56
0.2	16	87.49	83.12	64.38	62.99	294.39	1450.82	201.73	362.34	379.68	691.77	1298.76	2529.97
0.3	17	94.57	77.91	67.14	66.61	341.44	1280.71	175.16	267.16	657.45	1333.29	2418.75	[b]
0.4	18	99.33	78.46	73.58	71.48	384.10	1236.30	146.09	233.39	985.86	2076.09	[b]	[b]
0.5	19	111.30	84.30	86.50	75.62	427.16	1173.49	133.49	208.65	1350.82	[b]	[b]	[b]
0.75	20	128.26	99.35	115.00	102.51	410.98	1056.18	132.25	181.95	[b]	[b]	[b]	[b]
1	21	207.55	94.09	393.54	145.96	375.89	411.19	148.90	165.45	[b]	[b]	[b]	[b]
Average		63.30	80.26	134.46	134.45	238.93	1396.60	662.03	626.55	n/a	n/a	n/a	n/a

[--]Used CPLEX preprocessing parameters of presolve = ON and predual = Auto. [++]Average of 5 instances of LPs at each density.
[b] Runs with CPU times > 3000s are not reported.

Table 4.5 The comparison of computation times of dynamic active-set method and bounding technique (Random NNLP with 5,000 variables and 1,000,000 constraints, a_{ij} = 1-5, b_i = 1-100, c_j = 1-100)

No	Density	NVRAD[+]	RAD[+]	VIOL[+]	NVIOL[+]	NVRAD[+]	CPLEX Primal[--]	CPLEX Dual[--]	CPLEX Barrier [--]
		Dynamic Active-Set	Multi-Cuts & Multi-Bounds CPU Time (Sec)[++]						
1	0.0004	6.21	7.54	157.92	73.09	72.19	11.90	14.08	12.31
2	0.0005	9.22	12.26	177.96	100.86	106.31	23.41	29.83	16.61
3	0.0006	11.94	16.51	252.74	75.41	76.12	13.45	107.61	20.45
4	0.0007	15.45	22.19	282.95	92.70	93.64	18.99	176.50	24.60
5	0.0008	20.16	27.66	325.51	108.42	95.22	28.88	257.06	27.43
6	0.0009	23.70	33.24	346.76	120.57	91.06	40.17	339.49	29.80
7	0.0010	28.01	39.81	374.06	141.35	107.34	50.91	427.60	31.73
8	0.0020	70.01	89.57	393.48	222.63	174.78	173.03	1775.03	48.79
9	0.0030	90.09	104.83	368.92	245.17	190.37	244.01	[b]	61.31
10	0.0040	99.32	113.40	346.56	224.35	183.58	316.53	[b]	85.60
11	0.0050	103.78	113.17	322.98	215.49	172.42	366.80	[b]	91.11
12	0.0060	112.15	122.85	320.97	217.81	171.40	443.43	[b]	112.46
13	0.0070	106.61	116.00	283.16	214.48	160.63	474.40	[b]	136.03
14	0.0080	100.14	113.05	258.56	184.76	148.74	529.44	[b]	158.54
15	0.0090	94.43	104.68	229.32	165.47	138.51	566.20	[b]	198.31
16	0.0100	100.91	112.82	233.08	171.28	137.64	629.59	[b]	239.87
17	0.0200	76.77	83.83	142.85	106.45	90.60	1134.77	[b]	899.87
18	0.0300	69.41	76.69	114.25	86.83	76.77	1740.28	[b]	[b]
19	0.0400	65.87	67.36	103.22	79.26	71.60	1865.70	[b]	[b]
20	0.0500	63.71	64.58	100.60	80.35	71.87	2159.55	[b]	[b]
21	0.0600	64.57	65.62	102.05	82.42	74.41	[b]	[b]	[b]
Average		63.45	71.79	249.42	143.29	119.29	n/a	n/a	n/a

[++]Average of 5 instances of LP at each density. [b] Runs with CPU times > 2400s are not reported.
[--]Used CPLEX preprocessing parameters of presolve = ON and predual = Auto.
[+] Used CPLEX preprocessing parameters of presolve = off and predual = off.

4.9 Conclusions

An efficient posterior constraint selection metric NVRAD was developed here for NNLPs to utilize both prior global information and posterior local information, and a geometric interpretation was presented. In addition, a dynamic active-set approach was developed here to add varying numbers of constraints to for each P_r. The performance of dynamic NVRAD was checked on sets of large-scale NNLPs. Dynamic NVRAD outperformed the previously developed methods COST RAD and VIOL, as well as the CPLEX primal simplex, dual simplex, and barrier solvers.

CHAPTER 5

Application to Column Generation & Entering Variable Rule

5.1 Entering Variable Rule

Dantzig entering variable rule has been a dominant entering rule since the emergence of the simplex method. In the meantime, other entering rules are examined here to check their performance compared to the Dantzig entering rule. In the dual space of problem P, the likelihood of binding constraint at optimality can be determined by the corresponding dual version of RAD, called DRAD here.

$$DRAD(\boldsymbol{a}, \boldsymbol{b}, c) = j^* \in argmin\left\{ \frac{\boldsymbol{a}^{j^T}\boldsymbol{b}}{c_j} \mid \boldsymbol{a}^{j^T}\boldsymbol{y}_r^* < c_j \right\}, \tag{5.1}$$

where $\boldsymbol{a}^j$ is the j^{th} column of $\boldsymbol{A}$.

The performance of the DRAD entering rule is compared with the cosine simplex method discussed in Section (2.7.1) and the Dantzig entering rule. It tested over few, randomly generated problems with varying number of m and n with density of 1. In the DRAD entering rule, at each iteration, among all non-basic variables with negative reduced cost, the one with the lowest possible DRAD value is chosen to enter to the basis. In the next step, the minimum test determines the leaving variable. In the cosine simplex algorithm, all candidate variables with negative reduce cost, are required to calculate the corresponding cosine value, and the variable with the minimal cosine (2.8) is chosen as an entering variable.

To check the performance of the variable entering rules, two types of NNLPs, $a_{ij}, b_i, c_j \in \mathcal{R}^+$ and $a_{ij}, b_i, c_j \in \mathbb{Z}^+$ are randomly generated.

5.2 Entering Variable Rule by DRAD

To examine the effect of changing the variable entering rule from Dantzig's rule to use DRAD, several randomly generated problem samples at density of one are tested and

reported in Table 5.1 and Table 5.2. As described in Chapter 2, among all the potential variables with negative reduced cost, a variable considered favorable regarding each variable entering rule enters to basis in the next iteration. In problem set with $a_{ij}, b_i, c_j \in \mathcal{R}$, on average, Danzig's rule shows a superior performance compared to other variable pivoting rules. The comparison for the number of the iterations is shown in Table 5.1 and Table 5.2.

Table 5.1 Comparison of the number of the iterations between different variable entering rules

$a_{ij}, b_i, c_j \in \mathcal{R}$ $1 \leq c_j, b_i \leq 10$ and $1 \leq a_{ij} \leq 5$		Number of iterations		
Density	Problem Samples (Variables , Constraints)	Cosine Simplex	DRAD Simplex	Dantzig Simplex
1	800,100	39	20	7
1	700,100	19	6	5
1	600,100	21	8	7
1	500,100	70	29	15
1	400,100	66	35	13
1	300,100	7	9	5
1	200,100	16	15	6
1	100,100	16	21	7
Average		31.75	17.88	8.125

Table 5.2 Comparison of the number of the iterations between different variable entering rules

$a_{ij}, b_i, c_j \in \mathbb{Z}^+$ $1 \leq c_j, b_i \leq 10$ and $1 \leq a_i \leq 5$		Number of iterations		
Density	Problem Samples (Variables , Constraints)	Cosine Simplex	DRAD Simplex	Dantzig Simplex
1	800,100	100	70	22
1	700,100	41	27	9
1	600,100	17	19	11
1	500,100	17	19	6
1	400,100	66	66	16
1	300,100	21	14	13
1	200,100	13	5	13
1	100,100	13	10	7
1	80,80	24	13	8
Average		34.67	27.00	11.67

Consider problem D as the dual of linear programming problem P:

$$\text{(D)} \quad \underset{s.t.}{Min} \sum_{i=1}^{m} \boldsymbol{y_i} * \boldsymbol{b_i} \tag{5.2}$$

$$\sum_{i=1}^{n} \boldsymbol{y_i} * a_{ij} \geq c_j \quad \forall j = 1, \dots, n \tag{5.3}$$

$$\boldsymbol{y_i} \geq 0 \quad \forall i = 1, \dots, m \tag{5.4}$$

DARD measures the likelihood of constraints binding at optimality in problem D.

Theorem: *Complementary Slackness* Assume problem (P) has a solution $\boldsymbol{x}^*$ and Problem (D) has a solution $\boldsymbol{y}^*$.

1. If $x_j^* > 0$, then the jth constraint in (D) is binding.
2. If the jth constraint in (D) is not binding, then $x_j^* = 0$
3. If $y_i^* > 0$, then the ith constraint in (P) is binding.
4. If the ith constraint in (P) is not binding, then $y_i^* = 0$

The complementary slackness theorem identifies a relationship between constraint in one problem and variables in the other problem. *"It says if a variable is*

positive, then the associated dual constraint must be binding. It also says if a constraints fails to bind, the associated variable must be zero" [29].

$$\boldsymbol{y}^*(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}) = 0$$
$$(\boldsymbol{y}^*\boldsymbol{A} - \boldsymbol{c})\boldsymbol{x}^* = 0$$

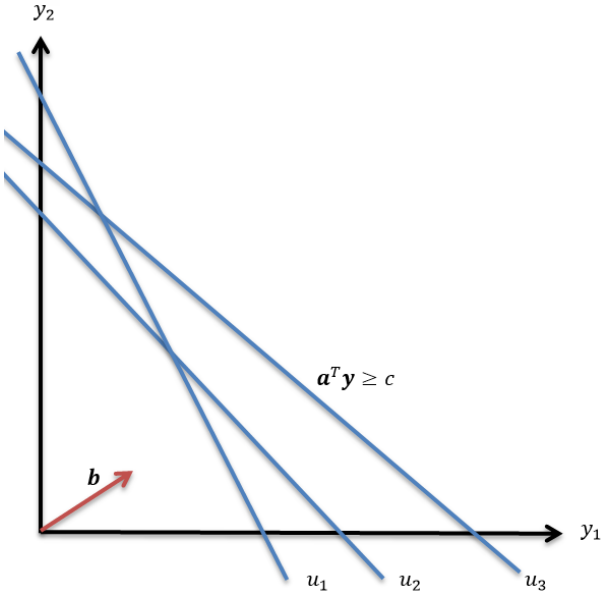

Figure 5.1. Interpretation of DRAD and complementary slackness

Considering the DRAD metric, u_1 and u_2 have a higher probability of getting positive values than u_3. Though u_3 has a high probability to be zero at optimal solution, the associated dual variable in the primal problem can be zero or positive. Consider a two-dimensional example of an NNLP

$$Min\ \ 3x_1 + 3x_2 \qquad \text{(P1)}$$

$$s.t.$$
$$x_1 + x_2 \ >= 3$$
$$2x_1 + x_2 \ >= 4$$
$$3x_1 + x_2 \ >= 6$$
$$x_1, x_2 >= 0$$

The solution to (P1) is $x_1^* = 1.5, x_2^* = 1.5,\ z^* = 9$. As seen, constraint 1 and 3 are bound at optimal solution, but the second constraint is loose ($u_1 = 0, u_2 = 0.5,\ u_3 = 0$). The dual of the (P1) is

$$Max\ 3y_1 + 4y_2 + 6y_3 \quad \text{(D1)}$$

$$s.t.$$

$$y_1 + 2y_2 + 3y_3 <= 3$$

$$y_1 + y_2 + y_3 <= 3$$

$$y_1, y_2, y_3 >= 0$$

Solution $y_1 = 3\ , y_2 = y_3 = 0,\ z^* = 9$

Though constraint 3 in (P1) is tight, its corresponding dual variable in (D1) is 0, and it is not a basic variable at optimal solution.

The provided example may explain the reason for inefficiency of DRAD metric as a variable selection rule. It assumes if a constraint binds at optimal solution in D, its associated dual variable is positive in P, which is not exactly accorded with the complementary slackness theorem.

5.3 DRAD column generation

In this section, DRAD is examined for generating columns in large scale LP problem with a short and wide structure. First, variables are sorted in an ascending order of DRAD (5.1) then CPLEX primal simplex is used to solve P_0. The reduced costs of non-basic variables are obtained by a pricing operation

$$\bar{\boldsymbol{c}}_N = \boldsymbol{c_B} B^{-1} \boldsymbol{a_j} - \boldsymbol{c_j} \qquad (5.5)$$

where $j = \{j \mid \bar{\boldsymbol{c}}_{\boldsymbol{j}} < 0, j \in N\}$. Multiple of columns j are added iteratively, since they price out favorably. First, variables are sorted in a descending order of DRAD. Then, P_0 is constructed by adding 10% of the most important variables. Eq. (3.1) and Eq. (3.2) are used to determine the number of multiple columns added in the next iterations. DRAD

column generation is compared with CPLEX Sifting and CPLEX Primal, CPLEX Dual, and CPLEX Barrier methods, and the results are illustrated on Table 5.3. All methods are tested on problem-sets with 20000 variables and 5000 constraints. The DRAD column generation represents a superior performance compared to Sifting in high density problem instances. On average, DRAD column generation represents 67% improvement compared to the Sifting method on problem-sets 3.

5.4 Dynamic DRAD COST

DRAD is also used as a constraint selection metric to solve NNLP problems. First, for each variable $DRAD(\boldsymbol{a}, \boldsymbol{b}, c)$ is calculated then, problem P is converted to dual problem D, and constraints are sorted in a descending order of DRAD. The dynamic active-set method, discussed in Section 3.2, is used to add multiple violated constraints at problem $P_r, r = 1,2,3, \dots,$. As illustrated in Table 5.3, using DRAD as a constraint selection technique in problem D is the most efficient method for solving NNLPs with short and wide structure. On average, DRAD COST is 57% faster than DRAD Column Generation, 22% faster than NRAD COST, and 85% more efficient than the sifting method, which is the standard LP column generation method.

Table 5.3 The comparison of computation times of DRAD Column Generation, Sifting, Primal, Dual, and Barrier on (Random NNLP with 20000 variables and 5000 constraints, a_{ij} =1 to 5, b_i =1 to 10, c_j =1 to 10)

NO	Density	DRAD COST Dynamic	DRAD Column Generation	NRAD COST	CPLEX Sifting	CPLEX Primal--	CPLEX Dual--	CPLEX Barrier --
				CPU Time (Sec)++				
1	0.005	57.04	90.50	70.12	82.22	240.79	273.74	28.89
2	0.01	39.93	71.00	42.72	70.45	189.77	244.05	29.97
3	0.02	20.54	39.80	22.19	31.72	154.07	189.55	35.81
4	0.03	14.60	31.89	17.19	26.56	148.08	215.39	41.30
5	0.04	10.83	25.55	15.39	24.03	191.50	155.25	47.81
6	0.05	9.22	22.22	13.53	25.52	226.33	146.91	60.25
7	0.06	8.23	20.30	12.99	23.92	197.12	148.64	74.90
8	0.07	7.29	19.50	11.97	24.53	173.60	175.36	90.40
9	0.08	6.66	16.96	11.56	26.56	161.33	187.01	105.60
10	0.09	6.17	16.59	10.87	26.46	159.93	203.78	126.51
11	0.1	5.71	15.90	10.18	28.89	141.78	222.30	140.54
12	0.2	5.84	30.41	9.53	46.92	136.24	202.87	394.03
13	0.3	5.19	30.20	9.25	75.62	137.25	194.21	NA
14	0.4	6.33	30.97	8.74	117.29	126.96	162.46	NA
15	0.5	6.54	25.22	8.41	158.05	122.06	151.68	NA
16	0.6	7.05	24.92	8.59	177.17	127.65	159.51	NA
17	0.7	8.13	21.22	8.74	207.61	109.20	170.25	NA
18	0.8	7.52	19.81	8.40	270.74	108.20	175.91	NA
19	1	8.36	12.95	8.99	239.37	77.22	197.83	NA
Average		12.69	29.78	16.28	88.61	154.16	188.25	NA

++ Average of 5 instances of LP at each density.

+ Used CPLEX preprocessing parameters of presolve = off and predual = off.

-- Used CPLEX preprocessing parameters of presolve = ON and predual = ON.

CHAPTER 6

GRAD for Equality Constraints

The dynamic active-set approach is applied to large-scale NNLPs with equality constraints in this section. This is done as follows.

An $\boldsymbol{A}'$ matrix is formed by converting every equality constraint in $\boldsymbol{A}$ into two inequalities $\boldsymbol{a}_i{}^T\boldsymbol{x} \leq b_i$ and $\boldsymbol{a}_i{}^T\boldsymbol{x} \geq b_i$. The dynamic GRAD, discussed in Section 3.3, is used to solve the general linear programming problem. Note, for an $\boldsymbol{A}$ matrix with all equality constraints, $\boldsymbol{A}'$ is twice as large in its constraints size as $\boldsymbol{A}$. To measure the performance of the proposed method, few NNLPs with equality constraints and $a_{ij}, b_i, c_j \in \mathcal{R}^+$ are randomly generated. To assure the randomly generated NNLPs have a feasible solution, $\boldsymbol{x}^*$ is randomly generated to derive random $\boldsymbol{b}$, where $\boldsymbol{a}_i{}^T\boldsymbol{x}^* = b_i$. In addition, to avoid having a constraint in an upper bound on a variable, each constraint must have at least two non-zero a_{ij}. The number of nonzero a_{ij} in each constraint was binomially distributed *B (n, p* =density).

As illustrated in Table 6.1, GRAD represents an efficiency lower than CPLEX solvers in the majority of problems. The possible explanation for the lack of efficiency in using GRAD is as follows:

- The number of rows in $\boldsymbol{A}'$ is twice as large as the number of rows in $\boldsymbol{A}$, which is a significant increase in the number of constraints.
- The main power of CPLEX is originated from its pre-solver and pre-dual computational advantage. The pre-processing capability of the CPLEX for problems with equality constraints is higher than the problems with inequality constraints. In addition CPLEX uses multiple threads for concurrent optimization which enables CPLEX to do its computation in a parallel mode.

In several problems, CPLEX solver eliminates all rows and columns which makes CPLEX to be extremely efficient for such problems. CPLEX AUTO has several tuning parameters that retune itself during the solving procedure. Some of these parameters are:

- Variable selection: Devix entering rule, Steepest Edge, Dantzig entering rule
- Parallel threads and computing
- Markowitz tolerance
- Cholesky decomposition
- Pre-processing (pre-solver)
- Pre-dual

Table 6.1 The computation times of dynamic GRAD and CPLEX used for NNLPs with equality constraints(Random NNLP,$1 \leq c_j \leq 10$ and $1 \leq a_{ij} \leq 5$, and $1 \leq x_{ij} \leq 10$)

Density	NNLPs (Var , Cons)	Dynamic GRAD	CPLEX Primal[--]	CPLEX Barrier [--]	CPLEX AUTO
		CPU Time			
0.01	1000,50000	1.24	0.12	0.29	0.09
0.05	2000,25000	163.02	78.53	307.09	73.66
0.05	1000,50000	20.00	63.82	125.41	4.88
0.08	1000,50000	39.53	73.19	147.78	5.56
0.1	1000,50000	13.98	75.85	155.54	6.81
0.5	1000,50000	125.31	103.37	191.76	28.21
0.005	1000,100000	0.66	0.13	0.4	0.13
0.05	1000,100000	19.07	147.17	261.13	7.12
0.1	1000,100000	20.86	163.46	319.72	11.39
0.005	2000,50000	8.82	0.11	0.24	0.09
0.05	2000,50000	177.98	162.92	592.57	70.80
0.1	2000, 50000	322.15	195.65	676.18	60.67

[--] Used CPLEX preprocessing parameters of presolve = ON and predual = ON.

CPLEX pre-solver and pre-dual is more powerful and efficient in LP problems with equality constraints than inequality constraints. Table 6.2 compares the CPU time for the same NNLPs with equality and inequality constraints. In all problem instances, the CPU

time for solving NNLP problems with equality constraints is smaller than the same problem with inequality constraints. On average, CPLEX pre-solver and pre-dual are 8 times more efficient for NNLPs with equality constraints than the same problem with inequality constraints. The CPU time of COST NRAD is still larger than CPLEX. Interestingly, for equality constraints, CPLEX pre-solver eliminates all columns and rows which confirms the main computational power of CPLEX is from its preprocessing (pre-solver and pre-dual) parameters.

Table 6.2 A comparison of the CPLEX pre-solver's performance on equality and inequality constraints (Random NNLP, $1 \leq c_j \leq 10$ and $1 \leq a_{ij} \leq 5$, and $1 \leq x_{ij} \leq 10$)

Density		CPU Time⁺		CPU Time
Density	NNLPs (Variables , Constraints)	CPLEX AUTO	CPLEX AUTO	COST NRAD
		Inequality $\boldsymbol{a}_i{}^T\boldsymbol{x} \leq b_i$	Equality $\boldsymbol{a}_i{}^T\boldsymbol{x} = b_i$	Inequality $\boldsymbol{a}_i{}^T\boldsymbol{x} \leq b_i$
0.01	1000,50000	0.80	0.09	0.43
0.1	1000,50000	0.34	0.10	0.11
0.05	1000, 100000	77.29	6.62	6.65
0.1	1000, 100000	108.11	7.22	14.92

⁺Used CPLEX preprocessing parameters of presolve = ON and predual = ON.

After considering the results, it was realized that the way that these problems are made effectively give a unique solution for a set of the linear equations in which in that case, CPLEX had to solve a system of overdetermined linear equations. Therefore, some further problems are constructed with the mixed equality and inequality constraints.

The dynamic COST RAD for solving NNLPs problems with mixed equality and equality constrains is described as follows.

Constraints are initially ordered by the RAD constraint selection metric (2.2). In order to solve problems with equality and equality constrains, all constraints are ranked in a descending order of RAD. Then, P_0 is formed by adding all of the $\boldsymbol{a}_i{}^T\boldsymbol{x} = b_i$ constraints. We add constraints $\boldsymbol{a}_i{}^T\boldsymbol{x} \leq b_i$ from (1.2) in descending order of RAD until each variables

x_j has an $a_{ij} > 0$ in the coefficient matrix of P_0. Then, dynamic RAD discussed in Section 3.2 is applied to solve the problem. As illustrated in Table 6.3, the proposed methodology solves the mixed constraints problem very efficiently.

Table 6.3 A comparison of the CPLEX's performance on mixed equality-inequality constraints NNLPs and dynamic RAD.

Density	NNLPs (Variables , Constraints ($\boldsymbol{a}_i{}^T\boldsymbol{x} \leq b_i,\ \boldsymbol{a}_i{}^T\boldsymbol{x} = b_i$)	CPLEX AUTO	Dynamic RAD
		CPU Time	CPU Time[+]
0.01	(1000, 200000)	9.86	0.88
0.01	(1000, 200000)	10.47	0.9
0.03	(1000, 200000)	78.97	1.21
0.04	(1000, 200000)	129.44	1.20
0.05	(1000, 200000)	144.49	1.32
0.1	(1000, 200000)	407.47	1.72
0.05	(5000, 20000)	150.34	5
0.1	(5000, 20000)	367.56	4.09

[+]Used CPLEX preprocessing parameters of presolve = ON and predual = ON.

CHAPTER 7

Conclusions

Two efficient dynamic active-set approaches were developed. In addition, a constraint selection rule NVRAD is developed, and its geometric interpretation was given. The performance of the developed methods was tested on sets of large-scale generated GLPs and NNLPs. The superior performance of the developed method over the COST NRAD multi-cuts is presented. Also, both Dynamic active-set methods significantly outperformed all of the CPLEX solver methods in solving GLPs and NNLPs with various densities from a low density to a density of 1. The improvement achieved on GLPs was more significant compared to the developed method by [18].

The dynamic active-set approach, presented in Chapter 4, increases the performance of the posterior methods, such as VIOL and NVIOL, significantly compared to the bounding technique. The importance of the posterior methods on the sparse matrix was illustrated. Also, a hybrid method is efficient, and it is understood that hybrid approaches can take advantage of both global and local information. A variable selection technique, based on the dual version of the radial constraint selection rule (DRAD), is implemented on a problem sets. The results are compared with the Sifting method, which is a popular method for generating column in LP problems with short and wide structure.

The study can also be extended to address the cases when all the equations are equalities $a^T x = b$. Also, the reduced problem by CPLEX pre-solver can be used to speed up the solving process. Other areas of future research may contain expanding RAD to solve integer programming problems.

References

[1]. Bixby, R.E., et al., *Very large-scale linear programming: a case study in combining interior point and simplex methods.* Operations research, 1992. **40**(5): p. 885-897.

[2]. Rosenberger, J.M., E.L. Johnson, and G.L. Nemhauser, *Rerouting Aircraft for Airline Recovery.* Transportation Science, 2003. **37**(4): p. 408-421.

[3]. Todd, M.J., *The many facets of linear programming.* Mathematical Programming, 2002. **91**(3): p. 417-436.

[4]. Elwes, R., *The algorithm that runs the world.* New Scientist, 2012. **215**(2877): p. 32-37.

[5]. Dare, P. and H. Saleh, *GPS network design: logistics solution using optimal and near-optimal methods.* Journal of Geodesy, 2000. **74**(6): p. 467-478.

[6]. Saito, G., et al., *Constraint Optimal Selection Techniques (COSTs) for nonnegative linear programming problems.* Applied Mathematics and Computation, 2015. **251**: p. 586-598.

[7]. Li, H.-L. and C.-J. Fu, *A linear programming approach for identifying a consensus sequence on DNA sequences.* Bioinformatics, 2005. **21**(9): p. 1838-1845.

[8]. Stone, J.J., *The cross-section method, an algorithm for linear programming*. 1958, DTIC Document.

[9]. Thompson, G.L., F.M. Tonge, and S. Zionts, *Techniques for removing nonbinding constraints and extraneous variables from linear programming problems.* Management Science, 1966. **12**(7): p. 588-608.

[10]. Adler, I., R. Karp, and R. Shamir, *A family of simplex variants solving an m× d linear program in expected number of pivot steps depending on d only.* Mathematics of operations research, 1986. **11**(4): p. 570-590.

[11]. Zeleny, M., *An external reconstruction approach (ERA) to linear programming.* Computers & Operations Research, 1986. **13**(1): p. 95-100.

[12]. Myers, D.C. and W. Shih, *A constraint selection technique for a class of linear programs.* Operations Research Letters, 1988. **7**(4): p. 191-195.

[13]. Curet, N.D., *A primal-dual simplex method for linear programs.* Operations Research Letters, 1993. **13**(4): p. 233-237.

[14]. Barnhart, C., et al., *Branch-and-price: Column generation for solving huge integer programs.* Operations research, 1998. **46**(3): p. 316-329.

[15]. Mitchell, J.E., *Computational experience with an interior point cutting plane algorithm.* SIAM Journal on Optimization, 2000. **10**(4): p. 1212-1227.

[16]. Corley, H.W. and J.M. Rosenberger, System, method and apparatus for allocating resources by constraint selection, 2011.

[17]. Corley, H.W., et al., *The cosine simplex algorithm.* The International Journal of Advanced Manufacturing Technology, 2006. **27**(9-10): p. 1047-1050.

[18]. Saito, G., H.W. Corley, and J. Rosenberger, *Constraint Optimal Selection Techniques (COSTs) for Linear Programming.* 2012.

[19]. Charnes, A. and W.W. Cooper, *On some works of Kantorovich, Koopmans and others.* Management Science, 1962. **8**(3): p. 246-263.

[20]. Dantzig, G.B. and M.N. Thapa, *Linear Programming 1: 1: Introduction*. Vol. 1. 1997: Springer.

[21]. Cottle, R., E. Johnson, and R. Wets, *George B. Dantzig (1914–2005).* Notices of the AMS, 2007. **54**(3): p. 344-362.

[22]. Khachiyan, L.G., *Polynomial algorithms in linear programming.* USSR Computational Mathematics and Mathematical Physics, 1980. **20**(1): p. 53-72.

[23]. Thapa, G.B.D.M.N., *Linear programming.* 2003.

[24]. Wright, S.J., *Primal-dual interior-point methods*. Vol. 54. 1997: Siam.

[25]. Desaulniers, G., J. Desrosiers, and M.M. Solomon, *Column generation*. Vol. 5. 2005: Springer.

[26]. Pan, P.-Q., *Efficient nested pricing in the simplex algorithm.* Operations Research Letters, 2008. **36**(3): p. 309-313.

[27]. Corley, H., et al., *The cosine simplex algorithm.* The International Journal of Advanced Manufacturing Technology, 2006. **27**(9-10): p. 1047-1050.

[28]. Tichý, M., *Applied methods of structural reliability*. Vol. 2. 2012: Springer Science & Business Media.

[29]. Luis, G., *Proving Optimality for Linear Programs* 2005.

Biographical Information

Alireza Noroziroshan was born in IRAN where he earned his bachelor's degree in Industrial and System Engineering from Mazandaran University of Science and Technology. In 2007 he started his Master's program in Industrial Engineering at the Universiti PUTRA Malaysia. During his Master's program he got interested into operation research and optimization. He came to US for his PhD at The Department of Industrial, Manufacturing and Systems Engineering, University of Texas at Arlington n 2011. During his course of stay at UTA he worked under the supervision of Dr. Bill Corley. He worked on developing techniques for solving large-scale linear programming problems. He also worked as a teaching assistant at the Department of Industrial and Manufacturing Systems Engineering. His research interests are optimizations and data analytics.