Comput Optim Appl
DOI 10.1007/s10589-007-9126-9

Algorithm for cardinality-constrained quadratic optimization

Dimitris Bertsimas · Romy Shioda

Received: 14 June 2006 / Revised: 25 May 2007

Abstract This paper describes an algorithm for cardinality-constrained quadratic optimization problems, which are convex quadratic programming problems with a limit on the number of non-zeros in the optimal solution. In particular, we consider problems of subset selection in regression and portfolio selection in asset management and propose branch-and-bound based algorithms that take advantage of the special structure of these problems. We compare our tailored methods against CPLEX's quadratic mixed-integer solver and conclude that the proposed algorithms have practical advantages for the special class of problems we consider.

Keywords Mixed-integer quadratic programming · Branch-and-bound · Lemke's method · Subset selection · Portfolio selection

1 Introduction

We present a method for solving cardinality-constrained quadratic optimization problems (CCQO), i.e., quadratic optimization problems that limit the number of non-zero

The research of D. Bertsimas was partially supported by the Singapore-MIT alliance.
The research of R. Shioda 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.

D. Bertsimas
Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, E53-363, Cambridge, MA 02139, USA
e-mail: dbertsim@mit.edu

R. Shioda (✉)
Department of Combinatorics and Optimization, Faculty of Mathematics, University of Waterloo, Waterloo, ON N2L 3G1, Canada
e-mail: rshioda@math.uwaterloo.ca

variables in the solution, using a tailored branch-and-bound implementation with pivoting algorithms. Specifically, we consider the following problem:

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

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 $\operatorname{supp}(\boldsymbol{x}) = \{i | x_i \neq 0\}$. 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), [4] proposes a tailored branch-and-bound algorithm and replaces the cardinality constraint $|\operatorname{supp}(\boldsymbol{x})| \leq K$ with a surrogate constraint $\sum_i (x_i/u_i) \leq K$. Moreover, the $\boldsymbol{x}$ variables are branched on directly instead of introducing binary variables, i.e., the constraint $x_j \leq 0$ is added when branching down on x_j and the constraint $x_j \geq \alpha_i$ is added when branching up on x_j. The underlying quadratic solver used in [4] is a primal feasible algorithm that searches for feasible descent directions, which includes Newton's direction, steepest descent direction and Frank-Wolfe's method. Warm-starting was done at each branch-and-bound node by using a quadratic penalty function.

Motivated by this work, we extend the algorithm of [4] by using Lemke's pivoting algorithm [7, 14] to solve the successive sub-problems in the branch-and-bound tree. Unlike [4], we do not explicitly add the variable bound constraints, $x_j \leq 0$ and $x_j \geq \alpha_i$, thus the size of the subproblems never increases. The other major distinction to [4] is that we use a pivoting algorithm to solve the subproblems, which allows for efficient warm-starting. Section 2 elaborates on this general methodology for solving CCQO's. In Sect. 3, we further tailor our method to solve two important problems in statistics and finance: subset selection in regression and portfolio selection in finance. We illustrate the results of our computational experiments in Sect. 4.

2 General methodology

In a branch-and-bound setting, we solve the convex 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 modify Lemke's method so that $x_s \geq \alpha_s$ is enforced during pivoting.

The relaxation we solve at each node is:

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x}, \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \\ & x_i \geq \alpha_i, \quad i \in U, \end{aligned} \tag{2}$$

where the cardinality constraint is removed and U is the set of indices of variables that have been branched up. The lower bound constraints $x_i \geq \alpha_i$ for α_i strictly positive are enforced by implementing Lemke's method with non-zero lower-bounds (analogous to the simplex method with lower and upper-bounds). 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. Section 2.3 describes a heuristic based on our branch-and-bound procedure for finding a good feasible solution.

2.1 Lemke's method as underlying quadratic optimizer

We use Lemke's pivoting method to optimize the convex relaxation of the subproblem at each branch-and-bound 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 basic solution. 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 approach has an advantage over interior point methods which may need to solve from scratch at each node.

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,

$$\boldsymbol{w} = \boldsymbol{M}\boldsymbol{z} + \boldsymbol{q}, \quad \boldsymbol{z} \geq \boldsymbol{0}, \quad \boldsymbol{w} \geq \boldsymbol{0}, \quad \boldsymbol{z}^\top \boldsymbol{w} = 0.$$

The above problem is referred to as LCP($\boldsymbol{q}, \boldsymbol{M}$). Clearly, the KKT necessary and sufficient optimality conditions of a convex quadratic programming problem is an LCP.

The Lemke's method first 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

$$\boldsymbol{w} = \boldsymbol{q} + \boldsymbol{h}z_0 + \boldsymbol{M}\boldsymbol{z} \geq \boldsymbol{0}, \quad z_0 \geq 0, \quad \boldsymbol{z} \geq \boldsymbol{0}, \quad \boldsymbol{z}^\top \boldsymbol{w} = 0, \tag{3}$$

where $\boldsymbol{h}$ is some user-defined *covering vector* with $\boldsymbol{h} > \boldsymbol{0}$. We need to start the algorithm with a complementary basis that does not necessarily satisfy the nonnegativity constraint. 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 (3). 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 [13].

During the branch-and-bound procedure, we want to resolve a new subproblem starting with the basic solution of the parent subproblem. Let $\overline{\boldsymbol{M}}$ and $\overline{\boldsymbol{q}}$ be the modified data for the current subproblem, and let $\overline{\boldsymbol{B}}$ and $\overline{\boldsymbol{N}}$ be the corresponding columns of the basic and nonbasic variables, respectively, of the parent node. We want Lemke's method to solve LCP($\boldsymbol{M}_{\bar{B}}, \boldsymbol{q}_{\bar{B}}$), where $\boldsymbol{M}_{\bar{B}} = -\overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{N}}$ and $\boldsymbol{q}_{\bar{B}} = \overline{\boldsymbol{B}}^{-1}\overline{\boldsymbol{q}}$, and have $\overline{\boldsymbol{B}}$ as its initial complementary basis matrix. This basis is most likely not feasible for LCP($\boldsymbol{M}_{\bar{B}}, \boldsymbol{q}_{\bar{B}}$), thus the problem is augmented by the auxiliary variable z_0, z_0 is increased until the initial basis is feasible for the augmented problem, and then we execute sequence of pivots until z_0 is pivoted out or the LCP is deemed infeasible.

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 via elementary row operations.

Let us assume that x_s is a basic variable and suppose $\boldsymbol{B}$ and $\boldsymbol{N}$ are basic and nonbasic columns, respectively, of the previous solution. We delete the column and row of $\boldsymbol{B}$ corresponding to x_s and the column and row of $\boldsymbol{N}$ corresponding to its dual variable w_s. 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 and $\boldsymbol{B} \in \mathbb{R}^{n\times n}$, so that:

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

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

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

thus,

$$\overline{\boldsymbol{U}\boldsymbol{B}} = \boldsymbol{I}_{n-1} - \boldsymbol{U_{col}}\boldsymbol{B_{row}}^\top. \tag{4}$$

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

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 $\boldsymbol{M}_B = -\boldsymbol{B}^{-1}\boldsymbol{N}$ 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,

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

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

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

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

$$\boldsymbol{M}_{\bar{B}} = \boldsymbol{E}(\overline{\boldsymbol{M}} + \boldsymbol{U}_{col}\boldsymbol{N}_{row}^{\top}). \tag{5}$$

There are several assumptions 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 $\overline{\boldsymbol{B}} = \boldsymbol{I}_{n-1}$. 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 nth column from $\boldsymbol{B}$, $\boldsymbol{B}^{-1}$, $\boldsymbol{N}$ and $\boldsymbol{M}_B$. The general case can be easily modified to this special case. Finally, if x_s is a nonbasic variable in the previous solution, we can apply the following methodology for its complementary variable w_s which must be a basic variable.

To update $\boldsymbol{q}_B$, we delete the sth 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:

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

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

$$\boldsymbol{q}_{\bar{B}} = \boldsymbol{E}(\tilde{\boldsymbol{q}}_B - q_s\boldsymbol{U}_{\boldsymbol{col}}). \tag{6}$$

LU decomposition of the basis

From (5) and (6), we only need 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 [22] to construct the LU decomposition of $\boldsymbol{B}$ and derive the sth column of $\boldsymbol{B}^{-1}$ using back-substitution. If x_s is the ith basic variable, then we get $\boldsymbol{U}_{col}$ by deleting the ith element of the column. Given $\boldsymbol{B}$, $\boldsymbol{N}$ and $\boldsymbol{U}_{col}$, we can update $\boldsymbol{M}$ and $\boldsymbol{q}$ according to (5) and (6), respectively.

2.3 A heuristic

To find a good feasible solution at the root node, we run a heuristic which combines the heuristics proposed by [4] and [12]. Let $\boldsymbol{x^*}$ be the solution of the continuous relaxation at the root node. We first run what [12] refers to as the "reoptimization heuristic" which "reflects common practice", where the continuous relaxation is solved again using only the variables with the K largest absolute values of $\boldsymbol{x}^*$. The lower-bound constraint $x_i \geq \alpha_i$ is also imposed for these variables. If this problem is feasible, the solution is a feasible solution to the original CCQO and let UB_0 be its corresponding objective value. To improve on UB_0, we then run the heuristic proposed by [4]. Let $G = \{i \mid |x_i^*|$ is one of the $K + W$ largest absolute values of $\boldsymbol{x}^*\}$, where W is a user-defined small positive integer such that $|G| = K + W \ll d$ (we have set $W = 0.1d$ in our computational experiments). We then solve the CCQO problem using only the variables in G, setting UB_0 as the initial upper-bound to the optimal value. We also put a limit on the number of nodes to examine. Thus, we are implementing our branch-and-bound procedure just on the variables in G.

3 Applications of CCQO

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

3.1 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 $\sum_i (y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta})^2$ is minimized. This has a closed-form solution, $\boldsymbol{\beta} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}$, where $\boldsymbol{X} \in \mathbb{R}^{m \times d}$ with $\boldsymbol{x}_i^\top$ as its ith row, and $\boldsymbol{Y} \in \mathbb{R}^m$ with y_i as its ith 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 [1, 18, 23]). For example, suppose we want to choose K variables $(K < d)$ that minimizes the total sum of squared errors. We formulate this problem as:

$$\begin{array}{ll} \text{minimize} & (\boldsymbol{Y} - \boldsymbol{X\beta})^\top (\boldsymbol{Y} - \boldsymbol{X\beta}), \\ \text{subject to} & |\operatorname{supp}(\boldsymbol{\beta})| \leq K, \end{array} \tag{7}$$

which is clearly a CCQO.

Authors of [2, 9, 10] use pivoting and enumeration to search for subsets with the best regression "fit" (e.g., minimum total sum of squared errors) for subsets of all sizes. Authors of [19] solve linear mixed-integer optimization problems that find a subset of K variables ($K < d$) that has the minimum total absolute error.

We solve (7) by tailoring our approach to this unconstrained version of a CCQO. When the cardinality constraint is relaxed, the optimal objective value is $\boldsymbol{Y}^\top \boldsymbol{Y} - \boldsymbol{Y}^\top \boldsymbol{X}^\top (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}$, thus we do not need to run the Lemke's method. The main computational work is in the branch down procedure. Clearly, we can extend our method to regression problems with linear constraints with respect to the $\boldsymbol{\beta}$'s by applying our general methodology. However to highlight the merits of our tailored approach versus a general purpose software such as CPLEX, we focus on the unconstrained regression in this paper.

When branching down on x_s, we delete the sth row and column of $\boldsymbol{X}^\top \boldsymbol{X}$, and the inverse $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ is updated as illustrated in Sect. 2.2. To further alleviate computation, we set $\boldsymbol{v} = \boldsymbol{X}^\top \boldsymbol{Y} \in \mathbb{R}^d$ at the root node. Thus, deleting x_s corresponds to deleting the sth element of $\boldsymbol{v}$. We do not need to multiply $\boldsymbol{X}^\top$ 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:

$$\boldsymbol{Y}^\top \boldsymbol{Y} - \overline{\boldsymbol{v}}^\top (\overline{\boldsymbol{X}^\top \boldsymbol{X}})^{-1} \overline{\boldsymbol{v}},$$

where $\overline{\boldsymbol{X}^\top \boldsymbol{X}}$ and $\overline{\boldsymbol{v}}$ are the updated $\boldsymbol{X}^\top \boldsymbol{X}$ and $\boldsymbol{v}$, respectively. Thus, calculating the objective value 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

Let us consider the traditional mean-variance portfolio optimization problem, which can be modeled 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, [24, 25] approximate the quadratic objective function by a linear and piece-wise linear function, and [11] further assumes equal weights across assets to formulate

the problem as a pure 0-1 problem. In [21], portfolio problems with fixed transaction costs are solved in polynomial time when the covariance matrix has equal off-diagonal elements. The second approach uses heuristics to find strong feasible solutions. For example, [17] proposes a linear mixed-integer optimization based heuristic, [5] introduces a dynamic programming based heuristic, and [6] proposes genetic algorithm, tabu search and simulated annealing approaches for the problem.

There have been also significant efforts in finding exact algorithms to solve discrete portfolio optimization problems. For example, [20] extends the work of [4] to limited diversification portfolios, and [12] solves portfolio problems with minimum transaction levels, limited diversification and round lot constraints (which requires investing in discrete units) in a branch-and-bound context. In [15], the authors solve a portfolio optimization problem that maximizes net returns where the transaction costs are modeled by a concave function. They successively estimate the concave function by a piecewise linear function and solve the resulting LP. They have shown that their solution converges to the optimal solution of the original problem. In [16], the authors present a divide-and-conquer algorithm that partitions the feasible set of the solutions to find the exact solution to a problem with fixed transaction costs and round lots.

In this paper, we focus on the traditional mean-variance portfolio optimization model with cardinality constraints. The key difference between our approach and those described above is our use of Lemke's pivoting algorithm to solve the underlying quadratic program in our branch-and-bound implementation. Let us further suppose that the d stocks can be categorized into S industry sectors. Investors may wish to limit the total investment change within each of S industry sectors. Let the current portfolio weights be $\boldsymbol{x}^0 \in \mathbb{R}^d$. 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 depends on 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 modeled by the following quadratic function

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

where $c_i > 0$ 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 CCQO framework using binary variables, but we will not include it in the present work because ticket costs are often second order compared to impact costs. This portfolio selection problem can be

represented by the following formulation:

$$\begin{aligned}
\text{minimize} \quad & -\boldsymbol{r}^\top \boldsymbol{x} + \tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^B)^\top \boldsymbol{\Sigma}(\boldsymbol{x} - \boldsymbol{x}^B) + \sum_{i=1}^{d} c_i (x_i - x_i^0)^2, \\
\text{subject to} \quad & \left| \sum_{i \in S_l} (x_i - x_i^B) \right| \le \epsilon_l, \quad l = 1, \ldots, S, \\
& \sum_{i=1}^{d} x_i = 1, \\
& |\operatorname{supp}(\boldsymbol{x})| \le K, \\
& x_i \ge \alpha_i, \quad i \in \operatorname{supp}(\boldsymbol{x}), \\
& x_i \ge 0, \quad i = 1, \ldots, d,
\end{aligned} \tag{8}$$

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 (8) is in the form of Problem (1) and can be solved using our methodology.

We rewrite Problem (8) as

$$\begin{aligned}
\text{minimize} \quad & -\tilde{\boldsymbol{r}}^\top \boldsymbol{x} + \tfrac{1}{2}\boldsymbol{x}^\top \tilde{\boldsymbol{\Sigma}} \boldsymbol{x} + C_0, \\
\text{subject to} \quad & \sum_{i \in S_l} x_i \le \epsilon_l + \sum_{i \in S_l} x_i^B, \quad l = 1, \ldots, S, \\
& -\sum_{i \in S_l} x_i \le \epsilon_l - \sum_{i \in S_l} x_i^B, \quad l = 1, \ldots, S, \\
& \sum_{i=1}^{d} x_i = 1, \\
& |\operatorname{supp}(\boldsymbol{x})| \le K, \\
& x_i \ge \alpha_i, \quad i \in \operatorname{supp}(\boldsymbol{x}), \\
& x_i \ge 0, \quad i = 1, \ldots, d,
\end{aligned} \tag{9}$$

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 ith diagonal element, and C_0 is a constant equal to $(\boldsymbol{x}^B)^\top \boldsymbol{\Sigma}\boldsymbol{x}^B + \sum_{i=1}^{d} c_i (x_i^0)^2$.

We solve the relaxation of Problem (9) using Lemke's method described in Sect. 2.1, and branch down on a variable as in Sect. 2.2. Section 4.2 illustrates the computational results of this method.

4 Computational results

We describe computational experiments on subset selection and portfolio selection problems in Sects. 4.1 and 4.2, respectively. For each problem, we compare our tailored approaches to CPLEX's quadratic mixed integer programming solver [8]. Clearly, CPLEX's implementation of the pivoting methods and branch-and-bound is far superior to ours, however, our motive is to measure the advantages of a tailored implementation over a general mixed-integer solver for these particular CCQO problems.

4.1 Results for subset selection

We compared our branch-and-bound implementation for solving subset selection with forward regression and CPLEX's quadratic mixed-integer programming solver. 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, $\beta_{(1)}$, corresponds to $\beta_{(1)} = \arg\min_{j=1,\ldots,d} \sum_i (y_i - x_{i,j}\beta_j)^2$. 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 $\bar{y}_i = y_i - x_{i,(1)}\beta_{(1)}$. This step is repeated until K variables are chosen [18].

We used CPLEX's quadratic mixed-integer optimizer to solve Problem (7) 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. By comparing our method with CPLEX, we hope to see the computational advantages, if any, of not using binary variables z_i and branching directly on β_i's. One obvious benefit is the elimination of the need for the so called "big-M" constraints.

Our branch-and-bound and CPLEX's branch-and-bound search procedure used depth-first-search, and branches on the variable with maximum absolute value first. In our algorithm, we ran the heuristic presented in Sect. 2.3 after solving the continuous relaxation of the root node and not in any subsequent nodes.

For each d (the number of variables), we randomly generated five instances of $\boldsymbol{X}$ and $\boldsymbol{\beta}$, and set $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, where $\epsilon_i \sim N(0,1)$ for each i. For each problem size, we present the average performance of the methods over all five instances in Tables 1 and 2. The performances of the individual instances are presented in Tables 6 and 7 in the Appendix. In all of the tables, the columns "Forward", "BnB", and "CplexMIQP" correspond to the results of the forward regression, our method, and CPLEX, respectively. We did not record the running time for forward regression since this simple heuristic was able to solve almost all the instances in a fraction of a second. The column labeled "time" is the total CPU seconds required to solve the problem up to a specified time limit (discussed below). This number includes the running time of the root node heuristic as well. The column labeled "nodes" is the total number of nodes in the branch-and-bound tree at termination, "best node" is the node corresponding to the best feasible solution and "RSS" is the best total sum of squared errors found for subsets of size K. All the numbers, except for the CPU time, were rounded to the nearest integer.

Table 1 Results for Subset Selection with 60 CPU seconds. The column "time" is in CPU seconds and "RSS" is the residual sum of squares

d	K	Forward	BnB				CplexMIQP			
		RSS	Time	Nodes	Best node	RSS	Time	Nodes	Best node	RSS
20	10	7,518	0.03	234	116	4,306	0.04	249	140	4,306
20	5	22,358	0.02	174	2	19,343	0.05	266	247	19,343
50	40	37,455	1.89	1,149	0	2,550	15.06	35,805	35,498	2,552
50	20	113,515	60.13	88,729	7,595	65,301	60.33	167,208	0	72,272
100	80	359,586	24.69	2,362	1,275	6,610	60.15	53,609	52,943	674,280
100	50	487,124	60.32	9,103	2,338	116,265	60.11	54,750	0	692,558
100	20	815,097	60.18	34,425	5,077	636,623	60.12	53,547	0	692,558
500	400	11,914,096	108.46	3	0	64,229	60.36	346	118	83,047,520
500	100	22,246,620	61.26	3	0	15,534,560	60.34	302	113	83,122,180
500	20	36,746,520	64.04	1,489	868	35,458,480	60.80	130	0	102,030,200

Table 2 Results for Subset Selection with 3600 CPU seconds. The column "time" is in CPU seconds and "RSS" is the residual sum of squares

d	K	BnB				CplexMIQP			
		Time	Nodes	Best node	RSS	Time	Nodes	Best node	RSS
50	20	310.28	485,646	7,595	65,301	323.85	893,967	774,687	65,301
100	80	483.97	82,489	77,020	5,851	3,255.31	2,842,164	2,836,986	110,216
100	50	3,600.00	655,182	11,957	115,926	3,600.00	3,324,088	0	692,558
100	20	3,600.00	2154,433	5,077	636,623	3,600.00	3,192,809	0	692,558
500	400	3,600.00	7,419	501	63,033	3,600.00	62,448	62,411	66,477,040
500	100	3,600.00	51,637	0	15,534,560	3,600.00	105,140	46,161	64,677,580
500	20	3,600.00	118,422	61,716	35,372,860	3,600.00	131,396	23,800	74,620,900

Table 1 shows the results when the time limit for both our method and CPLEX was set 60 CPU seconds. These results illustrate whether these exact methods can find a "good" feasible solution relatively quickly. It is apparent from this table that the exact approaches significantly improve upon the forward regression in terms of residual sum of squares, even when they do not solve to provable optimality. Both "BnB" and CPLEX solved the problems with $(d, K) = (20, 10), (20, 5), (50, 40)$ to provable optimality in several seconds. However, in all cases where CPLEX does not find the optimal solution within 60 seconds, the RSS of "BnB" is consistently lower than that of CPLEX. This is most evident in cases where K is large relative to d, i.e., for $(d, K) = (100, 80)$ and $(d, K) = (500, 400)$. CPLEX performs especially poorly in these cases, having RSS values worse than that of the forward heuristic. In contrast, "BnB" appears to do especially well, having significantly lower RSS values than the other two methods. For three out of the five instances with $(d, K) = (100, 80)$,

our method found the provably optimal solution in under one second (see Table 6). From the "best node" column, it is clear that our heuristic, which applies our branch-and-bound procedure on a smaller subproblem of the original CCQO, yields good solutions. The CPLEX routine also runs a general heuristic at the root node, but it does not appear to be as effective for these problems.

These results show that in a minute or less, our method is able to provide solutions that are often substantially better than that of the forward heuristic. Thus, if speed is important, using our method with a short time limit may be a viable alternative to the forward heuristic.

Table 2 illustrates the results when our method and CPLEX are run for 3600 CPU seconds. For $(d, K) = (50, 20)$, both "BnB" and CPLEX solved to provable optimality within minutes, where "BnB" had faster running times in three out of the five instances. Our method also solves all five instances of $(d, K) = (100, 80)$ in an average of 8 minutes, however, it is surprising to see that CPLEX could not solve three out of the five instances within one hour (two of them being instances that "BnB" solved in under one minute—see Table 7). CPLEX solved the remaining two instances in about 45 minutes each. As in Table 1, CPLEX performs relatively poorly when K is large, namely for $(d, K) = (100, 80)$ and $(500, 400)$. It is not clear why these instances are especially difficult for CPLEX to find a good feasible solution. Again, in every single instance where CPLEX did not find the optimal solution, the best solution found by "BnB" was superior to that of CPLEX. We also see that our heuristic solution is still very strong. With the longer running time, our method is able to improve on the heuristic solution, however, it is often very close to the best RSS value found in one hour.

We selected some problem instances and ran CPLEX for up to 12 hours in hopes of finding a provably optimal solution. However, CPLEX could not find a better solution than those found by "BnB" in one hour. For example, in one of the instance of $(d, K) = (100, 20)$ (this is problem instance $(d, K, v) = (100, 20, 1)$ in Table 7 of the Appendix), CPLEX could not find the optimal solution in 12 hours. Its best RSS value was $758,261$ (which was found by its root node heuristic) whereas the RSS value found by "BnB" in one hour was $690,532$. In one of the instance of $(d, K) = (500, 400)$ (this is problem instance $(d, K, v) = (500, 400, 1)$ in Table 7 of the Appendix), CPLEX's best RSS value was $56,390,541$, whereas the RSS value found by "BnB" in one hour was $38,781$. Thus, it appears that this general solver is not well suited to solved this particular type of CCQO.

The difference in the two methods is most likely due to the difficulty of solving the explicit mixed-integer quadratic program formulation of the subset selection problem with the binary variables and big-M constraints. These big-M constraints can lead to weak LP relaxations, making it hard to fathom nodes. In addition, our branch-and-bound based heuristic appears to be very effective in finding strong feasible solutions. This combination allows us to fathom many more nodes than CPLEX.

Note that the per node computation time for "BnB" decreases as K decreases. This is highlighted in $d = 100$ and $d = 500$, where the average number of nodes explored increases significantly as K decreases. This is because we delete variables that are branched down, making the subproblem smaller and smaller as we go down the branch-and-bound tree until at most $d - K$ variables are deleted. Thus, many of

the subproblems solved when $K = 20$ is much smaller than when $K = 400$, resulting in the difference in average per node computation time.

The main bottleneck of our method 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 a million nodes to prove optimality, even for moderate sized problems. 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. However, provable optimality may not be of critical importance to data mining practitioners who may be interested in finding a "good" solution "quickly". From these empirical studies, we have seen that our branch-and-bound procedure finds near-optimal solutions in one minute. Given the noise in the data, perhaps these solutions are sufficient in practice.

4.2 Results for portfolio selection

We tested our approaches described in Sect. 3.2 against two alternative methods. One method uses CPLEX's quadratic barrier method to solve the relaxation problem of (9), and the second uses CPLEX's quadratic mixed-integer solver to solve the explicit mixed-integer formulation. All branch-and-bound procedures, including CPLEX, were set to depth-first-search, branch up first and branch on variable with maximum absolute value. Depth-first-search was used primarily due to its ease of implementation and limited memory usage.

For each d (total number of assets), S (number of sectors), and K (the upper bound on diversification), we generated five random instance of Problem (8) and averaged the results. We set $\boldsymbol{x}^0 = \mathbf{0}$ for all of our instances. For $\boldsymbol{\Sigma}$, we generated a matrix $\boldsymbol{A}$ from a Gaussian distribution, then used $\frac{d}{d-1}\boldsymbol{A}^T\boldsymbol{A}$ as our covariance matrix. The values of $\boldsymbol{r}$ and c_i were taken from a Gaussian and uniform distribution, respectively. We acknowledge that using randomly generated data may give unrealistically optimistic running times. However, since our interest is to compare the computational performance of different solution methods, we hope that this relative difference extends to real stock data as well. Tables 3 and 4 illustrate the results. In these tables, "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 labeled "-" indicate that the method failed to find a feasible solution within 120 CPU seconds.

"LemkeBnB" refers to our method described in Sect. 2. "BarrierBnB" is the same as "LemkeBnB", except we use CPLEX's barrier method to solve the continuous quadratic optimization problem. Finally, "CplexMIQP" is the result of using CPLEX quadratic mixed-integer solver. All three methods were run for a total of 120 CPU seconds. The column labeled "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 three methods without running any heuristic methods to find a good feasible solution, whereas Table 4 runs a heuristic once at the root. For "LemkeBnB" and "BarrierBnB", we ran the heuristic for at most 30 CPU seconds after the root is solved. We were not able to put a time limit on the heuristic for

Table 3 Results for portfolio selection, *without* Heuristic, solved until 120 CPU seconds

d	K	S	LemkeBnB			BarrierBnB			CplexMIQP		
			Nodes	Best node	UB	Nodes	Best node	UB	Nodes	Best node	UB
100	50	10	16992.20	16039.60	28.46	4526.80	4472.00	44.51	119697.00	76987.00	19.02
100	10	4	27231.40	8329.40	18.83	5686.00	1678.20	19.57	58530.20	12736.60	18.49
200	100	10	2952.80	2933.80	142.73	751.20	729.40	150.57	30188.60	30062.80	81.86
200	20	4	6913.20	1281.00	38.73	1284.00	656.20	39.77	7236.80	1245.60	38.73
500	200	10	142.40	–	–	30.80	–	–	4864.00	4812.60	345.46
500	100	10	164.40	137.80	159.68	33.20	–	–	782.60	398.00	158.68
500	50	4	64.20	46.40	90.89	35.40	–	–	226.40	140.40	90.89

Table 4 Results for portfolio selection, with the root Heuristic, solved until 120 CPU seconds

d	K	S	LemkeBnB			BarrierBnB			CplexMIQP		
			Nodes	Best node	UB	Nodes	Best node	UB	Nodes	Best node	UB
100	50	10	13998.40	0.00	12.93	4029.40	2678.20	36.36	116468.20	70551.00	18.68
100	10	4	23926.40	0.00	14.87	4166.40	0.00	15.66	58410.40	12736.60	18.49
200	100	10	2490.60	0.00	32.90	485.40	0.00	34.51	18533.20	7800.80	52.84
200	20	4	6232.80	58.60	34.87	916.40	58.60	35.58	6937.00	1245.60	38.73
500	200	10	119.40	0.00	83.93	24.60	–	–	1098.00	0.00	138.83
500	100	10	134.00	0.00	80.40	27.00	–	–	1097.40	0.00	140.29
500	50	4	47.00	0.00	81.53	27.40	–	–	146.40	103.20	91.14

CplexMIQP, which ran until CPLEX deemed it had an acceptable upper-bound. 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., $K \leq 0.1d$).

Both pivoting-based methods, "LemkeBnB" and "CplexMIQP", are significantly faster than "BarrierBnB" 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 generated instances 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 "LemkeBnB" for most instances. However, the relative difference significantly decreases as K becomes small relative to d. For example, when $K \approx 0.5d$, the differ-

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

d	K	S	LemkeBnB			CplexMIQP		
			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

ence 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 "LemkeBnB" and "CplexMIQP" are about the same. We reach similar conclusions when we increase the running time. Table 5 shows results for running our method 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. Using $|G|$ small enough so that Lemke's method can run sufficiently fast allowed us to find a good feasible solution quickly.

5 Conclusion

From this computational study, we learn that:

1. Our tailored approaches for solving cardinality constrained quadratic optimization problems in regression and portfolio optimization show some computational advantages over a general mixed-integer solver.
2. For subset selection in regression, our method was able to find the subset of variables with significantly better fit than the forward regression heuristic even with a 60 second time limit. The results also show that our approach has significant computational advantages to CPLEX which needs to solve an explicit mixed-integer quadratic formulation with big-M constraints.
3. For the portfolio selection problem, the combination of our branch-and-bound implementation 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 and Lemke's method with non-zero lower bounds updates each subproblem without increasing the size of the problem. With further improvements in implementation (e.g., regarding data struc-

tures, 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 entering variable. Alternative pivoting methods, though more difficult to initialize, have the flexibility of choosing amongst several entering 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.

Our goal in this work was to investigate the computational merit of tailored branch-and-bound implementations that does not require introducing binary variables for solving subset selection in regression and portfolio selection problem in asset management. To find good (but not necessarily provably optimal) solutions quickly, these approaches appear to have advantages over generalized solvers. We hope to further improve our implementation and explore the practicality of our algorithm to other examples of CCQOs.

Appendix A: Additional tables

Table 6 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. Five different instances for each (d, K) pair were solved and v denotes the instance number

d	K	v	Forward	BnB				CplexMIQP			
			RSS	CPU sec.	# nodes	Best node	RSS	CPU sec.	# nodes	Best node	RSS
20	10	1	5,730	0.08	757	481	4,882	0.10	776	366	4,881
20	10	2	12,391	0.01	69	0	5,223	0.03	118	91	5,223
20	10	3	5,572	0.00	27	0	2,733	0.02	50	49	2,733
20	10	4	6,540	0.04	283	99	4,018	0.05	268	196	4,018
20	10	5	7,356	0.00	33	0	4,673	0.02	32	0	4,673
20	5	1	14,406	0.00	57	0	11,275	0.02	101	100	11,275
20	5	2	26,221	0.02	227	0	25,162	0.07	408	364	25,162
20	5	3	26,333	0.02	177	0	18,658	0.04	219	191	18,658
20	5	4	11,440	0.01	115	0	11,440	0.04	196	174	11,440
20	5	5	33,392	0.03	295	10	30,181	0.06	406	404	30,181
50	40	1	66,070	0.13	85	0	489	14.86	31,631	31,630	489
50	40	2	27,970	8.52	5,125	0	6,433	26.06	71,521	70,266	6,433
50	40	3	13,662	0.12	83	0	465	21.64	45,194	45,193	465
50	40	4	11,864	0.14	93	0	2,175	5.41	12,497	12,497	2,175
50	40	5	67,712	0.53	357	0	3,190	7.34	18,180	17,905	3,190

Table 6 (*Continued*)

d	K	v	Forward	BnB				CplexMIQP			
			RSS	CPU sec.	# nodes	Best node	RSS	CPU sec.	# nodes	Best node	RSS
50	20	1	94,662	60.00	88,631	0	47,481	60.00	166,669	0	55,029
50	20	2	173,330	60.00	88,147	0	99,159	60.00	168,326	0	101,614
50	20	3	101,757	60.00	90,017	37,975	66,051	60.00	167,051	0	81,622
50	20	4	70,347	60.00	87,941	0	52,589	60.00	167,051	0	53,437
50	20	5	127,480	60.00	88,907	0	61,223	60.00	166,943	0	69,659
100	80	1	339,592	60.00	4,035	0	16,108	60.00	53,260	52,648	722,841
100	80	2	410,155	0.82	161	0	954	60.00	53,985	51,888	626,166
100	80	3	376,343	60.00	7,289	6,377	13,899	60.00	54,479	54,450	664,734
100	80	4	302,706	0.81	161	0	965	60.00	53,463	52,927	761,439
100	80	5	369,136	0.96	163	0	1,126	60.00	52,856	52,804	596,220
100	50	1	611,976	60.00	9,193	0	129,706	60.00	54,517	0	758,261
100	50	2	461,085	60.00	8,929	8,741	104,644	60.00	54,738	0	630,966
100	50	3	443,940	60.00	9,213	0	131,327	60.00	55,002	0	678,234
100	50	4	379,711	60.00	9,165	63	111,738	60.00	54,826	0	776,809
100	50	5	538,908	60.00	9,013	2,885	103,912	60.00	54,668	0	618,520
100	20	1	918,579	60.00	35,619	25,385	690,532	60.00	52,773	0	758,261
100	20	2	807,784	60.00	33,031	0	591,700	60.00	53,273	0	630,966
100	20	3	758,718	60.00	31,655	0	622,728	60.00	54,000	0	678,234
100	20	4	793,692	60.00	33,889	0	713,576	60.00	53,694	0	776,809
100	20	5	796,714	60.00	37,931	0	564,578	60.00	53,994	0	618,520

Table 6 (*Continued*)

d	*K*	*v*	Forward	BnB				CplexMIQP			
			RSS	CPU sec.	# nodes	Best node	RSS	CPU sec.	# nodes	Best node	RSS
500	400	1	11,307,900	60.00	3	0	39,566	60.00	320	0	64,022,300
500	400	2	11,811,700	60.00	3	0	125,125	60.00	591	590	120,308,000
500	400	3	15,760,100	60.00	3	0	76,127	60.00	2	0	117,639,000
500	400	4	11,463,100	60.00	3	0	4,645	60.00	367	0	58,863,400
500	400	5	9,227,680	60.00	3	0	75,680	60.00	448	0	54,404,700
500	100	1	21,815,400	60.00	3	0	15,267,300	60.00	120	0	64,022,300
500	100	2	22,578,200	60.00	3	0	14,331,100	60.00	575	564	120,682,000
500	100	3	25,562,500	60.00	3	0	17,802,300	60.00	0	0	117,639,000
500	100	4	21,751,200	60.00	3	0	15,591,100	60.00	383	0	58,863,400
500	100	5	19,525,800	60.00	3	0	14,681,000	60.00	430	0	54,404,700
500	20	1	34,013,800	60.00	1,451	795	32,687,900	60.00	255	0	64,022,300
500	20	2	36,103,300	60.00	1,675	457	33,350,000	60.00	0	0	133,198,000
500	20	3	42,841,900	60.00	1,601	1,017	41,550,000	60.00	12	0	117,639,000
500	20	4	36,622,500	60.00	1,233	811	35,780,100	60.00	384	0	58,863,400
500	20	5	34,151,100	60.00	1,483	1,259	33,924,400	60.00	0	0	136,429,000

Table 7 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. Five different instances for each (d, K) pair were solved. v denotes the instance number

d	K	v	Forward	BnB				CplexMIQP			
			RSS	CPU sec.	# nodes	Best node	RSS	CPU sec.	# nodes	Best node	RSS
50	20	1	94,662	90.41	135,731	0	47,481	139.54	383,863	383,310	47,481
50	20	2	173,330	813.68	1,277,927	0	99,159	548.20	1,521,968	1,074,025	99,159
50	20	3	101,757	344.91	549,877	37,975	66,051	670.38	1,843,030	1,813,134	66,051
50	20	4	70,347	210.61	326,439	0	52,589	132.79	367,315	294,049	52,589
50	20	5	127,480	91.78	138,257	0	61,223	128.34	353,658	308,918	61,223
100	80	1	339,592	1,987.89	319,185	295,947	15,415	3,600.00	3,122,527	3,119,793	66,881
100	80	2	410,155	0.81	161	0	954	3,600.00	3,150,143	3,150,142	100,756
100	80	3	376,343	429.38	92,775	89,153	10,794	2,987.66	2,622,673	2,605,407	10,794
100	80	4	302,706	0.81	161	0	965	2,467.48	2,085,836	2,085,836	965
100	80	5	369,136	0.96	163	0	1,126	3,600.00	3,229,639	3,223,752	371,680
100	50	1	611,976	3,600.00	651,435	0	129,706	3,600.00	3,284,774	0	758,261
100	50	2	461,085	3,600.00	650,955	40,559	103,311	3,600.00	3,294,914	0	630,966
100	50	3	443,940	3,600.00	656,449	0	131,327	3,600.00	3,399,950	0	678,234
100	50	4	379,711	3,600.00	664,523	63	111,738	3,600.00	3,345,433	0	776,809
100	50	5	538,908	3,600.00	652,547	19,161	103,550	3,600.00	3,295,369	0	618,520

Table 7 (*Continued*)

d	K	v	Forward	BnB				CplexMIQP			
			RSS	CPU sec.	# nodes	Best node	RSS	CPU sec.	# nodes	Best node	RSS
100	20	1	918,579	3,600.00	2,206,247	25,385	690,532	3,600.00	3,167,071	0	758,261
100	20	2	807,784	3,600.00	2,044,353	0	591,700	3,600.00	3,174,397	0	630,966
100	20	3	758,718	3,600.00	2,069,317	0	622,728	3,600.00	3,222,568	0	678,234
100	20	4	793,692	3,600.00	2,158,699	0	713,576	3,600.00	3,209,023	0	776,809
100	20	5	796,714	3,600.00	2,293,547	0	564,578	3,600.00	3,190,984	0	618,520
500	400	1	11,307,900	3,600.00	26,687	2,507	38,781	3,600.00	102,689	102,688	62,104,041
500	400	2	11,811,700	3,600.00	1,675	0	124,883	3,600.00	58,455	58,354	77,880,026
500	400	3	15,760,100	3,600.00	4,329	0	73,667	3,600.00	49,378	49,306	79,180,639
500	400	4	11,463,100	3,600.00	3	0	4,640	3,600.00	51,088	51,081	58,845,161
500	400	5	9,227,680	3,600.00	4,403	0	73,195	3,600.00	50,630	50,628	54,375,334
500	100	1	21,815,400	3,600.00	50,165	0	15,267,300	3,600.00	55,981	0	64,022,341
500	100	2	22,578,200	3,600.00	51,543	0	14,331,100	3,600.00	132,033	131,000	72,052,226
500	100	3	25,562,500	3,600.00	49,763	0	17,802,300	3,600.00	99,900	99,804	74,045,239
500	100	4	21,751,200	3,600.00	52,161	0	15,591,100	3,600.00	122,689	0	58,863,361
500	100	5	19,525,800	3,600.00	54,551	0	14,681,000	3,600.00	115,099	0	54,404,734
500	20	1	34,013,800	3,600.00	105,517	70,717	32,626,300	3,600.00	117,459	0	64,022,341
500	20	2	36,103,300	3,600.00	149,009	46,825	33,120,700	3,600.00	151,840	62,000	100,628,626
500	20	3	42,841,900	3,600.00	122,813	112,443	41,534,300	3,600.00	151,242	57,000	95,185,439
500	20	4	36,622,500	3,600.00	107,375	811	35,780,100	3,600.00	121,559	0	58,863,361
500	20	5	34,151,100	3,600.00	107,395	77,783	33,802,900	3,600.00	114,882	0	54,404,734

References

1. Arthanari, T.S., Dodge, Y.: Mathematical Programming in Statistics. Wiley, New York (1993)
2. Beale, E.M.L., Kendall, M.G., Mann, D.W.: The discarding of variables in multivariate analysis. Biometrika **54**(3/4), 357–366 (1967)
3. Bertsimas, D., Darnell, C., Soucy, R.: Portfolio construction through mixed-integer programming at Grantham, Mayo, Van Otterloo and Company. Interfaces **29**(1), 49–66 (1999)
4. Bienstock, D.: Computational study on families of mixed-integer quadratic programming problems. Math. Program. **74**, 121–140 (1996)
5. Blog, B., van der Hoek, G., Rinnooy Kan, A.H.G., Timmer, G.T.: Optimal selection of small portfolio. Manag. Sci. **29**(7), 792–798 (1983)
6. Chang, T.J., Meade, N., Beasley, J.E., Sharaiha, Y.: Heuristics for cardinality constrained portfolio optimisation. Comput. Operat. Res. **27**, 1271–1302 (2000)
7. Cottle, R.W., Pang, J., Stone, R.E.: The Linear Complementarity Problem. Academic, San Diego (1992)
8. ILOG CPLEX 8.1 User Manual. ILOG CPLEX Division, Incline Village, NV (2002)
9. Furnival, G.M., Wilson Jr., R.W.: Regression by leaps and bounds. Technometrics **16**(4), 499–511 (1974)
10. Hockings, R.R., Leslie, R.N.: Selection of the best subset in regression analysis. Technometrics **9**(4), 531–540 (1967)
11. Jacob, N.: A limited diversification portfolio selection model for the small investor. J. Finance **29**(3), 847–856 (1974)
12. Jobst, N., Horniman, M., Lucas, C., Mitra, G.: Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints. Quant. Finance **1**(5), 489–501 (2001)
13. Lemke, C.E.: Bimatrix equilibrium points and mathematical programming. Manag. Sci. **11**(7), 681–689 (1965)
14. Lemke, C.E., Howson, J.T. Jr.: Equilibrium points of bimatrix games. J. Soc. Ind. Appl. Math. **12**(2), 413–423 (1964)
15. Konno, H., Wijayanayake, A.: Portfolio optimization problem under concave transaction costs and minimal transaction unit constraints. Math. Program. **89**, 233–250 (2001)
16. Mansini, R., Speranza, M.G.: An exact approach for portfolio selection with transaction costs and rounds. IIE Trans. **37**, 919–929 (2005)
17. Mansini, R., Speranza, M.G.: Heuristic algorithms for portfolio selection problem with minimum transaction lots. Eur. J. Operat. Res. **114**(1), 219–233 (1999)
18. Miller, A.: Subset Selection in Regression. Monographs on Statistics and Applied Probability, vol. 40. Chapman and Hall, London (1990)
19. Narula, S., Wellington, J.: Selection of variables in linear regression using the minimum sum of weighted absolute errors criterion. Technometrics **21**(3), 299–311 (1979)
20. Owens-Butera, G.: The solution of a class of limited diversification portfolio selection problems. Ph.D. thesis, Rice University, CRPC-TR97724-S (1997)
21. Patel, N., Subrahmanyam, M.: A simple algorithm for optimal portfolio selection with fixed transaction costs. Manag. Sci. **28**(3), 303–314 (1982)
22. Press, W.H., Flannery, B., Teukolsky, S., Vetterling, W.: Numerical Recipes in C, 2nd edn. Cambridge University Press, Cambridge (1992). http://www.nr.com
23. Ryan, T.P.: Modern Regression Methods. Wiley Series in Probability and Statistics. Wiley, New York (1997)
24. Sharpe, W.: A linear programming algorithm for mutual fund portfolio selection. Manag. Sci. **13**(7), 499–510 (1967)
25. Sharpe, W.: A linear programming approximation for the general portfolio analysis problem. J. Financ. Quant. Anal. **6**(5), 1263–1275 (1971)