An Algorithm For Cardinality Constrained Quadratic Optimization Problems

Dimitris Bertsimas *　　　Romy Shioda †

May, 2004

Abstract

In this paper, we describe an implicit enumeration algorithm for cardinality-constrained quadratic mixed-integer optimization problems motivated by applications in subset selection in regression and portfolio selection in asset management. The proposed algorithm uses the special structure of these problems by implementing a combination of implicit branch-and-bound, Lemke's pivoting method, variable deletion and problem reformulation. We compare the performance of the method against widely used heuristic methods and CPLEX 8.0's quadratic mixed-integer solver, and conclude that the proposed algorithm has significant advantages over these methods for the special class of problems we consider.

1 Introduction

We present a method for solving quadratic variable selection problems, i.e., quadratic optimization problems that limit the number of non-zero variables in the solution, motivated by applications in subset selection in regression and portfolio selection in asset management. Specifically, we consider the

*Boeing Professor of Operations Research, Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, E53-363, Cambridge, MA 02139, dbertsim@mit.edu. The research of the author was partially supported by the Singapore-MIT alliance.

†Department of Combinatorics and Optimization. Faculty of Mathematics, University of Waterloo, Waterloo, ON N2L 3G1, Canada. rshioda@math.uwaterloo.ca. The research of the author was partially supported by the Singapore-MIT alliance, the Discovery Grant from NSERC and a research grant from the Faculty of Mathematics, University of Waterloo.

following problem:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}'\boldsymbol{x}, \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & |supp(\boldsymbol{x})| \leq K, \\ & x_i \geq \alpha_i, \qquad i \in supp(\boldsymbol{x}), \\ & 0 \leq x_i \leq u_i, \qquad i = 1, \ldots, d, \end{aligned} \tag{1}$$

where $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$ is symmetric positive semi-definite, $\boldsymbol{c} \in \mathbb{R}^d$, $\boldsymbol{A} \in \mathbb{R}^{m\times d}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\alpha_i > 0$, u_i is the nonnegative upper bound of x_i, K is some positive integer, and $supp(\boldsymbol{x}) = \{i | x_i \neq 0\}$. The second set of constraints, referred to as the cardinality constraint, and the third set of constraints, referred to as the lower bound constraints, introduce discreteness to the problem, making this a quadratic mixed-integer optimization problem.

Compared to linear integer optimization, quadratic mixed-integer optimization problems have received relatively little attention in the literature. In the first study of problems of type (1), [5] proposes an implicit branch-and-bound algorithm and replaces the cardinality constraint, $|supp(\boldsymbol{x})| \leq K$, with a surrogate constraint, $\sum_i (x_i/u_i) \leq K$. Moreover, the $\boldsymbol{x}$ variables are branched on directly, i.e., when branching down on x_j, the constraint $x_j \leq 0$ is added, and when branching up on x_j, the constraint $x_j \geq \alpha_i$ is added. However, [5] still solves the quadratic relaxation using gradient methods that have difficulty starting from an infeasible point. Also, it only considers the case where $\alpha_i = 0$ for all i. We instead utilize an idea, first proposed in [21] in the context of pure 0-1 quadratic programming, to use Lemke's pivoting algorithm to solve the successive subproblems in the enumeration tree when variables are restricted to zero or one, without adding constraints.

Our objective in this paper is to propose a computationally efficient method that solves Problem (1). The proposed method combines implicit branch-and-bound, Lemke's method [19, 8], variable deletion and reformulation techniques. Section 2 elaborates on this general methodology for solving cardinality constrained quadratic mixed-integer optimization problems. We further tailor our method to solve two important problems in Statistics and Finance: subset selection in regression and portfolio selection in finance.

Subset Selection in Regression

In traditional multivariate regression, we are given m data points $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, and we want to find $\boldsymbol{\beta} \in \mathbb{R}^d$ such that $\sum_i (y_i - \boldsymbol{x}_i'\boldsymbol{\beta})^2$ is minimized. This has a closed-form solution, $\boldsymbol{\beta} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{Y}$, where $\boldsymbol{X} \in \mathbb{R}^{m \times d}$ with $\boldsymbol{x}_i'$ as its i^{th} row, and $\boldsymbol{Y} \in \mathbb{R}^m$ with y_i as its i^{th} element.

Primarily for robustness purposes, i.e., to limit the variance of the predicted $\boldsymbol{Y}$, it is desirable to use a small subset of the variables (see [22, 26, 1]). [2], [14] and [12] use pivoting and branch-and-bound techniques to search for subsets with the best regression "fit" (e.g., minimum total sum of squared errors) for subsets of all sizes. [23] solve linear mixed-integer optimization problems that find a subset of K variables ($K < d$) that has the minimum total absolute error.

The above exact enumeration methods, however, have been deemed impractical for large problems (e.g., $d > 20$, $K > 6$) to be solved in practical time (see [26]). Currently, simple greedy heuristics, such as forward, backwards and stepwise regression, are the most commonly used methods to choose which variables to include and exclude [22, 26].

We solve the problem of choosing a subset of K variables that minimizes the total sum of squared errors to optimality, by modelling it as an unconstrained, free-variable version of Problem (1). We will show that our implicit branch-and-bound procedures, together with efficient computation of the optimal objective value at each node, finds solutions that significantly out-performs forward regression, in terms of total sum of squares, within few seconds. Section 3.1 elaborates on our method, and Section 4.1 illustrates the performance of our approach compared to forward regression heuristic and an explicit branch-and-bound procedure.

Optimal Portfolio Selection Problem

The problem of constructing a portfolio of K stocks that minimizes the total variance, while ensuring that the expected total rate of return is greater than a given threshold, can be formulated as a convex quadratic optimization problem. Large asset management companies manage assets against a benchmark that has d securities. So that they are not seen as indexers, it is desirable that they only use $K \ll d$ securities in the portfolio. In addition, several portfolios are marketed as focused funds that are only allowed by their prospectus to own a small collection of securities. Finally, asset management companies that manage separate accounts for their clients that only have say \$100,000 can only realistically own a small number of securities, since otherwise, the transaction costs would significantly affect performance. Such limited diversification constraints, along with fixed transaction costs and minimum transaction levels, present discrete constraints and variables to the quadratic problem (see [3]).

Given the difficulty of solving quadratic integer optimization problems, such a portfolio problem has commonly been approached in one of two ways. The first approach is approximating the problem to a simpler form. For example, [27, 28] approximates the quadratic term by a linear and piece-wise linear function. [15] similarly linearizes the quadratic objective function, and further assumes equal weights across assets to formulate the problem as a pure 0-1 problem. [25] shows how the resulting portfolio problem with fixed transaction costs can be solved efficiently when the covariance matrix has all off diagonal elements equal. The second approach uses heuristics to find strong feasible solutions. [20] proposes a linear mixed-integer optimization based heuristic, and [6] introduces a dynamic programming based heuristic. [7] proposes genetic algorithm, tabu search and simulated annealing approaches for the problem.

There has been, however, significant efforts in finding exact algorithms to efficiently solve the discrete portfolio problem. [24] extends the implicit branch-and-bound techniques of [5] to limited diversification portfolios. [16] implements FortQP as the quadratic solver, and solve portfolio problems with minimum transaction levels, limited diversification and roundlot constraints (which requires investing in discrete units) in a branch-and-bound context. FortQP uses an approach similar to the simplex method to solve the quadratic problem.

We propose to solve portfolio problems with limited diversification via implicit branch-and-bound, Lemke's pivoting method, variable deletion and reformulation. We elaborate on our methodology in Section 3.2 and illustrate computational results in Section 4.2.

2 General Methodology

In a branch-and-bound setting, we solve the continuous relaxation of Problem (1) via Lemke's method, then choose a branching variable x_s. When branching down, we update the subsequent subproblem by deleting the data associated to x_s, and when branching up, we reformulate the problem or modify Lemke's method so that $x_s \geq \alpha_s$.

The continuous relaxation we solve at each node has the following form:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}'\boldsymbol{x}, \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{2}$$

where the cardinality constraint and lower bound constraints are removed. We do not replace the cardinality constraint with [5]'s surrogate constraint, $\sum_i (x_i/u_i) \leq K$, because u_i is unknown in the subset selection problem and the constraint is dominated by a constraint in $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ in the portfolio selection problem. The lower bound constraint, $x_i \geq \alpha_i$ for x_i strictly positive, are enforced either by the reformulation procedure in the branching up step or implicitly enforced by Lemke's method. Section 2.1 describes the use of Lemke's method to solve Problem (2) in the context of branch-and-bound. Section 2.2 illustrates the procedure for updating the subproblem after the branching variable is deleted, and Section 2.3 describes our reformulation procedure and Lemke's method with lower bounds to enforce the lower bound constraint without adding new constraints or changing the structure of the problem.

2.1 Lemke's Method as Underlying Quadratic Optimizer

We use Lemke's pivoting method to optimize the continuous relaxation of the subproblem at each node. This method was originally developed to solve linear complementarity problems (of which quadratic programs are a special case) via pivoting akin to the simplex method. As with the dual simplex method in linear optimization, the key advantage of Lemke's method is its ease and efficiency of starting from an infeasible point. This is critical in the branch-and-bound setting, since the optimal solution of the parent node can be used as the initial point to solve the problem of the current node. Thus, this solver has a significant advantage over interior point methods which needs to solve from scratch at each node.

First, we will introduce linear complementarity problems and their extensions to quadratic optimization. Then we present an overview of the Lemke's method to solve LCPs. Finally, we describe the implementation of Lemke's method in the context of branch-and-bound.

The Linear Complementarity Problem

A linear complementarity problem (LCP) is the following: Given $\boldsymbol{q} \in \mathbb{R}^n$ and $\boldsymbol{M} \in \mathbb{R}^{n\times n}$, find $\boldsymbol{z} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^n$ such that,

$$\boldsymbol{w} = \boldsymbol{M}\boldsymbol{z} + \boldsymbol{q}, \tag{3}$$

$$\boldsymbol{z} \geq \boldsymbol{0}, \boldsymbol{w} \geq \boldsymbol{0}, \tag{4}$$

$$\boldsymbol{z}'\boldsymbol{w} = 0. \tag{5}$$

The above problem is referred to as LCP$(\boldsymbol{q}, \boldsymbol{M})$. A vector $\boldsymbol{z}$ is called *feasible* if it satisfies Equations (3) and (4), and *complementary* if it satisfies (5). A vector $\boldsymbol{z}$ that is both feasible and complementary is called an *equilibrium point* or a *solution* of LCP$(\boldsymbol{q}, \boldsymbol{M})$. Note that due to the nonnegativity of the variables, $\boldsymbol{z}$ is complementary if and only if

$$z_i w_i = 0 \qquad \forall i = 1, \ldots, n. \tag{6}$$

The pair z_i and w_i are called *complementary pairs.*

Now, suppose we have the following quadratic optimization problem:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{x}'\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}'\boldsymbol{x}, \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{7}$$

where $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$, $\boldsymbol{A} \in \mathbb{R}^{m\times d}$ and $\boldsymbol{b} \in \mathbb{R}^{m}$ are given. For symmetric positive semi-definite matrix $\boldsymbol{Q}$, the KKT conditions for (7), which are necessary and sufficient for finding global minima, are as follows:

$$\begin{aligned} \boldsymbol{c} + \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{A}'\boldsymbol{y} - \boldsymbol{u} &= \boldsymbol{0}, \\ \boldsymbol{y}'(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) &= 0, \\ \boldsymbol{u}'\boldsymbol{x} &= 0, \\ \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u} &\geq \boldsymbol{0}, \end{aligned} \tag{8}$$

where $\boldsymbol{y} \in \mathbb{R}^{m}$ and $\boldsymbol{u} \in \mathbb{R}^{d}$ are the dual variables. By introducing another variable $\boldsymbol{v} \in \mathbb{R}^{m}$, (8) can be written as:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{c} + \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{A}'\boldsymbol{y}, \\ \boldsymbol{v} &= \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}, \\ \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v} &\geq \boldsymbol{0}, \\ \boldsymbol{x}'\boldsymbol{u} = 0, \boldsymbol{y}'\boldsymbol{v} &= 0. \end{aligned} \tag{9}$$

The system in (9) can be represented as an LCP$(\boldsymbol{q}, \boldsymbol{M})$ with

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{b} \end{bmatrix}, \boldsymbol{M} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}' \\ -\boldsymbol{A} & \boldsymbol{0} \end{bmatrix}, \tag{10}$$

and $\boldsymbol{z}$ and $\boldsymbol{w}$ of (3)-(5) as

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix}.$$

In order to speak of bases and basic variables, let us first re-write (3) as

$$\boldsymbol{I}_n\boldsymbol{w} - \boldsymbol{M}\boldsymbol{z} = \boldsymbol{q}, \qquad \text{or} \qquad \begin{bmatrix} \boldsymbol{I}_n, -\boldsymbol{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{z} \end{bmatrix} = \boldsymbol{q},$$

where $\boldsymbol{I_n}$ is an n-dimensional identity matrix. We can find a feasible solution to Equations (3)-(4), by selecting n linearly independent columns of $\left[\boldsymbol{I_n}, -\boldsymbol{M}\right]$, forming the basis $\boldsymbol{B}$, such that $\boldsymbol{B}^{-1}\boldsymbol{q} \geq \boldsymbol{0}$. To further get a complementary solution to Equations (3)-(5), the basis contains z_i if and only if w_i is not in the basis, and vice versa. Let β be the set of indices of $\boldsymbol{z}$ that are in the basis. Then the associated *complementary basis*, $\boldsymbol{B}$, is such that:

$$\boldsymbol{B}_{.i} = \begin{cases} -\boldsymbol{M}_{.i} & \text{if} \quad i \in \beta \\ \boldsymbol{e}_i & \text{if} \quad i \notin \beta \end{cases} \tag{11}$$

where $\boldsymbol{B}_{.i}$ is the i^{th} column of $\boldsymbol{B}$ and $\boldsymbol{e}_i$ is an unit vector with the i^{th} element equal to 1. The goal of Lemke's method is to find such a complementary basis, $\boldsymbol{B}$, such that $\boldsymbol{B}^{-1}\boldsymbol{q} \geq 0$. The following section elaborates on this methodology.

Lemke's Method

Given $\boldsymbol{q}$ and $\boldsymbol{M}$ and some user-defined *covering vector* $\boldsymbol{h} \in \mathbb{R}^n$, where $\boldsymbol{h} > \boldsymbol{0}$, Lemke's method solves LCP$(\boldsymbol{q}, \boldsymbol{M})$ via series of pivot operations. First, it checks whether $\boldsymbol{q} \geq \boldsymbol{0}$, in which case $\boldsymbol{z} = \boldsymbol{0}$ and $\boldsymbol{w} = \boldsymbol{q}$ is a solution. Otherwise, it augments LCP$(\boldsymbol{q}, \boldsymbol{M})$ to

$$\boldsymbol{w} = \boldsymbol{q} + \boldsymbol{h}z_0 + \boldsymbol{M}\boldsymbol{z} \geq \boldsymbol{0}, \quad z_0 \geq 0, \quad \boldsymbol{z} \geq \boldsymbol{0}, \quad \boldsymbol{z}'\boldsymbol{w} = 0. \tag{12}$$

Akin to the dual simplex method, we need to start the algorithm with a complementary basis that does not necessarily satisfy the nonnegativity constraint (4). A simple default basis is to have all the $\boldsymbol{z}$ variables be nonbasic and $\boldsymbol{w}$ be basic. We then set the auxiliary variable z_0 to the smallest positive value such that $\boldsymbol{w} \geq \boldsymbol{0}$ when $\boldsymbol{z} = \boldsymbol{0}$, i.e., $z_0 = \max_i(-q_i/h_i), i = 1, \ldots, d$. Thus, $z_i w_i = 0, i = 1, \ldots, n$ and $z_0 > 0$, and z_0 is pivoted into the basis in place of w_r, where $r = \text{argmax}_i(-q_i/h_i)$. Such a point is called an *almost complementary* point for the augmented problem (12). The algorithm follows a path from one almost complementary basic solution to the next, until z_0 is pivoted out to be a nonbasic variable or LCP$(\boldsymbol{q}, \boldsymbol{M})$ is shown to be infeasible [18].

After the initial pivot, there are exactly one complementary pair that are both nonbasic variables (for all other pairs, one of the complement is basic and the other is nonbasic). Such a pair is called the *nonbasic pair*. In each iteration, one of the nonbasic pair just became nonbasic via the previous pivot. Its complement can thus be increased to a positive value and maintain the almost complementary condition. This complement, called the *driving variable*, is increased until one of the basic variables becomes zero. This basic variable that blocks further increase of the driving variable is called the *blocking variable.* The driving variable enters into the basis in place of the blocking variable which becomes nonbasic. This process is repeated until either

1. z_0 becomes the blocking variable, or

2. We fail to find a blocking variable.

In the first case, z_0 is pivoted out of the basis and set to 0. Thus, we have a basis that is both feasible and complementary for the original problem. If we encounter the second case, we can show that the original linear complementarity problem and thus the quadratic optimization problem (7) is infeasible.

The user-defined covering vector, $\boldsymbol{h} \in \mathbb{R}^n$, is set to the default value of $h_i = 1, \quad i = 1, \ldots, n$. Although, the h_i's need only be strictly positive, it has been shown that different choices of $\boldsymbol{h}$ can greatly effect the number of pivots for some instances of the problem. Smart selection of the covering vector is a problem on its own and needs further investigation. Assuming that the Problem (12) is nondegenerate (an assumption that can be relaxed with certain lexicographic pivoting rules discussed in section 4.9 of [8]), Lemke's method is a finite algorithm, since there are only finite number of complementary bases. Also, given that the matrix $\boldsymbol{Q}$ is positive semi-definite, the value of z_0 monotonically decreases at each pivot.

Lemke's Method in the Context of Branch-and-Bound

As in linear mixed-integer optimization, we want to be able to resolve a new subproblem starting with the basic solution resulting from the parent subproblem. In the linear case, the dual simplex method can easily start from an infeasible basis, and thus is key to solving each subproblem efficiently. The same is true for Lemke's method, which can start from an infeasible complementary basis. The following describes this resolve procedure.

Suppose $\hat{\boldsymbol{Q}}$, $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{c}}$ and $\hat{\boldsymbol{b}}$ are the input data for the current subproblem (2), and $\hat{\boldsymbol{M}}$ and $\hat{\boldsymbol{q}}$ are constructed according to (10). At an intermediary pivot step, with $\boldsymbol{B}$ as the complementary basic columns and $\boldsymbol{N}$ as the nonbasic columns of $\hat{\boldsymbol{M}}$, the full tableau would look as follows:

$$\left[\boldsymbol{q}_B \;\middle|\; \boldsymbol{I} \quad -\boldsymbol{M}_B\right],$$

where $\boldsymbol{q}_B = \boldsymbol{B}^{-1}\hat{\boldsymbol{q}}$ and $\boldsymbol{M}_B = -\boldsymbol{B}^{-1}\boldsymbol{N}$. This tableau is pivoted until the auxiliary variable z_0 is pivoted out of the basis. As in the revised simplex method, we maintain $\boldsymbol{B}^{-1}$, $\boldsymbol{M}_B$ and $\boldsymbol{q}_B$ at each pivot iteration. Suppose Lemke's method terminates with $\boldsymbol{B}^*$, $\boldsymbol{N}^*$ and thus $\boldsymbol{M}_B^* = -\boldsymbol{B}^{*-1}\boldsymbol{N}^*$ and $\boldsymbol{q}_B^* = \boldsymbol{B}^{*-1}\boldsymbol{q}$. The solution to this subproblem is $x_i = q_{B(i)}^*$ if x_i is the $B(i)^{th}$ basic variable, and $x_i = 0$ if x_i is nonbasic.

When constructing the subsequent subproblem after branching, all or some of the data (namely, $\hat{\boldsymbol{Q}}$, $\hat{\boldsymbol{A}}$, $\hat{\boldsymbol{c}}$ and or $\hat{\boldsymbol{b}}$) may be modified. Let $\overline{\boldsymbol{M}}$ and $\overline{\boldsymbol{q}}$ correspond to $\hat{\boldsymbol{M}}$ and $\hat{\boldsymbol{q}}$, respectively, using the modified data. Also, let $\overline{\boldsymbol{B}}$ and $\overline{\boldsymbol{N}}$ be the basis and the nonbasic columns, respectively, after the modification. To solve the next subproblem, we want Lemke's method to start with $\overline{\boldsymbol{B}}$ as its initial complementary basis, instead of its default basis. Empirical testing shows that starting from the previous basis dramatically reduces the total number of pivots required to pivot out the auxiliary variable z_0. In this case, Lemke's initializes with $\boldsymbol{M}_{\bar{B}} = -\overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{N}}$ and $\boldsymbol{q}_{\bar{B}} = \overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{q}}$. Since recomputing the inverse basis and the matrix multiplication to compute $\boldsymbol{M}_{\bar{B}}$ is expensive, Section 2.2 and 2.3 describes methods to update the new inputs efficiently after branching down and up, respectively.

2.2 Branching Down

When branching down on x_s, we delete all the data associated to x_s for the subsequent subproblems and update the basis accordingly. We chose to delete the variable instead of explicitly adding the constraint $x_s = 0$ to prevent increasing the size of the subproblem as well as for numerical stability purposes. We will show that in most cases, the inverse of the new basis can be efficiently derived from the inverse of the old basis by using elementary row operations.

Let us assume that x_s is a basic variable in the previous solution (note that if x_s is a nonbasic variable, we can apply the following methodology for its complementary variable, w_s, which must be a basic variable). We delete the column and row of $\boldsymbol{B}$ corresponding to $\boldsymbol{x}_s$ and the column and row of $\boldsymbol{N}$ corresponding to its dual variable $\boldsymbol{w}_s$, where $\boldsymbol{B}$ is the basis and $\boldsymbol{N}$ are nonbasic columns of the solution to the subproblem of the parent node (i.e., if x_s is the i^{th} basic variables, then we delete the i^{th} column and s^{th} row of $\boldsymbol{B}$. Similarly for $\boldsymbol{w}_s$ and $\boldsymbol{N}$). Although we can get the new inverse of the basis simply by inverting the modified basis, calculating the inverse can be a significant bottleneck. Instead, we calculate the new inverse from the previous inverse using elementary row operations.

Suppose, for notational purposes, that the column and row needed to be deleted in $\boldsymbol{B}$ are the first column and first row (i.e., x_s is the first basic variable and $s = 1$), and $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, so that:

$$\boldsymbol{B} = \begin{bmatrix} v & \boldsymbol{B}'_{row} \\ \boldsymbol{B}_{col} & \overline{\boldsymbol{B}} \end{bmatrix}, \qquad \boldsymbol{B}^{-1} = \begin{bmatrix} u & \boldsymbol{U}'_{row} \\ \boldsymbol{U}_{col} & \overline{\boldsymbol{U}} \end{bmatrix},$$

where $\overline{\boldsymbol{B}}$ and $\overline{\boldsymbol{U}}$ are $n-1$ by $n-1$ lower-right submatrices of $\boldsymbol{B}$ and $\boldsymbol{B}^{-1}$, respectively, $\boldsymbol{B}_{col}$, $\boldsymbol{B}_{row}$, $\boldsymbol{U}_{col}$ and $\boldsymbol{U}_{row}$ are $(n-1)$-dimensional column vectors, and v and u are scalars. We know that:

$$\boldsymbol{B}^{-1}\boldsymbol{B} = \begin{bmatrix} uv + \boldsymbol{U}'_{row}\boldsymbol{B}_{col} & u\boldsymbol{B}'_{row} + \boldsymbol{U}'_{row}\overline{\boldsymbol{B}} \\ v\boldsymbol{U}_{col} + \overline{\boldsymbol{U}}\boldsymbol{B}_{col} & \boldsymbol{U}_{col}\boldsymbol{B}'_{row} + \overline{\boldsymbol{U}\boldsymbol{B}} \end{bmatrix} = \boldsymbol{I}_n.$$

Thus we get

$$\overline{\boldsymbol{U}\boldsymbol{B}} = \boldsymbol{I}_{n-1} - \boldsymbol{U}_{col}\boldsymbol{B}_{row}'. \tag{13}$$

Since $\boldsymbol{U}_{col}\boldsymbol{B}'_{row}$ is a rank one matrix, we can execute linear number of elementary row operations to the matrix $\boldsymbol{I}_{n-1} - \boldsymbol{U}_{col}\boldsymbol{B}'_{row}$ to get $\boldsymbol{I}_{n-1}$. Let $\boldsymbol{E}$ be the matrix representing those operations. Then if $\overline{\boldsymbol{B}}$ is invertible, then $\boldsymbol{E}\overline{\boldsymbol{U}}$ is the inverse matrix of $\overline{\boldsymbol{B}}$.

In the previous section, we stated that we use $\boldsymbol{M}_{\bar{B}} = -\overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{N}}$ as input to Lemke's. We avoid this matrix multiplication via similar elementary row operations. Suppose $\boldsymbol{N}$ are the nonbasic columns and $\boldsymbol{M}_B = -\boldsymbol{B}^{-1}\boldsymbol{N}$ at termination of Lemke's at the parent node. Again, let us assume that the column corresponding to w_s in $\boldsymbol{N}$ is the first one. Then,

$$\boldsymbol{M}_B = -\boldsymbol{B}^{-1}\boldsymbol{N} = -\begin{bmatrix} u & \boldsymbol{U}'_{row} \\ \boldsymbol{U}_{col} & \overline{\boldsymbol{U}} \end{bmatrix}\begin{bmatrix} p & \boldsymbol{N}'_{row} \\ \boldsymbol{N}_{col} & \overline{\boldsymbol{N}} \end{bmatrix} = \begin{bmatrix} w & \boldsymbol{M}'_{row} \\ \boldsymbol{M}_{col} & \overline{\boldsymbol{M}} \end{bmatrix},$$

where $\overline{\boldsymbol{N}}$ and $\overline{\boldsymbol{M}}$ are $n-1$ by $n-1$ lower-right submatrices of $\boldsymbol{N}$ and $\boldsymbol{M}_B$, respectively, and $\boldsymbol{N}_{col}$, $\boldsymbol{N}_{row}$, $\boldsymbol{M}_{col}$ and $\boldsymbol{M}_{row}$ are $(n-1)$-dimensional column vectors, and p and w are scalars. Again, we know that:

$$-\overline{\boldsymbol{U}\boldsymbol{N}} = \overline{\boldsymbol{M}} + \boldsymbol{U}_{col}\boldsymbol{N}'_{row}.$$

Since $\boldsymbol{E}\overline{\boldsymbol{U}} = \overline{\boldsymbol{B}}^{-1}$, the new $\boldsymbol{M}_{\bar{B}}$ matrix will be

$$\boldsymbol{M}_{\bar{B}} = \boldsymbol{E}(\overline{\boldsymbol{M}} + \boldsymbol{U}_{col}\boldsymbol{N}'_{row}). \tag{14}$$

There are several cases that need to be checked before executing the above procedures. Most critically, if $\overline{\boldsymbol{B}}$ is singular, then $\boldsymbol{E}$ may be undefined. In such a case, we start Lemke's method from scratch with the initial basis, $\overline{\boldsymbol{B}}$, equal to $\boldsymbol{I}_{n-1}$. Clearly, this is not the only solution to this problem, but the scenario occurred rarely enough in practice so that this method was adequate for our purposes. Also, we assumed that we deleted the first row and n^{th} column from $\boldsymbol{B}$, $\boldsymbol{B}^{-1}$, $\boldsymbol{N}$ and $\boldsymbol{M}_B$. The general case can be easily modified to this special case.

To update $\boldsymbol{q}_B$, we delete the s^{th} element of $\boldsymbol{c}$, giving us $\overline{\boldsymbol{c}}$. Suppose $\boldsymbol{q}_B = \boldsymbol{B}^{-1}\boldsymbol{q}$ at the termination of Lemke's method, where $\boldsymbol{q} = \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{b} \end{bmatrix}$. Again, assuming that $s = 1$, we have:

$$\boldsymbol{q}_B = \begin{bmatrix} \tilde{q}_s \\ \tilde{\boldsymbol{q}}_B \end{bmatrix} = \boldsymbol{B}^{-1}\boldsymbol{q} = \begin{bmatrix} u & \boldsymbol{U}'_{row} \\ \boldsymbol{U}_{col} & \overline{\boldsymbol{U}} \end{bmatrix} \begin{bmatrix} q_s \\ \overline{\boldsymbol{q}} \end{bmatrix},$$

where $\tilde{\boldsymbol{q}}_B$ and $\overline{\boldsymbol{q}}$ is the $(n-1)$ lower subvector of $\boldsymbol{q}_B$ and $\boldsymbol{q}$, respectively, and $q_s = c_s$. Similarly to $\boldsymbol{M}_{\bar{B}}$, we get:

$$\boldsymbol{q}_{\bar{B}} = \boldsymbol{E}(\tilde{\boldsymbol{q}}_B - q_s \boldsymbol{U}_{col}). \tag{15}$$

LU Decomposition of the Basis

From Equations (14) and (15), we only require to know one column of $\boldsymbol{B}^{-1}$ to update $\boldsymbol{M}$ and $\boldsymbol{q}$. Thus, instead of explicitly maintaining $\boldsymbol{B}^{-1}$, we calculate the LU decomposition of the basis $\boldsymbol{B}$ at the termination of Lemke's method, and use it only to derive the required column of $\boldsymbol{B}^{-1}$.

We use Crout's algorithm to construct the LU decomposition of $\boldsymbol{B}$, and derive the s^{th} column of $\boldsymbol{B}^{-1}$ using backsubstitution. If $\boldsymbol{x}_s$ is the i^{th} basic variable, then we get $\boldsymbol{U}_{col}$ by deleting the i^{th} element of the column. Given $\boldsymbol{B}$, $\boldsymbol{N}$ and $\boldsymbol{U}_{col}$, we can update $\boldsymbol{M}$ and $\boldsymbol{q}$ according to Equations (14) and (15), respectively.

2.3 Branching Up

When $\alpha_i > 0$ in Problem (1), we need to ensure that $x_s \geq \alpha_s$ in the subsequent subproblems when branching up on x_s. We present two potential methods to maintain this condition: one using reformulation and the other by modifying Lemke's method to deal with nonzero lower-bounds.

Branch Up by Reformulation

We avoid explicitly adding the constraint $x_s \geq \alpha_s$ by replacing $\boldsymbol{x}$ by $\boldsymbol{y}$, where $y_i = x_i$, for $i \neq s$ and $y_s = x_s - \alpha_s$. The $\boldsymbol{A}$ matrix stays the same, but the right-hand-side vector $\boldsymbol{b}$ and the cost vector $\boldsymbol{c}$ need to be updated accordingly.

The original constraints were:

$$\sum_{j \neq s} \boldsymbol{A}_{.j} x_j + \boldsymbol{A}_{.s} x_s \leq \boldsymbol{b}$$

where $\boldsymbol{A}_{.j}$ is the j^{th} column of matrix $\boldsymbol{A}$. Replacing x_s with $y_s + \alpha_s$ gives us

$$\sum_{j \neq s} \boldsymbol{A}_{.j} y_j + \boldsymbol{A}_{.s} y_s \leq \boldsymbol{b} - \alpha_s \boldsymbol{A}_{.s}.$$

Thus, the new right hand side would be $\tilde{\boldsymbol{b}} = \boldsymbol{b} - \alpha_s \boldsymbol{A}_{.s}$.

The objective function is also modified to

$$\frac{1}{2}(\boldsymbol{y} + \alpha_s \boldsymbol{e}_s)' \boldsymbol{Q} (\boldsymbol{y} + \alpha_s \boldsymbol{e}_s) + \boldsymbol{c}'(\boldsymbol{y} + \alpha_s \boldsymbol{e}_s)$$

which simplifies to

$$\frac{1}{2}\boldsymbol{y}'\boldsymbol{Q}\boldsymbol{y} + \tilde{\boldsymbol{c}}'\boldsymbol{y} + C_0$$

where $\tilde{\boldsymbol{c}} = \boldsymbol{c} + \alpha_s \boldsymbol{Q}_{.s}$ and $C_0 = \frac{1}{2}\alpha_s^2 Q_{s,s} + \alpha_s c_s$ is a constant. The reformulated problem is thus:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{y}'\boldsymbol{Q}\boldsymbol{y} + \tilde{\boldsymbol{c}}'\boldsymbol{y} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{y} \leq \tilde{\boldsymbol{b}}, \\ & \boldsymbol{y} \geq 0, \end{aligned} \tag{16}$$

which is in the same format as Problem (2). If $\boldsymbol{B}$ is the basis corresponding to the solution of the parent node, then $\boldsymbol{q}_B$ is modified to

$$\tilde{\boldsymbol{q}}_B = \boldsymbol{B}^{-1} \begin{bmatrix} \tilde{\boldsymbol{c}} \\ \tilde{\boldsymbol{b}} \end{bmatrix}$$

If we have the LU decomposition of the basis instead of the entire inverse matrix, then $\tilde{\boldsymbol{q}}_B$ can similarly be calculated by backsubstitution of the vector $\begin{bmatrix} \tilde{\boldsymbol{c}} \\ \tilde{\boldsymbol{b}} \end{bmatrix}$. Note that $\boldsymbol{M}$ need not be updated.

Lemke's Method with Lower Bounds

Another approach to enforce the lower bound, $x_s \geq \alpha_s$, without adding the constraint is to modify the Lemke's method to deal with nonzero lower bounds. Just like in simplex method with lower and upper bounds, the pivoting step can easily be altered if some basic variables have nonzero lower bounds. In a pivot step, we want to increase the value of the driving variable from zero to some positive number, while ensuring that all of the variables remain larger than its lower bound. The basic variable that hits its lower bound first is the blocking variable and would be pivoted out of the basis. Thus, the only modification required is in determining the blocking variable. When branching up on x_s, we need to update the lower bound of that variable for all subsequent subproblems so that the Lemke's method would prevent x_s from being less than α_s when pivoting in a variable. At the termination of Lemke's method, if a branched-up variable is nonbasic, it's value would equal its lower bound.

Using this modified Lemke's method, we do not need to update the vector $\boldsymbol{q}$ as in the previous section. However, when branching down on subsequent nodes, we need to check whether currently nonbasic variable has been branched up. If U is the set of variables that are branched up and NBV are the set of variables that are currently nonbasic, then when updating the $\boldsymbol{q}$ vector as in Equation (15), q_s will equal $c_s + \sum_{i \in U \cap NBV} \alpha_i Q_{s,i}$, since $x_i = \alpha_i$ for $i \in U \cap NBV$, and thus the data need to be updated accordingly.

2.4 Additional Algorithmic Ideas within Branch and Bound

To reduce the total number of nodes in the branch-and-bound tree, we introduce heuristics to find strong upper bounds and stop branching after branching up on K variables.

Heuristic for Upper Bounds

We use the heuristic similar to those in [5] and [16] at the root. Let $\boldsymbol{x}^*$ be the solution of the continuous relaxation at the root, and $G = \{i | x_i$ has one of $K + W$ largest absolute value out of all d variables $\}$, where W is a user-defined small positive integer such that $|G| = K + W \ll d$. We then set all $x_i = 0$ where $i \in \{1, \ldots, d\} \setminus G$ and branch exclusively on variables x_j where $j \in G$, and also limit the number of nodes to examine.

Solving Subproblem after Branching up K Variables

After branching up on K variables, the remaining variables must be zero in order to be feasible for Problem (1). However, conventional branch-and-bound procedures would continue branching down the tree, until all of the variables are branched. We can halt all branching once we branch up K variables, then solve Problem (1) only on the K branched up variables. This saves considerable computation time, especially if there are many variables yet to be branched.

3 Applications of Quadratic Variable Selection Problems

We focus on applying the methodology described in Section 2 to the K-subset selection problem in regression and optimal portfolio selection in Section 3.1 and 3.2, respectively.

3.1 Subset Selection in Regression

Suppose we are given m data points $(\boldsymbol{x}_i, y_i)$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, and we want to choose K variables $(K < d)$ that minimizes the total sum of squared errors. We formulate this problem as:

$$\begin{aligned} \text{minimize} \quad & (\boldsymbol{Y} - \boldsymbol{X\beta})'(\boldsymbol{Y} - \boldsymbol{X\beta}) \\ \text{subject to} \quad & |supp(\boldsymbol{\beta})| \leq K. \end{aligned} \tag{17}$$

When the cardinality constraint is relaxed, the optimal objective value is $\boldsymbol{Y'Y} - \boldsymbol{Y'X'}(\boldsymbol{X'X})^{-1}\boldsymbol{X'Y}$, thus we clearly do not need to run the Lemke's method. The main computational work is in the branch down procedure. There are, however, regression problems that have linear constraints with respect to the coefficients. We would use the general methodology in such cases, but let us focus on the unconstrained regression for now.

When branching down on x_s, we delete the s^{th} row and column of $\boldsymbol{X'X}$, and the inverse $(\boldsymbol{X'X})^{-1}$ is updated as illustrated in Section 2.2. To further alleviate computation, we set $\boldsymbol{v} = \boldsymbol{X'Y} \in \mathbb{R}^d$ at the root node. Thus, deleting x_s corresponds to deleting the s^{th} element of $\boldsymbol{v}$. Thus, there is no need to multiply $\boldsymbol{X}'$ and $\boldsymbol{Y}$ in subsequent nodes – we simply need to delete corresponding elements from $\boldsymbol{v}$. The optimal objective value of a given node is then:

$$\boldsymbol{Y'Y} - \overline{\boldsymbol{v}}'(\overline{\boldsymbol{X'X}})^{-1}\overline{\boldsymbol{v}}.$$

where $\overline{\boldsymbol{X'X}}$ and $\overline{\boldsymbol{v}}$ are the updated $\boldsymbol{X'X}$ and $\boldsymbol{v}$, respectively. Thus, to calculating the objective requires only matrix-vector multiplications. There is no need to update the subproblem when branching up, since the optimal solution of the parent node is optimal for the next node. Section 4.1 illustrates computational results of our approach.

3.2 Portfolio Selection

Investors often aim to construct a portfolio of at most K stocks that matches a benchmark portfolio comprised of d stocks $(K < d)$ as closely as possible while minimizing transaction costs and limiting the total investment change within each of S industry sectors.

Suppose the current portfolio weights are $\boldsymbol{x}^0 \in \mathbb{R}^d$. The traditional mean-variance model determines the new weights for the d stocks, $\boldsymbol{x} \in \mathbb{R}^d$, that maximizes the total expected return minus a penalty times total variance. In practice, there are other direct and indirect transaction costs, such as price impact costs and ticket costs. Impact costs reflect the stock price impact resulting from purchase or sale orders, and the magnitude of this cost is dependent of the particular stock and the trade sizes. For example, large purchase orders will increase the price and large sales orders will decrease the price of the stock. Assuming symmetric impact for purchases and sales, this effect is often modelled by the following quadratic function

$$\sum_{i=1}^{d} c_i(x_i - x_i^0)^2,$$

where c_i, $c_i > 0$, is the impact coefficient for Stock i. The second form of transaction costs is ticket cost, which is a fixed cost associated to trading a positive volume of a stock. This cost can easily be incorporated into our quadratic variable selection problem framework using binary variables, but we decided not to include it in the present work becasue often ticket costs are second order compared to impact costs, and also in order to focus on the key aspect of the present work of cardinality constrained quadratic programs.

The portfolio selection problem with limited diversification, minimum transaction levels and price impact costs can be represented by the following formulation:

$$\begin{array}{lll} \text{minimize} & -\boldsymbol{r}'\boldsymbol{x} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^B)'\boldsymbol{\Sigma}(\boldsymbol{x} - \boldsymbol{x}^B) + \sum_{i=1}^{d} c_i(x_i - x_i^0)^2, & \\ \text{subject to} & |\sum_{i \in S_l}(x_i - x_i^B)| \leq \epsilon_l, & l = 1, ..., S, \\ & \sum_{i=1}^{d} x_i = 1, & \\ & |supp(\boldsymbol{x})| \leq K, & \\ & x_i \geq \alpha_i, & i \in supp(\boldsymbol{x}), \\ & x_i \geq 0, & i = 1, \ldots, d, \end{array} \tag{18}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the rates of return, x_i^B is the benchmark weight for stock i, $\boldsymbol{r}$ is a $d-$dimensional vector of the expected rates of return, α_i is the minimum transaction level of stock i, c_i is the price impact coefficient for Stock i, and S_l is the set of indices of stocks in Sector l. The first constraint limits the total change in the portfolio weights in Sector l to be less than some ϵ_l. The second set of constraints ensures that the weights sum up to 1, and the third constraint limits investing up to K stocks. The fourth set of constraints implies that if we invest in stock i, then x_i must be at least α_i. Clearly, Problem (18) is in the form of Problem (1) and can be solved using our methodology.

We rewrite Problem (18) as

$$
\begin{aligned}
\text{minimize} \quad & -\tilde{\boldsymbol{r}}'\boldsymbol{x} + \tfrac{1}{2}\boldsymbol{x}'\tilde{\boldsymbol{\Sigma}}\boldsymbol{x} + C_0, && && (19)\\
\text{subject to} \quad & \sum_{i \in S_l} x_i \le \epsilon_l + \sum_{i \in S_l} x_i^B, && l = 1, \ldots, S,\\
& -\sum_{i \in S_l} x_i \le \epsilon_l - \sum_{i \in S_l} x_i^B, && l = 1, \ldots, S,\\
& \sum_{i=1}^{d} x_i = 1,\\
& |supp(\boldsymbol{x})| \le K,\\
& x_i \ge \alpha_i, && i \in supp(\boldsymbol{x}),\\
& x_i \ge 0, && i = 1, \ldots, d,
\end{aligned}
$$

where $\tilde{\boldsymbol{r}} = \boldsymbol{r} + \boldsymbol{\Sigma}\boldsymbol{x}^B + 2\boldsymbol{C}\boldsymbol{x}^0$, $\tilde{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma} + 2\boldsymbol{C}$ where $\boldsymbol{C}$ is the diagonal matrix with c_i as the i^{th} diagonal element, and C_0 is a constant equal to $\boldsymbol{x}^{B\,\prime}\boldsymbol{\Sigma}\boldsymbol{x}^B + \sum_{i=1}^{d} c_i (x_i^0)^2$.

We solve the relaxation of Problem (19) using Lemke's method described in Section 2.1, and branch down and up on a variable as in Section 2.2 and 2.3, respectively. Section 4.2 illustrates the computational results of this method.

Heuristic Tailored to Portfolio Selection

When K is relatively large (i.e., $K > 100$), solving the root heuristic of Section 2.4 with even $W = 0$ can be time consuming. Since the goal of this heuristic is to find a good feasible solution quickly, we may limit $|G|$ to be smaller than K – perhaps to 10% of the total number of variables. Unlike the subset selection problem in regression, where solutions to most subproblems have no nonzero values, the nonnegativity constraints make the solution to the portfolio problem have substantial number of variables equal to zero. Thus, the cardinality constraint in the optimal solution may not necessary be tight. Also, although the set of variables in the optimal solution may not coincide with our choice of G, we see that diminishing the size of our root problem gave us good feasible solutions within a short amount of time, as we illustrate in Section 4.2.

4 Computational Results

We describe computational experiments on subset selection and portfolio selection problems in Section 4.1 and 4.2, respectively. Each section gives the performance results of our methods compared to alternative approaches, and describe the advantages and weaknesses of our method.

4.1 Results for Subset Selection

We compared our implicit branch-and-bound method for solving subset selection with forward regression and an explicit branch-and-bound model solved by CPLEX 8.0 [9]. Forward regression is a greedy heuristic that, given the variables already chosen, chooses another variable that reduces the residual error the most, i.e., the first variable chosen, $x_{(1)}$, minimizes $\sum_i (y_i - x_{i,j}\beta_j)^2$ for all $j = 1, \ldots, d$. The next variable minimizes $\sum_i (\bar{y}_i - x_{i,j}\beta_j)^2$ for all $j \in \{1, \ldots, d\} \setminus (1)$, where $\bar{y}_i = y_i - x_{i,(1)}\beta_{(1)}$ and $\beta_{(1)}$ is the regression coefficient for $x_{(1)}$ found in the previous step. This step is repeated until K variables are chosen [22]. We used CPLEX 8.0's quadratic mixed-integer optimizer to solve Problem (17) by introducing binary inclusion variables z_i, replacing the cardinality constraint by $\sum_i z_i \leq K$ and adding constraints $\beta_i \geq -Mz_i$ and $\beta_i \leq Mz_i$, where M is some large positive number. We found that setting $M = 100$ was sufficiently large to solve our generated problems effectively. CPLEX 8.0 uses a pivoting method similar to Lemke's method to solve the relaxation problem at each node. It does not, however, delete variables nor reformulate the problem during branching.

For each d (the number of total variables) and K (the size of the desired subset), we randomly generated instances of $\boldsymbol{X}$ and $\boldsymbol{\beta}$, and set $\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon}$, where $\epsilon_i \sim N(0,1)$ for each i. For each problem size, we generate five such instances and averaged the performances of the methods over all of them. We also set the time limit to 60 CPU seconds. Our implicit branch-and-bound and CPLEX's branch-and-bound search procedure were set depth first search, and branches on the variable with maximum absolute value first. Table 1 illustrates the performance results of our model, forward regression and CPLEX.

The columns "Forward", "ImplicitBnB", and "CplexMIQP" corresponds to the results of the forward regression, our method, and CPLEX, respectively. We did not record the running time for forward regression, since the simple heuristic was able to solve all the instances in a fraction of a second. The column labelled "CPU sec" is the total CPU seconds required to solve the problem up to 60 CPU seconds. "ImplicitBnB" solved all the instances to optimality except for $d = 100$. "CplexMIQP" hit the 60 CPU limit before getting a provable optimal solution for all instances with $d = 100$, and four instances with $d = 50$, $K = 20$. The column labelled "nodes" is the total number of nodes in the branch-and-bound tree at termination, and "RSS" is the best total sum of squared errors for subsets of size K found. Both the number of nodes and RSS were rounded to the nearest integer.

		Forward	ImplicitBnB			CplexMIQP		
d	K	RSS	CPU sec	# nodes	RSS	CPU sec	# nodes	RSS
20	10	4,464	0.016	263	2,176	0.096	197	2,176
20	5	9,819	0.010	224	6,765	0.118	227	6,765
50	40	47,755	0.914	1,023	2,156	19.402	15,679	2,156
50	20	125,520	34.226	98,365	55,753	54.204	50,037	60,597
100	80	383,666	60.000	18,623	16,896	60.000	16,723	692,896
100	50	386,922	60.000	23,291	115,388	60.000	19,222	528,967
100	20	696,772	60.000	68,000	541,950	60.000	18,825	570,801

Table 1: Results for Subset Selection with 60 CPU seconds. d is the number of variables, K is the size of the selected subset, and RSS is the residual sum of squares.

It is apparent from Table 1 that the optimization approaches significantly improve upon the forward regression in terms of residual sum of squares, even when they do not solve to provable optimality. "ImplicitBnB" also finds strong feasible solutions significantly faster than the explicit branch-and-bound formulation solved by CPLEX, mainly because the former takes great advantage of the special structure of Problem (17), which does not require introducing binary variables. Even for several hundred variables, the node-to-node computation time in our model takes a fraction of a CPU second. Also, the heuristic in Section 2.4 often finds the optimal objective value at the root.

We reach similar conclusions when using longer running times. Table 2 illustrates results when our method and CPLEX are run for 3600 CPU seconds (note: these were run on similar but different data sets than those of Table 1). The column labelled "best node" refers to the node that found the best feasible solution. Although CPLEX was now able to find the optimal solution for $d = 50$ (though not always to provable optimality) within 3600 CPU seconds, the solutions are found significantly later than by our method.

The main bottleneck, however, is the number of nodes needed to prove optimality. Even when the heuristic finds the optimal solution at the root, the branch-and-bound tree can grow to several million nodes to prove optimality, even for moderate sized problems (e.g. $d = 50$). The main factors preventing significant pruning of the tree are the free variables and the lack of constraints. A subproblem solution almost always has all non-zero variables.

		Forward	ImplicitBnB			CplexMIQP		
d	K	RSS	# nodes	best node	RSS	#nodes	best node	RSS
50	40	24,116	183	0	1,075	55,399	55,358	1,075
50	20	111,735	710,243	17,657	56,708	915,159	892,052	56,708
100	80	326,914	257,003	74,186	14,234	992,506	987,366	365,582
100	50	555,370	779,800	2,840	128,936	1,086,256	0	737,592
100	20	862,900	1,959,170	8,292	624,438	1,006,151	0	657,418

Table 2: Results for Subset Selection with 3600 CPU seconds. d is the number of variables, K is the size of the selected subset, and RSS is the residual sum of squares.

4.2 Results for Portfolio Selection

We tested our approaches described in Section 3.2 against two alternative methods. One method uses CPLEX's quadratic barrier method to solve the relaxation problem of (19), and the second uses CPLEX's quadratic mixed-integer solver to solve the explicit integer formulation. We also tested reformulation versus using Lemke's method with lowerbounds when branching up. Again, for each d (number of total assets), S (number of sectors), and K (the upper bound on diversification), we generated five random instance of Problem (18) and averaged the results. We set $\boldsymbol{x}^0 = \mathbf{0}$ for all of our instances. All branch-and-bound procedures were set to depth-first-search, branch up first and branch on variable with maximum absolute value. Tables 3 and 4 illustrate the results.

"LemkeRef" is the combination of Lemke's method and implicit branch-and-bound using reformulation when branching up. "LemkeLB" is the same, but uses Lemke's method with lowerbound instead of reformulation. "Barrier" is also implemented with implicit branch-and-bound, but uses CPLEX's barrier method to solve the continuous quadratic optimization problem. Finally, "CplexMIQP" is the result of using CPLEX quadratic mixed-integer solver. All four methods were run for a total of 120 CPU seconds. The column labelled "nodes" is the average of the total number of nodes each method explored, "best node" is the average of the node where the best feasible solutions were found, and "UB" is the best feasible objective value found within the time limit.

Table 3 runs all four methods without running any heuristic methods to find a good feasible solution, whereas Table 4 runs such a heuristic once at the root. For "LemkeRef", "LemkeLB" and "Barrier", we ran the heuristic described in Section 2.4 for at most 30 CPU seconds after the root is solved. For every instance, we ran the heuristic using $K = 0.1d$ and $|G| = K + 10$. We were not able to put a time limit on the heuristic for CplexMIQP, which ran until CPLEX deemed it had an acceptable upperbound. When the size of K was relatively large ($K > 0.1d$), the CPLEX heuristic ran for at most 10 CPU seconds, whereas it can last over 100 CPU seconds when d is large ($d > 200$) and K is small compared to d (e.g., $K \leq 0.1d$).

The node to node runtime of "LemkeRef" and "LemkeLB" were statistically the same, thus avoiding reformulation did not seem to gain significant running time. Since the computational characteristics of both methods were very similar, we will refer to both methods as "Lemke" from here on end. Both pivoting methods, "Lemke" and "CplexMIQP", are significantly faster than "Barrier" for every instance. Although the relative difference in the total number of nodes explored did decrease as the problem size increased, the advantage of interior point methods in large dimensions over pivoting methods did not compensate the latter's advantage in warm starting. For example, for problems that would take an average of 400 pivots to solve from scratch, any intermediary node would require only about 5 pivots to resolve the subproblem of that node. Also, the pivoting methods always give a basic feasible solution to the KKT equations of the quadratic programming problem. Thus, it is guaranteed to give the solution to the relaxation with the minimum support, unlike the interior point method. Since many of the instances generated and most real world problems do not guarantee positive definiteness (only positive semi-definiteness) in the quadratic matrix, this difference can be another significant advantage.

It is clear that the node to node computation time of "CplexMIQP" is faster than "Lemke" for most instances. However, the relative difference significantly decreases as K becomes small relative to d. For example, when $K \approx 0.5d$, the difference in number of nodes explored is about a factor of 10, whereas when $K \approx 0.1d$, it reduces to a factor of 2 to 3. For the case when $d = 200$ and $K = 20$, the total computation time of "Lemke" and "CplexMIQP" are about the same. We believe that the gap gets closer when K is small, since "Lemke" takes advantage of the special structure of Problem (18) more than the explicit formulation can, which becomes increasingly important and thus advantageous as K becomes smaller. We reach similar conclusions when we significantly increase the running time. Table 5 shows results for running our method with Lemke's with lower bounds and CPLEX for 3600 CPU seconds.

Running the heuristic, even for just 30 CPU seconds, brought significant improvement to our models in terms of finding good feasible solutions. Again, we tailored our heuristic to the special structure of the problem. Using $|G|$ small enough so that Lemke's method can run sufficiently fast allowed us to find a good feasible solution quickly.

			LemkeRef			LemkeLB		
d	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	17486.20	16227.80	28.26	16992.20	16039.60	28.46
100	10	4	26937.20	8345.40	18.83	27231.40	8329.40	18.83
200	100	10	2936.60	2924.80	142.80	2952.80	2933.80	142.73
200	20	4	6924.80	1281.00	38.73	6913.20	1281.00	38.73
500	200	10	142.20	-	-	142.40	-	-
500	100	10	161.60	126.20	159.71	164.40	137.80	159.68
500	50	4	63.80	44.00	91.06	64.20	46.40	90.89

			Barrier			CplexMIQP		
d	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	4526.80	4472.00	44.51	119697.00	76987.00	19.02
100	10	4	5686.00	1678.20	19.57	58530.20	12736.60	18.49
200	100	10	751.20	729.40	150.57	30188.60	30062.80	81.86
200	20	4	1284.00	656.20	39.77	7236.80	1245.60	38.73
500	200	10	30.80	-	-	4864.00	4812.60	345.46
500	100	10	33.20	-	-	782.60	398.00	158.68
500	50	4	35.40	-	-	226.40	140.40	90.89

Table 3: Results for portfolio selection, without Heuristic, solved until 120 CPU seconds. d is the number of variables, K is the size of the selected subset, S is the number of sectors (Problem (18)), "UB" is the best feasible solution found, "best node" is the node where "UB" was found and "nodes" is the total number of nodes explored. Entries labelled "-" indicate that the method failed to find a feasible solution within 120 CPU seconds.

			LemkeRef			LemkeLB		
d	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	14235.80	0.00	12.93	13998.40	0.00	12.93
100	10	4	23926.40	0.00	14.87	23858.00	0.00	14.87
200	100	10	2501.00	0.00	32.90	2490.60	0.00	32.90
200	20	4	6248.80	58.60	34.87	6232.80	58.60	34.87
500	200	10	117.80	0.00	83.93	119.40	0.00	83.93
500	100	10	132.80	0.00	80.40	134.00	0.00	80.40
500	50	4	47.20	0.00	81.53	47.00	0.00	81.53

			Barrier			CplexMIQP		
d	K	S	nodes	best node	UB	nodes	best node	UB
100	50	10	4029.40	2678.20	36.36	116468.20	70551.00	18.68
100	10	4	4166.40	0.00	15.66	58410.40	12736.60	18.49
200	100	10	485.40	0.00	34.51	18533.20	7800.80	52.84
200	20	4	916.40	58.60	35.58	6937.00	1245.60	38.73
500	200	10	24.60	-	-	1098.00	0.00	138.83
500	100	10	27.00	-	-	1097.40	0.00	140.29
500	50	4	27.40	-	-	146.40	103.20	91.14

Table 4: Results for portfolio selection, with the root Heuristic, solved until 120 CPU seconds. d is the number of variables, K is the size of the selected subset, S is the number of sectors (Problem (18)), "UB" is the best feasible solution found, "best node" is the node where "UB" was found and "nodes" is the total number of nodes explored. Entries labelled "-" indicate that the method failed to find a feasible solution within 120 CPU seconds.

			Lemke			CplexMIQP		
d	K	S	nodes	best node	UB	nodes	best node	UB
500	200	10	6,100	0	83.93	49,720	0	138.83
500	100	10	5,960	0	80.40	55,183	0	140.29
500	50	4	13,828	0	80.01	15,078	7,995	89.34

Table 5: Results for portfolio selection, with the root Heuristic, solved for 3600 CPU seconds.

5 Conclusion

From this computational study, we learn that:

1. Our tailored approaches for solving quadratic variable selection problems in regression and portfolio optimization show computational advantages over general mixed-integer optimization formulation and significantly better solution quality than simple heuristic methods.

2. For subset selection in regression, the optimization based approaches were able to find the subset of variables with significantly better fit than the forward regression heuristic. Also, since we took advantage of the special structure of the problem, our implicit branch-and-bound formulation had faster node-to-node running times than CPLEX's quadratic mixed-integer solver.

3. For the portfolio selection problem, the combination of implicit branch-and-bound and Lemke's method has significantly faster running times compared to using the barrier method to solve the continuous quadratic optimization problem. The key bottleneck for efficient quadratic mixed-integer optimization has been the inability of interior point methods to start at infeasible points. Although they are undoubtedly more effective in solving high dimensional quadratic optimization problems than pivoting methods started from scratch, the pivoting methods can re-solve each subproblem more efficiently at each node of the branch-and-bound tree.

4. CPLEX's quadratic mixed-integer solver has a more sophisticated pivoting and branch-and-bound implementation, yet our tailored approach compensates for our lack of software engineering prowess. Our root heuristic finds good upper bounds quickly, and our variable deletion, reformulation and modified Lemke's method updates each subproblem without increasing the size of the problem. With further improvements in implementation (e.g., regarding data structures, decompositions, and memory handling), we believe our methodology will have comparable node-to-node

running times.

There are several potential improvements to our model. We use depth-first-search in all of our branch-and-bound procedures due to the ease of implementation. Although we can find good upper bounds faster with this approach, we often get stuck in a subtree. Also, with large number of nodes, we cannot utilize best lower bounds effectively, since the root relaxation would be the lower bound for a large majority of the nodes we explore. There is also merit in investigating other principal pivoting techniques other than Lemke's. The main drawback of Lemke's method is the lack of choice in the driving variable. Alternative pivoting methods, though more difficult to start, have the flexibility of choosing amongst several driving variables, akin to the simplex method. It also does not augment the LCP, thus not introducing an auxiliary variable and column. These properties may allow us to converge faster to the solution.

References

[1] Arthanari, T.S.; Dodge, Y. (1993). *Mathematical Programming in Statistics*, Wiley, Wiley Classics Library Edition, New York.

[2] Beale, E.M.L.; Kendall, M.G.; Mann, D.W. (1967). "The Discarding of Variables in Multivariate Analysis", *Biometrika*, Vol. 54, Issue 3/4, 357-366.

[3] Bertsimas, D.; Darnell, C.; and Soucy, R. (1999). "Portfolio construction through mixed-integer programming at Grantham, Mayo, Van Otterloo and Company," *Interfaces*, Vol. 29, No. 1 (January-February), 49-66.

[4] Bertsimas, D.;Gamarnik, D.;Tsitsiklis, J. (1999). "Estimation of Time-Varying Parameters in Statistical Models: An Optimization Approach", *Machine Learning*, vol. 35, 225-245.

[5] Bienstock, D. (1996). "Computational Study on Families of Mixed-Integer Quadratic Programming Problems", *Mathematical Programming*, Vol. 74, 121-140.

[6] Blog, B.; van der Hoek, G.. Rinnooy Kan, A.H.G.;Timmer, G.T. (1983). "Optimal Selection of Small Portfolio", *Management Science*, Vol. 29, No. 7, 792-798.

[7] Chang, T.J.; Meade, N.; Beasley, J.E.; Sharaiha, Y.; (2000). "Heuristics for Cardinality Constrained Portfolio Optimisation", *Computers and Operations Research*, Vol. 27, 1271-1302.

[8] Cottle, R. W.; Pang, J.; Stone, R. E., (1992). *The Linear Complementarity Problem,* Academic Press, Inc.

[9] ILOG CPLEX 8.1 User Manual, (2002). ILOG CPLEX Division, Incline Village, NV.

[10] Eaves, B.C. (1971). "The Linear Complementarity Problem", *Management Science*, Vol. 17., No. 9, 612-634.

[11] Faaland, B. (1974). "An Integer Programming Algorithm for Portfolio Selection", *Management Science*, Vol. 20, No. 10, 1376-1384.

[12] Furnival, G.M.; Wilson Jr., R.W. (1974). "Regression by Leaps and Bounds", *Technometrics*, Vol. 16, No. 4, 499-511.

[13] Geoffrion, A.M. (1967). "Integer Programming by Implicit Enumeration and Balas' Method," *SIAM Review*, Vol. 9., 178-190.

[14] Hockings, R.R.; Leslie, R.N. (1967). "Selection of the Best Subset in Regression Analysis", *Technometrics*, Vol. 9, No. 4, 531 - 540.

[15] Jacob, N. (1974). "A Limited Diversification Portfolio Selection Model for the Small Investor", *The Journal of Finance*, Vol. 29, Issue 3, 847-856.

[16] Jobst, N.; Horniman, M.; Lucas, C.; Mitra, G. (2001). "Computational Aspects of Alternative Portfolio Selection Models in the Presense of Discrete Asset Choice Constraints", *Quantitative Finance*, Vol. 1 , No. 5, 489-501.

[17] LaMotte, L.R.; Hockings, R.R. (1970). "Computational Efficiency in the Selection of Regression Variables", *Technometrics*,Vol. 12, No. 1 , 83 - 93.

[18] Lemke, C.E. (1965). "Bimatrix Equilibrium Points and Mathematical Programming", *Management Science*, Vol.11, No. 7, 681-689.

[19] Lemke, C.E.; Howson Jr., J.T. (1964). "Equilibrium Points of Bimatrix Games", *Journal of the Society for Industrial and Applied Mathematics*, Vol. 12, No. 2, 413-423.

[20] Mansini, R.; Speranza, M.G. (1999). "Heuristic Algorithms for Portfolio Selection Problem with Minimum transaction Lots", *European Journal of Operations Research*, Vol. 114, Issue 1, 219-233.

[21] McBride, R. D.; Yormark, J.S. (1980). "An Implicit Enumeration Algorithm for Quadratic Integer Programming", *Management Science*, Vol. 26, No. 3, 282-296.

[22] Miller, A., (1990). *Subset Selection in Regression*, Monographs on Statistics and Applied Probability 40, Chapman and Hall.

[23] Narula, S.; Wellington, J. (1979). "Selection of Variables in Linear Regression Using the Minimum Sum of Weighted Absolute Errors Criterion",*Technometrics*,Vol. 21, No. 3, 299-311.

[24] Owens-Butera, G. (1997). " The Solution of a Class of Limited Diversification Portfolio Selection Problems", Rice University Ph.D. Thesis, CRPC-TR97724-S.

[25] Patel, N.; Subrahmanyam, M. (1982). "A Simple Algorithm for Optimal Portfolio Selection with Fixed Transaction Costs", *Management Science*, Vol. 28, No. 3, 303-314.

[26] Ryan, T.P. (1997). *Modern Regression Methods*,Wiley Series in Probabilty and Statistics, New York.

[27] Sharpe, W. (1967). "A Linear Programming Algorithm for Mutual Fund Portfolio Selection",*Management Science*, Vol. 13, No. 7, 499-510.

[28] Sharpe, W. (1971) "A Linear Programming Approximation for the General Portfolio Analysis Problem", *Journal of Financial and Quantitative Analysis*, Vol. 6, Issue 5, 1263-1275.