KYBERNETIKA — VOLUME *62* (2026), NUMBER 3, PAGES 373–399

AN ALGORITHM FOR CARDINALITY-CONSTRAINED OPTIMIZATION WITH AN APPLICATION TO THE BEST SUBSET SELECTION AND SPARSE PORTFOLIO PROBLEMS

Vikram Singh and Min Sun

A lot of problems, from fields like sparse signal processing, statistics, portfolio selection, and machine learning, can be formulated as a cardinality-constrained optimization problem. The cardinality-constraint gives the problem a discrete nature, making it computationally challenging to solve as the dimension of the problem increases. In this work, we present an algorithm for solving the cardinality-constrained quadratic optimization problem, inspired by the interval branch-and-bound framework. The proposed method is guaranteed to find the global optimal solution and is capable of solving problems of a wide range of dimensions. In particular, we solve the classical best subset selection problem in regression and the cardinality-constrained portfolio optimization problem. The numerical results show that our algorithm is competitive with the state-of-the-art solvers to solve the best subset selection problem and is capable of solving the cardinality-constraint portfolio optimization problem in a practical amount of time.

Keywords: global optimization, cardinality-constraint, quadratic optimization, branch-and-bound, best subset selection, portfolio optimization

Classification: 90C26, 90C57, 65K05

1. INTRODUCTION

In recent years, there has been a growing interest among researchers to solve optimization problems subject to a Cardinality-Constraint (CC), perhaps because of its applications in various fields like high-dimensional statistics, machine learning, and sparse portfolio selection (see [31]). In the field of sparse signal approximation, the goal is to find a sparse array $x \in \mathbb{R}^p$ which fits the model $b = Ax + e$, where $A \in \mathbb{R}^{n \times p}$ is the matrix of linear measurements with $n << p$ and e represents noise (see [7]). This problem can be formulated as minimizing the quadratic function $\|b - Ax\|_2^2$ subject to a CC. Another optimization problem that can be formulated using a CC is the classical best subset selection problem in regression, which requires selecting a small number of predictors to be included in the model that minimizes the residual sum of squares (see [4, 22]). Machine learning and portfolio selection are examples of other important areas that would result in optimization problems with a bound on the number of nonzero entries

DOI: 10.14736/kyb-2026-3-0373

in the solution (see [1, 9]). Motivated by these applications, we consider the following Cardinality-Constrained Quadratic Optimization (CCQO) problem

$$\min_{x \in B} \quad f(x) := \frac{1}{2} x^T Q x + q^T x + c \quad \text{subject to} \quad \|x\|_0 \leq k, \tag{CCQO}$$

where $Q \in \mathbb{R}^{p \times p}$ is a symmetric positive semi-definite matrix, $q \in \mathbb{R}^p$, $B \subseteq \mathbb{R}^p$ is the constraint set consisting of equality and/or inequality constraints, $k < p$ is a positive integer, and $\|\cdot\|_0$ is a pseudo-norm which gives the number of nonzero entries in an array. The constraint $\|x\|_0 \leq k$ is called the CC. Note that even though the objective function is convex, the CC makes (CCQO) non-convex and NP-hard (see [26]). Due to the discrete nature of CC, we cannot use the existing techniques from the vast literature of continuous optimization to solve (CCQO); hence, designing new procedures is crucial.

Contribution In this paper, we propose a novel branch-and-bound algorithm to solve the (CCQO) problem with a guaranteed convergence to the optimal solution. The new algorithm converges to the optimal solution of (CCQO) in a finite number of iterations. We also propose an efficient local search method to get high-quality feasible points within our branch-and-bound algorithm, when B consists of only bound constraints. We demonstrate the efficiency of our algorithms by solving the best subset selection problem using synthetic datasets of dimensions up to 2000, along with a real Ozone dataset, and a semi-synthetic Leukemia dataset. We also demonstrate the application of our branch-and-bound algorithm to solve the cardinality-constrained portfolio optimization problem using synthetic datasets and a real-world dataset from captial market indices, when the number of assets is in 100s.

Notation For an index set I containing k indices, $q_I \in \mathbb{R}^k$ represents a subarray of q array indexed by I, and $Q_I \in \mathbb{R}^{k \times k}$ represents a submatrix of Q matrix with rows and columns indexed by I. $|I|$ represents the number of indices in I. All arrays are considered to be column arrays. The lowercase e represents an array of 1s with an appropriate dimension. $|x|$ returns the absolute value of x if x is a constant or returns an array with absolute values of all entries if x is an array.

Structure of the Paper The rest of the paper is organized as follows. In section 2, we briefly introduce the best subset selection and the cardinality-constrained portfolio optimization problems, along with two existing methods to solve the best subset selection problem. In section 3, we present the methodology of the new algorithm to solve (CCQO), along with its application to solve both problems. We present the numerical results in section 4 followed by the conclusion in section 5.

2. APPLICATIONS

In this section, we discuss two well-known problems that can be formulated as (CCQO): the best subset selection problem in linear regression and the cardinality-constrained portfolio optimization problem.

2.1. The best subset selection problem

Consider a linear regression model $y = X\beta + \varepsilon$, where $y \in \mathbb{R}^n$ is a response array, $X \in \mathbb{R}^{n\times p}$ is a design matrix, $\beta \in \mathbb{R}^p$ is an unknown predictor array, and $\varepsilon \in \mathbb{R}^n$ represents noise. The columns of X have been standardized to have zero mean and unit ℓ_2-norm. One common objective is to find a desired coefficient array β by minimizing the Residual Sum of Squares (RSS). This is the so-called ordinary least squares problem

$$\min_{\beta\in\mathbb{R}^p} \quad \|y - X\beta\|_2^2. \tag{OLS}$$

(OLS) is an unconstrained convex optimization problem and can be solved efficiently using many existing optimization algorithms. In particular, if X has full rank, its optimal solution is uniquely determined by $\hat{\beta} = (X^TX)^{-1}X^Ty$.

However, to better interpret the underlying data, we want to choose a model with only k (out of p) predictors that would fit the data well. This task gives rise to the following BSS problem

$$\min_{\beta\in\mathbb{R}^p} \quad \|y - X\beta\|_2^2 \quad \text{subject to} \quad \|\beta\|_0 \le k. \tag{BSS}$$

(BSS) is well known to be computationally challenging to solve as the number of possible subsets grows rapidly with an increase in the dimension of the problem. This leads to researchers devising greedy heuristics, which may not provide a solution to the (BSS) problem, but are computationally fast and can be used for high-dimensional data. Forward stepwise selection is a popular greedy search method that starts from an empty model and adds one predictor at a time until we get a model with a desired number of predictors in it (see [22]). In [35], the authors introduced an Adaptive Best Subset Selection (ABESS) algorithm that, under appropriate assumptions on the model parameters, finds the solution to (BSS) and has polynomial-time complexity. The highly celebrated LASSO introduced in [30] replaces the CC in the (BSS) problem with an ℓ_1-norm, resulting in the following convex optimization problem

$$\min_{\beta\in\mathbb{R}^p} \quad \|y - X\beta\|_2^2 + \lambda\|\beta\|_1, \tag{LASSO}$$

that can be solved efficiently. See [34] for a survey of the numerical techniques to solve the (LASSO) problem. The (BSS) problem enforces sparsity of the solution by explicitly using the CC in its formulation. On the other hand, LASSO uses the penalty parameter λ to get sparse solutions with penalized coefficients. In the past three decades, there has been a lot of research on understanding properties of the LASSO as a sparse estimator to perform selection and prediction using linear regression models (see [10]). It is also known that both BSS and LASSO have advantages over one another in different data settings, as demonstrated in [18]. Thus, finding new methods to solve the (BSS) problem is highly desirable. Next, we present two exact methods to solve the (BSS) problem.

2.1.1. Branch and bound algorithm

In pattern recognition literature, the Branch-and-Bound (BB) algorithm for feature selection using a monotone criterion function was first introduced by [25], and it has

become popular for solving the feature selection problem. Several variants have appeared since then (see [13, 15, 29]). In particular, [13] introduced an in-level node ordering to improve the BB algorithm by eliminating undesired features at early stages. Another improvement was given by [33], which provided a "minimum-solution-tree", a subtree of the original BB tree, that is enough to explore to get the optimal solution, saving some computational effort. Because of the unique structured nature of BB, it can be naturally visualized or represented by a tree. In particular, BB for (BSS) has been represented by a regression tree (see Fig. 2).

2.1.2. Mixed integer optimization formulation

In a recent work, [4] converted (BSS) into a mixed integer optimization problem defined as

$$\begin{aligned} \min_{\beta,z} \quad & \|y - X\beta\|_2^2 \\ \text{subject to} \quad & -Mz_i \leq \beta \leq Mz_i, \text{ for } i = 1, 2, \ldots, p, \\ & \sum_{i=1}^{p} z_i \leq k, \quad z \in \{0,1\}^p, \end{aligned} \tag{MIO}$$

where z is an array of binary variables and M is a technical uniform variable bound. The resulting problem can be solved using any MIO solver. In particular, they used two different problem formulations for $p \leq n$ and $p > n$ cases and demonstrated that solving (BSS) problem in higher dimensions is within reach now. A special formulation for the $p > n$ case results in a problem with dimension n, which is particularly useful when $p \gg n$ case.

2.2. The cardinality-constrained portfolio optimization problem

Portfolio optimization aims to select optimal proportions of assets to be included in the portfolio to minimize the risk, subject to a minimum level of return. To limit the number of assets in the optimal portfolio, a cardinality constraint has been added to get a more realistic model (see [20]). Given p assets with mean vector return $r \in \mathbb{R}^p$, covariance matrix $\Sigma \in \mathbb{R}^{p \times p}$ and expected portfolio return $R > 0$, suppose we want to include a maximum of $0 < k \leq p$ assets in the portfolio. Using the mean-variance Markowitz optimization model proposed in [21], we can formulate the cardinality-constrained portfolio optimization problem as follows

$$\begin{aligned} \min_{\beta \in \mathbb{R}^p} \quad & \beta^T \Sigma \beta \\ \text{subject to} \quad & e^T \beta = 1, \quad r^T \beta \geq R, \\ & 0 \leq \beta \leq u, \\ & \|\beta\|_0 \leq k, \end{aligned} \tag{CCPO}$$

where $u \in \mathbb{R}^p$ and $e \in \mathbb{R}^p$ is an array of 1s. (CCPO) problem can be converted to the following mixed integer portfolio optimization problem

$$\begin{aligned} \min_{\beta,z} \quad & \beta^T \Sigma \beta \\ \text{subject to} \quad & e^T\beta = 1, \;\; r^T\beta \geq R, \\ & 0 \leq \beta_i \leq u_i z_i, \;\; \text{for } i = 1, 2, \ldots, p, \\ & \sum_{i=1}^{p} z_i \leq k, \;\; z \in \{0,1\}^p, \end{aligned} \tag{MIPO}$$

where z is a binary variable. Over the past two decades there have been a high amount of research to effectively or reliably solve the (CCPO) problem (see [3, 5, 8, 14]).

3. A NOVEL BRANCH AND BOUND

In this section, firstly, we briefly describe the interval branch-and-bound method, and then we present the proposed branch-and-bound algorithm to solve the (CCQO) problem along with its application to solve the best subset selection and cardinality-constrained portfolio selection problem.

3.1. Interval branch and bound

Consider the Constrained Optimization Problem (COP) defined as follows

$$\begin{aligned} \min_{x \in B} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, m, \\ & g_j(x) \leq 0, \quad j = 1, \ldots, s, \end{aligned} \tag{COP}$$

where $B = [\underline{B}, \overline{B}]$ is the domain of the problem with $\underline{B}, \overline{B} \in \mathbb{R}^p$. The objective function f and the constraint functions h_i and g_j are real-valued scalar functions. A basic Interval Branch and Bound (IBB) based on Hansen's algorithm ([17]) to solve (COP) is given in Algorithm 1. Usually, no convexity for the objective and constraint functions is assumed, and the continuity of these functions would be sufficient to guarantee global convergence.

The idea of IBB algorithm is to maintain a list of boxes by repeatedly partitioning the parent box Y into smaller child boxes while looking for a feasible point that provides the smallest value of the objective function. The process starts from the original box B and an initial feasible point $\tilde{y}$. Throughout the process the algorithm keeps track of the current best function value $\tilde{f}$, which is initialized as equal to $f(\tilde{y})$. If we find a better feasible point from a child box, we update $\tilde{f}$ using that feasible point. In addition to searching for a better feasible point, the algorithm also incorporates some deletion conditions. A box is deleted using the bound-based deletion if the current best solution has a smaller function value then the infimum of the inclusion function of the current box. A box is also deleted if it does not contain any feasible point. There are well-established ways to construct an inclusion function F for a given function f as discussed in [23].

Algorithm 1: IBB

Input: p, f, B, F, G, H
Output: B^*, f^*

1 Take $Y := B$.
2 Find a feasible point $\tilde{y}$ and set $\tilde{f} = f(\tilde{y})$.
3 Set $y = \inf F(Y)$.
4 Initialize the list $L = \{(Y, \tilde{y}, y)\}$.
5 Select a node $(Y, \tilde{y}, y)$ from the list.
6 (*Branch*) Choose a coordinate direction η and partition the box Y_η at the midpoint to get two sub-boxes V_1 and V_2 such that $Y = V_1 \cup V_2$.
7 Remove $(Y, \tilde{y}, y)$ from the list L.
8 **for** *j=1,2* **do**
9 (*Feasibility sampling*) Find a feasible point $\tilde{v}_j$ in V_j. If V_j is infeasible, go to step 13, else, update $\tilde{f}$ if possible.
10 (*Bound*) Calculate $v_j = \inf\ F(V_j)$.
11 (*Deletion*) If $\tilde{f} < v_j$, go to step 13.
12 Add $(V_j, \tilde{v}_j, v_j)$ at the end of the list L.
13 End of j loop.
14 If a termination criterion holds, set B^* as the union of all the boxes in L, $f^* = \tilde{f}$, and **return**.
15 Go to step 5.

The convergence of Algorithm 1 is established in Theorem 2, section 5.7 in [27]. The theorem states that if U_i is the union of all the boxes present in the list L_i at the ith iteration, then $U_i \to B^*$, $\tilde{f}_i \to f^*$ as $i \to \infty$, where B^* is the set of global minimizers of (COP), $\tilde{f}_i$ is the current value of $\tilde{f}$ in the list L_i, and f^* is the global minimum objective function value of (COP). There are several stopping conditions that we can use in Algorithm 1. The desirable stopping conditions are as follows: • if the width of every box in the list is small enough; • if y is the smallest $\inf F(Y)$ for any box Y in the list and $\tilde{f} - y$ is small enough; • if the list becomes empty. Step 5 in Algorithm 1 requires selecting a box from the list for further partition. There are several possible ways in the literature to perform this step. The following are some popular options: • select the box with the minimum $\inf F(\cdot)$ of all the boxes in the list; • select the oldest box in the list; • select the box with the smallest width in the list; • select the box with the smallest width of $F(\cdot)$ of all the boxes in the list. There is no clear rule as to which option is the best. Depending on the problem, different selection options may show advantages.

3.2. Branch-and-bound for cardinality constraint

To the best of our knowledge, IBB has not been applied to solve (CCQO), perhaps because the CC is not included in the formulation of COP. Nevertheless, we intend to follow the framework of IBB to propose a numerical algorithm to solve the (CCQO). We would have to modify the IBB algorithm to make it possible to treat the CC effectively.

We now highlight the major features in IBB to take advantage of special properties of the (CCQO) problem.

3.2.1. Branching

The IBB would normally require repeated partitions of the search domain B to yield the desired global convergence. It is no longer necessary for the optimization problems with CC. In particular, we need to partition an interval only at 0; if an interval does not include 0, there is no need to partition the interval. If 0 is strictly between a and b, interval $[a, b]$ would be split into $[a, 0)$, $[0, 0]$, and $(0, b]$. If 0 is equal to a or b, $[a, b]$ would be split into two sub-intervals $[0, 0]$ and $(0, b]$ or $[0, 0]$ and $[a, 0)$, respectively. Hence, for the partition step in IBB, we get 3 child boxes if 0 is an interior point of Y_η and 2 child boxes if 0 is at the boundary of Y_η.

3.2.2. Bounding

For a working box V, a standard way of bounding $f(V) = \{f(x) : x \in V\}$ is done by constructing an inclusion function $F(\cdot)$ with $\inf F(V)$ as an acceptable lower bound of $f(V)$. Consequently, $\inf F(V)$ could also be used to test bound-based deletion conditions. However, if partitions occur only at zeroes, the bound-based deletion condition won't be effective since there is no chance to improve the accuracy of the bound after initial branching. In such a circumstance, we propose to use a tight lower bound of $f(V)$, denoted by $lb\, f(V)$, defined as

$$lb\ f(V) = \min \{f(x) : x \in V\}.$$

The convergence of the new algorithm will not follow directly from the convergence analysis of the IBB due to the special cut only at 0, as the mesh of the partition will not get smaller. However, if $lb\ f(V)$ is used, we still have global convergence to the optimal solution and monotone convergence to the optimal objective function value in a finite number of steps.

If bounding is done using $lb\ f(V)$ instead of $\inf\ F(V)$ we have to accept the round-off errors, and the use of interval box becomes optional. Instead, we use an integer flag to represent a box in each coordinate direction. For $a < 0$, $b > 0$, the flag 0 represents the degenerate interval $[0, 0]$, the flag 1 represents an interval containing 0 in it ($[a, 0]$,$[0, b]$, $[a, b]$), and the flag 2 represents an interval not containing 0 in it ($[a, 0)$ or $(0, b]$ or $[a, 0) \cup (0, b]$). Further, using an integer flag array to represent a box Y ensures that we get exactly 2 child boxes ($[0, 0]$ and $[a, 0) \cup (0, b]$) after partitioning Y along a coordinate direction. The initial box B can be represented by a $p-$dimensional array with all the components as integer 1.

3.2.3. Deletion

In addition to the standard bound-based deletion condition, we apply the CC to remove additional sub-boxes in the process. A working box V is infeasible, or all its sub-boxes are infeasible (thus V can be deleted) if it satisfies any one of these deletion conditions:

$(D1, V)$ $\sum_{i=1}^{p} 1(V_i = 2) > k$, the number of coordinate directions for the box V not containing 0 is greater than k;

$(D2, V)$ $p - k < \sum_{i=1}^{p} 1(V_i = 0)$, the number of coordinate directions for the box V fixed to 0 is greater than $p - k$;

$(D3, V)$ $\sum_{i=1}^{p} 1(V_i = 2) = k$, the number of coordinate directions for the box V not containing 0 is equal to k;

$(D4, V)$ $p - k = \sum_{i=1}^{p} 1(V_i = 0)$, the number of coordinate directions for the box V fixed to 0 is equal to $p - k$.

With all these strategies included, we obtain Algorithm 2 (BBCC) to solve (CCQO) that uses an integer flag array to represent every box and uses $lb\ f(\cdot)$ to check the bound-based deletion condition.

Algorithm 2: BBCC

Input: $p, k, \underline{B}, \overline{B}, f$
Output: x^*, f^*

1 Set Y to be a p-dimensional array of integers 1.
2 Find a feasible point $\tilde{y}$ and set $\tilde{f} = f(\tilde{y})$, $\tilde{x} = \tilde{y}$.
3 Set $y = lb\, f(Y)$.
4 Initialize the list $L = \{(Y, \tilde{y}, y)\}$.
5 **if** *L is empty* **then**
6 Set optimal point $x^* = \tilde{x}$, optimal value $f^* = \tilde{f}$, **return**.
7 **else**
8 Select a node $(Y, \tilde{y}, y)$ from the list L.
9 Choose a coordinate direction η such that $Y_\eta = 1$. If there is no such coordinate direction, go to step 5.
10 Partition the box Y to get two child boxes V_1 and V_2 such that $V_{1_j} = Y_j$ if $j \neq \eta$ with $V_{1_\eta} = 0$ and $V_{2_j} = Y_j$ if $j \neq \eta$ with $V_{2_\eta} = 2$.
11 Remove $(Y, \tilde{y}, y)$ from the list L.
12 **for** *j=1,2* **do**
13 If any of $(D1, V_j)$ or $(D2, V_j)$ hold, go to step 19.
14 If any of $(D3, V_j)$ or $(D4, V_j)$ hold, find a feasible point $\tilde{v}_j$ such that $f(\tilde{v}_j) = lb\, f(V_j)$, update $\tilde{f}$ (also $\tilde{x}$) if possible, go to step 19.
15 *(Feasibility sampling)* Find a feasible point $\tilde{v}_j$ in the box V_j. Update $\tilde{f}$ (also $\tilde{x}$) if possible.
16 (*Bound*) Calculate $v_j = lb\, f(V_j)$.
17 If $\tilde{f} < v_j$, go to step 19.
18 Add $(V_j, \tilde{v}_j, v_j)$ at the end of the list L.
19 Go to the next iteration of j loop.
20 Go to step 5.

Although BBCC was originally motivated by IBB, BBCC can no longer be classified as a version of IBB in the standard sense. The fact that BBCC generates only two child boxes after partitioning a box instead of three, as required, if we use an interval box, BBCC solves (CCQO) more efficiently. Table 1 shows the comparison of the important metrics for the IBB algorithm using modified branching and bounding procedures as discussed above, while still using the interval box and BBCC that uses an integer flag array to represent a box. BBCC uses fewer iterations, $f(\cdot)$ calls, and $lb\ f(\cdot)$ calls, and generates less number of boxes as compared to IBB.

	no. of iter.	no. of boxes gen.	no. of $f(\cdot)$ calls	no. of $lb\, f(\cdot)$ calls
IBB	1850	3701	2543	5550
BBCC	144	144	198	233

Tab. 1. Metrics comparing IBB with appropriate modifications for handling CC and BBCC algorithms to solve the BSS problem with $p = 30$ and $k = 5$.

The convergence of BBCC is stated in the following theorem.

Theorem 3.1. BBCC reaches the optimal solution of (CCQO) in a finite number of iterations.

P r o o f . Each box in BBCC is visited exactly once. We either delete a selected box (hence discarding all of its child boxes) or add it to the list for further branching. As there are a finite number of boxes to visit, BBCC will find the optimal solution in a finite number of iterations. In the worst case, the standard bound-based deletion condition is never satisfied before reaching the optimal solution. In that event, the tree generated by the BBCC would contain all the feasible subsets and an optimal solution would be identified since $lb\, f(\cdot)$ is used in BBCC. □

Selecting a new node from the list (step 8) to partition further is also a crucial step in any branch-and-bound algorithm. There are several selection criteria in the literature to select a node from the list (see [24]); among them, Depth-First Search (DFS) and Best-First Search (BFS) are two popular choices. In our framework, BFS corresponds to selecting a node with the smallest $lb\, f(\cdot)$ value, and DFS corresponds to selecting a node with the maximum number of 2 flags in the box.

An additional property of BBCC can be stated in terms of $T(p, k)$ and $T_n(p, k)$, where $T(p, k)$ defines the underlying tree of BBCC for given p and k values, and $T_n(p, k)$ defines the number of nodes (including the root and leaf nodes) of $T(p, k)$.

Proposition 3.2. For a given p and k, $T_n(p, k) = T_n(p-1, k) + T_n(p-1, k-1) + 1$.

P r o o f . The root node of $T(p, k)$ produces 2 child nodes. The one corresponding to flag 2 can be thought of as a root node for the tree $T(p-1, k-1)$, and the other one corresponding to flag 0 can be thought of as a root node for the tree $T(p-1, k)$. So, $T(p-1, k)$ and $T(p-1, k-1)$ are the subtrees of $T(p, k)$. Consequently, we get the above result. □

3.3. Solving the best subset selection using BBCC

The (BSS) can be re-formulated as (CCQO) problem and can be written as

$$\min_{\beta \in B} \quad \frac{1}{2}\beta^T Q\beta + q^T\beta + c \quad \text{subject to} \quad \|\beta\|_0 \leq k, \tag{BBSS}$$

where $Q = 2X^TX$, $q = -2X^Ty$, and $c = y^Ty$. If the domain B is big enough to contain the solution of (BSS), then (BSS) and (BBSS) will have the same optimal solution (see [4]). Common reasons to use the domain B may include: • ensuring a finite optimal solution; • facilitating certain effective search procedures; • incorporating any prior knowledge about the bounds of the unknown parameters. In step 16 of BBCC, for a working box V, if $J = \{j : V_j \neq 0, j = 1, \ldots, p\}$ is an index set, then

$$lb\, f(V) = \min_{x \in B_J} \quad \frac{1}{2}x^T Q_J\, x + q_J^T x + c,$$

where box $B_J = \prod_{j \in J} B_j$. We can use any convex quadratic optimization solver to solve the above problem.

3.3.1. Feasibility sampling

Finding good-quality feasible points within BBCC helps with more bound-based deletions and can accelerate the algorithm significantly to reach the optimal solution. We introduce Sequential Feature Swapping (SFS) given by Algorithm 3, which is an iterative procedure that relies on the idea of swapping bad predictors from the currently selected model with good predictors not in the model at each iteration, starting from a model with k predictors.

Let U be the set of indices of all the available predictors and I be the set of indices of the k predictors already selected in the model. The set $U \backslash I$ represents the complement of the set I w.r.t. U, and let

$$d(I) = \min_{x \in B_I} \quad \frac{1}{2}x^T Q_I\, x + q_I^T x + c.$$

For an index $s \in I$, we define the gain in the function value as $G(s) := d(I \backslash s) - d(I)$. For an index $s \in U \backslash I$, we define the reduction in the function value as $R(s) := d(I) - d(I \cup s)$. At each iteration of SFS, we drop the predictor with the minimum gain and pick the predictor with the maximum reduction to be included in the model. If the new model that we get after swapping the two predictors is better than the previous model, we accept the new model; otherwise, we stop the search, with the current model as the final output from SFS.

Algorithm 3: SFS

Input: k, U, B, d

Output: I^*

1 Choose an initial index set $I \subseteq U$ with k indices.

2 *(Drop)* Find the index $j \in I$ such that

$$j = \arg\min_{s \in I} \; G(s),$$

and set $\hat{I} = I \backslash j$.

3 *(Pick)* Find the index $i \in U \backslash I$ such that

$$i = \arg\max_{s \in U \backslash I} \; R(s).$$

4 **if** $d(\hat{I} \cup i) < d(I)$ **then**

5 *(Switch)* Update $I = \hat{I} \cup i$. Go to step 2.

6 **else**

7 Set $I^* = I$, **return**.

Proposition 3.3. SFS terminates after a finite number of iterations.

P r o o f. At each iteration, for the updated set I, the value of $d(I)$ decreases monotonically, and $d(I)$ is bounded from below by $d(I^*)$, where I^* is the optimal set of indices of k predictors for the (BBSS) problem. Hence, SFS terminates after a finite number of iterations. □

We note that SFS shares a similar idea as in Efroymson's stepwise algorithm (see [22]) with the difference that SFS starts when we already have k predictors selected in the model, whereas the Efroymson stepwise algorithm starts with no predictor in the model and sequentially selects a new predictor with a check included to drop some previously selected predictor from the model at each step. Additionally, see the "splicing algorithm" described in [35], which further generalizes the idea of SFS by letting more than one predictor be swapped to pick a desirable sparse model. Swapping more than one predictor increases the chance of finding a better feasible point at the cost of an increase in the computation time.

3.3.2. Bounding using QR decomposition

We can also use a recursive way of updating the $lb\, f(\cdot)$ of a child box using the $lb\, f(\cdot)$ of the parent box. Suppose the design matrix X has full column rank. Using QRD we have $Q^T X = R$, where $Q \in \mathbb{R}^{n \times p}$ is an orthogonal matrix, $R \in \mathbb{R}^{p \times p}$ is an upper triangular matrix. For the initial box Y, $lb\, f(Y) = f(\hat{\beta})$ where $\hat{\beta} = \arg\min_\beta \|y - X\beta\|_2^2 = R^{-1} Q^T y$. Suppose we partition the initial box Y along the coordinate direction η to get two child boxes. The child box with flag 2 has the same $lb\, f(\cdot)$ as its parent box Y. The child

box V with flag 0 corresponds to the linear model $y = X_1\beta + \varepsilon$ with the matrix X_1 obtained from X by dropping its η^{th} column. We can compute $lb\, f(V)$ as follows. Update the Q and R matrices to get $Q_1^T X_1 = R_1$ where $Q_1 \in \mathbb{R}^{(p-1)\times(p-1)}$ is an orthogonal matrix and $R_1 \in \mathbb{R}^{(p-1)\times(p-1)}$ is an upper triangular matrix (see [32] for updating QRD after dropping a column from the X matrix). Then, $lb\, f(V) = f(\beta_1)$ where $\beta_1 = \arg\min_\beta \|y - X_1\beta\|_2^2 = R_1^{-1}Q_1^T y$. We can modify Q_1 and R_1 to find $lb\, f(\cdot)$ for the child boxes of V in a similar way. However, this procedure is not very efficient in practice, because we have to save the Q and R matrices for each box as attributes of a node, which uses a lot of memory space.

3.3.3. BBCC versus BB

Both BBCC and BB are globally convergent algorithms. Figures 1 and 2 show BBCC and BB trees for a small instance where we want to choose 2 predictors out of 5 available predictors, assuming no bound-based deletion condition has taken place and we are branching on the first available variable in a box. Each of the gray boxes in the BBCC tree has the same $lb\, f(\cdot)$ value as their parent boxes. Thus, in the BBCC tree, we do not have to compute $lb\, f(\cdot)$ for one of the child boxes unless it is a terminal box where the remaining flag 1s are changed to flag 0s. On the other hand, in the BB tree, we evaluate $lb\, f(\cdot)$ at each node by removing the predictor given by the number inside the node from the current model, one at a time, where 0 represents the root node with all the predictors included. The gray nodes in the BB tree show the nodes that can be skipped using the "minimum-solution-tree" strategy given by [33], saving $lb\, f(\cdot)$ evaluations. Including the root node in the BB tree, both trees use the same number of $lb\, f(\cdot)$ calls to reach the optimal solution without using any bound-based deletion. However, in practice, the number of $lb\, f(\cdot)$ calls for BB with "minimum-solution-tree" with in-level node ordering are much higher than BBCC due to bound-based deletions, as shown in Table 2. Further, BB works by deleting one predictor at each node to reach a feasible solution represented by a leaf node at the bottom level, whereas BBCC uses the idea of selecting different combinations of predictors to enumerate all feasible solutions, and there is no notion of level in the BBCC tree.

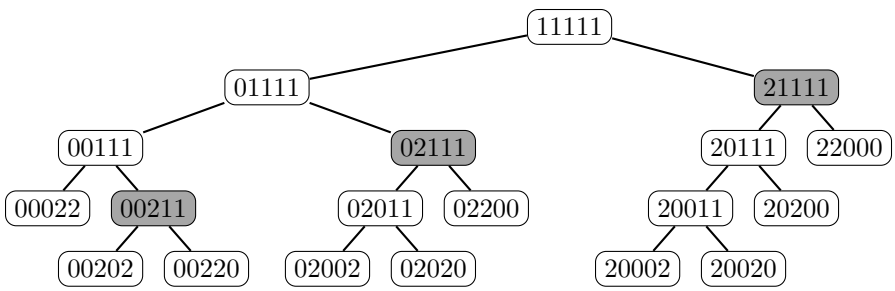

Fig. 1. BBCC tree for $p = 5$ and $k = 2$.

Remark 3.4. Assuming no bound-based deletions, BBCC and BB using "minimum-solution-tree" approach without an in-level node ordering procedure will take exactly $\binom{p+1}{k+1} - \binom{p-1}{k+1}$ number of $lb\,f(\cdot)$ calls to reach the optimal solution.

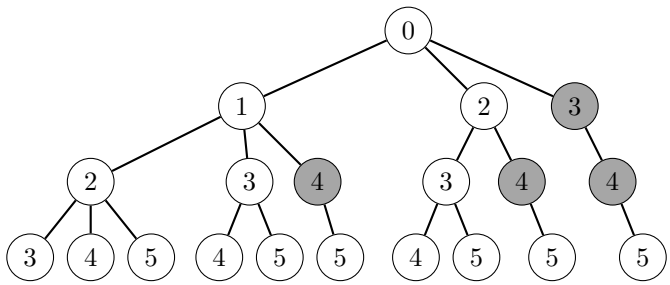

Fig. 2. BB tree for $p = 5$ and $k = 2$.

		$k = 10$		$k = 5$	
Example type	SNR	BBCC	BB	BBCC	BB
small-1	0.05	658	1036	378	1134
small-1	0.5	752	1148	381	1106
small-1	1	742	1139	367	1080
small-1	5	975	1554	254	773
small-2	0.05	39653	210179	7763	56131
small-2	0.5	39001	206137	7424	53632
small-2	1	38900	205324	7299	52504
small-2	5	39778	210140	6987	50729

Tab. 2. Number of OLS solutions needed to get to the optimal solution for BBCC and BB with in-level node ordering for OD examples of small-1 and small-2 type with $k \in \{5, 10\}$.

3.4. Solving the cardinality-constrained portfolio selection problem using BBCC

In this section, we discuss the main characteristics of Algorithm 2 that needs modification to solve the (CCPO) problem using BBCC. In particular, we need an efficient procedure to find a feasible point for (CCPO) and another procedure to find $lb\,f(\cdot)$ for any working box V.

3.4.1. Feasibility sampling

Because (BBSS) has only the CC constraint, so for any working box V in BBCC either it is infeasible (hence, we can discard it) or we can always find a feasible point with at least $p-k$ zero entries. However, for (CCPO) we have additional linear inequality and equality constraints along with the CC constraint, which makes finding a feasible point for a working box not that straightforward. Along with the infeasibility with respect to the CC, a box V is also infeasible if it satisfies the following deletion condition, where $J = \{j : V_j \neq 0, j = 1, \ldots, p\}$ is the set of indices corresponding to flag 1 or 2 of the box V.

$(E1, V)$ $\sum_{i \in I} u_i < 1$, where $I \subseteq J$ is the set of k indices with the largest magnitude of u array.

The deletion $E1$ comes from the fact that we will not be able to satisfy the equality constraint $e^T\beta = 1$ for an array with k non-zero entries if $E1$ is true for the box V. If $E1$ is not true for V, we cannot conclude that V is infeasible. In fact, we have to go over every possible set $I \subseteq J$ with k indices and check that $E1$ is not true for each set I, only then can we conclude that the box V is infeasible.

As the feasible set in (CCPO) is a polytope, after selecting the set I with k indices, we can use the Bounded Variables Simplex Method (BVSM) (see [6]) to find a feasible point by solving the following linear optimization problem.

$$\begin{aligned} \min_{x,s,t_1,t_2} \quad & t_1 + t_2 \\ \text{subject to} \quad & r_I^T x - s + t_1 = R, \\ & e^T x + t_2 = 1, \\ & 0 \leq x \leq u_I, \\ & 0 \leq s, \\ & 0 \leq t_1, \\ & 0 \leq t_2, \end{aligned} \tag{LP1}$$

where $x \in \mathbb{R}^k$, s is a surplus variable and t_1, t_2 are artificial variables. As $s = r_I^T x - R < |r_I^T x| \leq |r_I^T u_I|$, we have, $t_1 = R + s - r_I^T x \leq R + s + |r_I^T x| \leq R + 2|r_I^T u_I|$ and $t_2 = 1 - e^T x \leq 1$. Thus, we obtain the following modified problem to solve

$$\begin{aligned} \min_{x,s,t_1,t_2} \quad & t_1 + t_2 \\ \text{subject to} \quad & r_I^T x - s + t_1 = R, \\ & e^T x + t_2 = 1, \\ & 0 \leq x \leq u_I, \\ & 0 \leq s \leq |r_I^T u_I|, \\ & 0 \leq t_1 \leq R + 2|r_I^T u_I|, \\ & 0 \leq t_2 \leq 1. \end{aligned} \tag{LP2}$$

In contrast to the standard simplex method in linear optimization, the BVSM handles the bound constraints implicitly. Further, in the standard simplex method, nonbasic variables are those variables that are fixed at the lower bound. However, in BVSM, nonbasic variables are those that are fixed either at the upper or at the lower bound. If $\mathcal{B}$ and $\mathcal{N}$ represent the set of basic and nonbasic variables, respectively, then $\mathrm{x} = (x_{\mathcal{B}}, x_{\mathcal{N}})$. Furthermore, for the extended initial basic feasible solution that we need to provide to BVSM, the basic variables $x_{\mathcal{B}}$ must be within the bounds, and nonbasic variables $x_{\mathcal{N}}$ can be either at the upper or at the lower bounds. The extended initial basic feasible solution to solve the problem LP2, can be taken as $x = 0$, $s = 0$, $t_1 = R$, and $t_2 = 1$, where t_1 and t_2 are the basic variables and x, s are nonbasic. For a working box V, Algorithm 4 either finds a feasible point or concludes that the box is infeasible and can be discarded.

Algorithm 4: Finding a Feasible Point for (CCPO)

Input: k, u, r, R, V
Output: $\tilde{v}$, `isfeasible`

Set $J = \{j : V_j \neq 0, j = 1, \ldots, p\}$ and `isfeasible`=0.
Construct $\mathcal{I} = \{I_i\}$ containing sets of indices of all the possible combinations of choosing k indices from J.
for $i = 1$ **to** $|\mathcal{I}|$ **do**
    If $(E1, I_i)$ holds, go to the next iteration of i loop.
    Use BVSM to solve (LP2) in the reduced space determined by the set I_i to find the optimal solution $\mathrm{x}^* = (x^*, s^*, t_1^*, t_2^*)$.
    if $t_1^* + t_2^* > 0$ **then**
        The chosen support I_i is infeasible. Go to the next iteration of i loop.
    else
        Set $\tilde{v} = x^*$, `isfeasible`=1 and **return**.

3.4.2. Bounding

To find $lb\, f(\cdot)$ for a working box V in the reduced space determined by the set $J = \{j : V_j \neq 0, j = 1, \ldots, p\}$ with m indices, we have to solve the Constrained Quadratic Programming (CQP) problem.

$$\begin{aligned} \min_{x \in \mathbb{R}^m} \quad & x^T \Sigma_J \, x \\ \text{subject to} \quad & e^T x = 1, \;\; r_J^T x \geq R, \\ & 0 \leq x \leq u_J. \end{aligned} \tag{CQP}$$

Any effective procedure for solving (CQP) can be adopted as a bounding subroutine.

4. COMPUTATIONAL STUDY

In this section, we present the results of the proposed algorithm to solve the best subset selection problem and the cardinality-constrained portfolio optimization problem. We

compare the results with other state-of-the-art methods to solve both of these problems. We first describe the software and hardware setup in the next section.

4.1. Software and hardware setup

All the algorithms except ABESS have been implemented on `MATLAB` R2021a, and testing has been done on the University of Alabama High-Performance Computer. For the MIO approach, `GUROBI`([16]) has been called using the `MATLAB` interface for `GUROBI` 9.0.2. To test ABESS for solving (CCQO) we used the `abess` package developed in `R` (see [36]) by calling `R` through `MATLAB`.

4.2. Numerical results for the best subset selection problem

We provide numerical results comparing the following four algorithms to solve the (BBSS) problem.

BBCC The branch-and-bound algorithm for cardinality-constraint following the Algorithm 2;

BB The branch-and-bound algorithm following the implementation given in [29] with "minimum-solution-tree" strategy included as suggested in [33];

MIO The mixed integer optimization formulation solved by GUROBI.

ABESS The adaptive best subset selection method proposed in [35] and implemented in R.

As LASSO solves a convex penalized regression problem that acts as surrogate to the (BSS) problem, we do not include LASSO in our numerical results.

Implementation issues For the MIO approach to solve the (BBSS) problem, we followed the implementation given in [4] for setting up the MIO formulation. For $p \leq n$ and $p > n$ cases, we use two different MIO formulations given by equations 2.5 and 2.6 in [4], respectively. After converting (BBSS) to an equivalent MIO formulation, we use the `GUROBI` solver with its presolve feature turned off and restricting the number of threads to one for better comparison, leaving all the other parameters set to their default values. To test ABESS, while calling `abess` package in `R`, we set `tune.path`="sequence", `weight`=NULL, `normalize`=0, `fit.intercept`=FALSE, `beta.low`=-1, `beta.up`=1, and `support.size` $= \min(k, n, \lfloor \frac{n}{\log(p)\log(\log n)} \rfloor)$ where $[x]$ represents the integer part of x, leaving rest of the parameters set to default. Selection of a new box in step 8 at each iteration of Algorithm 2 is done using the BFS criteria. In addition, the coordinate direction η to partition the box Y to obtain child boxes (step 10) is done by choosing $\eta = \{i : \hat{x}_i = \max |\hat{x}|\}$, where $f(\hat{x}) = lb\, f(Y)$. To find a feasible point in step 15, we use the SFS algorithm and $lb\, f(\cdot)$ in step 16 is computed using a custom implementation of the parallel tangent method to solve convex quadratic optimization problems.

Choosing the initial domain The initial search domain B in the (BBSS) problem is not known a *priori*. If the chosen domain B is not big enough to contain the solution of (BSS), then the optimal solution of (BBSS) may not be the same as the optimal solution of (BSS). However, if we do not use B to find a solution to the (OLS) problem within BBCC and BB, we can solve (BSS) directly. While a bounded initial domain is necessary for MIO solver of (BSS), BBCC and BB remain applicable for solving (BSS) directly as long as (OLS) solver does not require a bounded search space. In particular, the parallel tangent method used in BBCC works better over the entire Euclidean space. In [4], the authors provided both theoretical and data-based approaches to choose the bound M used in their MIO formulation. In our testing, instead of using a uniform bound for β we use a non-uniform bound to define the search domain in (BBSS). We first find a solution $\hat{\beta}$ to (OLS) and define $m = \max(|\hat{\beta}_i| : i = 1, \ldots, p)$. Then the search domain $B = B_1 \times \ldots \times B_p$ is defined as

$$-\tau m - |\hat{\beta}_i| \leq B_i \leq |\hat{\beta}_i| + \tau m,$$

where $i \in \{1, \ldots, p\}$ and τ is an enlargement factor that has been set to $\tau = 1$. For a fair comparison, we use the same domain B for all four algorithms (i. e. whenever we have to find a solution to (OLS) problem at any step of these algorithms, we use this B).

4.2.1. Synthetic datasets

We present the computational experiments with synthetic datasets in this section.

Generating synthetic data We generate the synthetic datasets as follows. Firstly, we find the design matrix $X \in \mathbb{R}^{n \times p}$ by sampling each row (i.i.d.) from a p-dimensional multivariate normal distribution $N(0, \Sigma)$, with mean zero and covariance matrix Σ. We normalize the columns of X to have zero mean and unit ℓ_2-norm. We construct the true coefficient vector β^0. We choose a noise vector ε (i.i.d.) from the normal distribution $N(0, \sigma^2)$, where the variance σ^2 is chosen according to the given signal-to-noise (SNR) ratio defined as $\text{SNR} := \frac{\|X\beta^0\|_2^2}{\sigma^2}$. Finally, we get the response vector y using $y = X\beta^0 + \varepsilon$. Tab. 3 shows test examples we used in 3 different dimension regimes for the overdetermined (OD, $p < n$) case and the underdetermined (UD, $p > n$) case. We chose k_0 (the number of non-zero entries in β^0) to be 10, and the covariance matrix Σ is such that $\Sigma_{i,j} = 0.8$ when $i \neq j$ and $\Sigma_{i,i} = 1$. We want to select the best model with 5 and 10 predictors, i. e. $k \in \{5, 10\}$. We tested the following three examples by varying β^0.

Example 4.1. Generate β^0 such that $\beta_i^0 = 1$ for k_0 equally spaced indices from the set $\{1, \ldots, p\}$, rounding to the greatest integer if needed.

Example 4.2. Generate β^0 by assigning the first k_0 entries as 1, that is $\beta_i^0 = 1$ for $i = 1, \ldots, k_0$.

Example 4.3. Generate β^0 by picking a random subset from $\{1, \ldots, p\}$ of k_0 indices, and then we assign random integer values between 1 and 5 to those indexed variables.

Type	p	n for OD	n for UD
small-1	20	100	10
small-2	40	200	20
small-3	60	300	30
small-4	80	400	40
medium-1	200	1000	100
medium-2	300	1000	100
medium-3	400	2000	100
medium-4	500	2000	100
large-1	800	4000	200
large-2	1000	4000	200
large-3	1500	8000	300
large-4	2000	8000	300

Tab. 3. Test examples dimension setup.

We run the three examples in small, medium, and large dimensional settings as given in Tab. 3 with 4 SNR values SNR $\in \{0.05, 0.5, 1, 5\}$. We compare two aspects of these algorithms: solution quality and running time.

Solution quality To compare the solution quality, we use the Relative Gap % defined as

$$\left(\frac{\tilde{f} - f^*}{f^*}\right) 100,$$

where, for a particular example, f^* is the best objective function value found by any algorithm and $\tilde{f}$ is the objective function value from a given algorithm. A small Relative Gap % implies that the solution from a given algorithm is close to the best solution.

Figure 3 shows box plots of the Relative Gap % for the three examples in small, medium, and large dimension regimes in OD and UD cases. For each dimension regime we get 96 instances of (BBSS) corresponding to three examples, four SNR and two k values. The general trend is that MIO provides solutions with the lowest Relative Gap % in all the three dimension regimes we considered. In the OD case, for the small dimension examples, MIO provides the best solutions for all the instances. BBCC and BB are competitive with MIO, and ABESS performs the worst among all the algorithms. For almost every medium and large dimension example, MIO provides the highest quality solutions, followed closely by ABESS. BBCC performs worse than ABESS, and BB is not able to find the best solution in any of the medium and large-dimensional examples. In the UD case, the general trend is that MIO provides solutions with the smallest Relative Gap %, followed by BBCC. Between BB and ABESS, BB performs better than ABESS in the small dimension examples; however, BB performs the worst in the medium and large dimensional examples. Note that in the UD case, although BBCC is not able to provide the best solutions for all the examples, but it is competitive with MIO and shows less variance as compared to BB and ABESS.

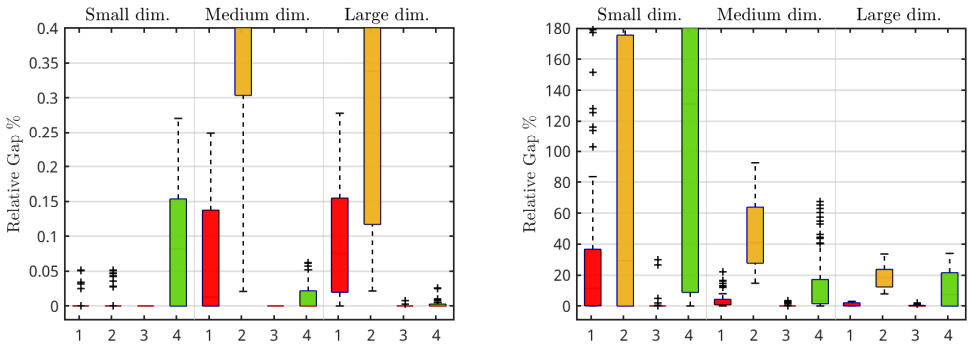

Fig. 3. Box plots of Relative Gap % for examples 1, 2, and 3 in small, medium, and large dimension regimes with 4 SNR values in OD (left) and UD (right) cases with $k \in \{5, 10\}$ for 1) BBCC; 2) BB; 3) MIO; 4) ABESS.

Running times It is well known that finding a globally optimal solution using an algorithm can take a lot of CPU time. Even if the algorithm finds the optimal solution quickly, the only way to certify its optimality is to go over the space of all the possible solutions. In our testing, we are using a hard CPU time limit of 10 minutes, which means that after 10 minutes of CPU time, we will stop the algorithm and report the current best solution as the final solution for that algorithm. Also, for BBCC and BB algorithms, we set the maximum iteration limit to be 10^6 and we stop the algorithm once this limit has been reached (reporting the current best solution as the final solution). We have incorporated two soft stops in BBCC along with these hard limits. We will stop the algorithm if $\tilde{f}$ in BBCC does not improve for 500 iterations or for 5 minutes of CPU time. With these soft stops included, BBCC can provide a solution close to optimal with much less CPU time.

Figure 4 shows the performance profiles of CPU time for the three examples in small, medium, and large dimension regimes in OD and UD cases. ABESS is the quickest among all the algorithms, taking only a few seconds even for large dimension examples. BBCC generally stops using a soft stopping criteria making it faster than MIO and BB. For the small dimension examples, MIO found the optimal solution within the given CPU time limit. In OD case, MIO stops prematurely for 67 out of 96 medium dimension instances, and all of the 96 large dimension instances by hitting the maximum CPU time limit provided. In UD case, MIO stops prematurely for 13 out of 96 medium instances and 18 out of 96 large instances using the maximum CPU time limit provided. BB is the slowest among all the algorithms and always stops after reaching maximum CPU time limit provided for both medium and large dimension examples, in both OD and UD cases.

Discussion Clearly, MIO consistently performs better compared to other algorithms in terms of the quality of solutions. In the OD case, for small dimension examples, the three exact methods performed better than ABESS; however, for the medium and large

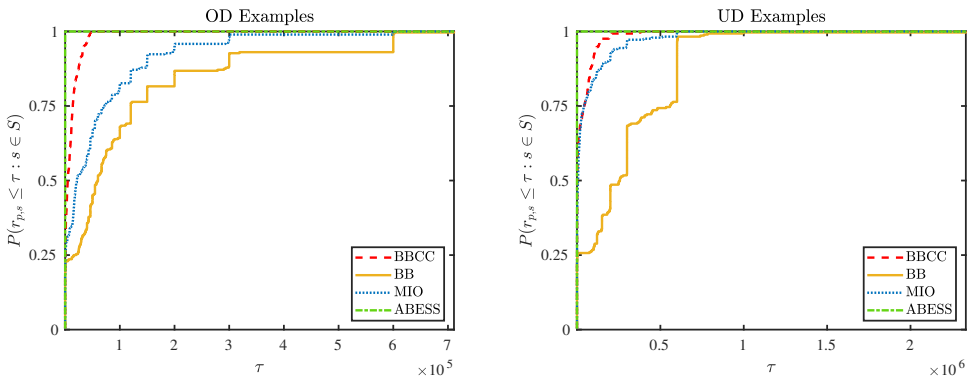

Fig. 4. Performance profiles of CPU time for examples 1, 2, and 3 in small, medium, and large dimension regimes with 4 SNR values and $k \in \{5, 10\}$.

dimension examples, the performance of BBCC and BB deteriorates as they suffered from premature termination. ABESS, being a polynomial-time algorithm, shines in the medium and large dimension examples, performing better than in the small dimension examples. UD examples are more computationally challenging than OD examples because of the greater randomness in the data and the lower chance of bound-based deletions for every branch-and-bound algorithm. In the UD case, BBCC is competitive with MIO, providing high-quality solutions with less CPU time, and outperforms both BB and ABESS. The performance of ABESS is the worst for the UD examples. Note that in the UD case, due to the special formulation, the dimension of the problem to solve by MIO is $\min(p, n)$, which gives MIO an additional advantage over BBCC and BB algorithms.

4.2.2. Real datasets

In this section, we consider two real datasets to compare the performance of the given algorithms.

Ozone dataset We used the Los Angeles Ozone dataset (downloaded from `https://hastie.su.domains/ElemStatLearn/`), which consists of 8 meteorological variables measured on 330 days in 1976 in Los Angeles. The response variable is the maximum hourly average ozone concentration. The first 8 variables are:
X_1 : 500 mb height
X_2 : wind speed
X_3 : humidity %
X_4 : surface temperature
X_5 : inversion height
X_6 : pressure gradient
X_7 : inversion temperature

X_8 : visibility
The remaining 36 variables have been generated as X_1^2, X_1X_2, X_2^2, X_1X_3, X_2X_3, X_3^2, $\cdots$, X_8^2. The design matrix X and response y have been normalized to have zero mean and unit ℓ_2-norm. This results in a model with 44 predictors and 330 responses.

	$f(\beta)$		CPU time	
k	BBCC	MIO	BBCC	MIO
10	0.2235	0.2225	>**0.2228**	1.9342
9	0.2257	0.2243	>**0.1645**	1.3842
8	0.2267	0.2262	>**0.1192**	1.0170
7	0.2306	0.2288	>**0.0795**	>0.2853
6	0.2322	0.2322	>**0.0565**	0.1255
5	0.2385	0.2385	>**0.0552**	0.0750
4	0.2441	0.2440	>0.0427	**0.0370**
3	0.2577	0.2577	>0.0333	**0.0200**
2	0.2715	0.2715	>0.0227	**0.0052**
1	0.3091	0.3091	**0.0013**	0.0030

Tab. 4. $f(\beta)$ and CPU time in minutes for ozone data set with $k \in \{1, \ldots, 10\}$.

Table 4 shows the test results for BBCC and MIO to solve the (BBSS) problem with k ranging from 1 to 10. The symbol > before the CPU time indicates that the algorithm stopped prematurely due to the built-in stopping criteria. MIO found the optimal solution for all the but one instances, with the solutions provided by BBCC following closely. Using our customized sampling procedure to solve the BSS and the soft stops, BBCC outperforms MIO in 7 out of the total 10 instances tested in terms of CPU time.

Leukemia dataset with synthetic continuous response To demonstrate the applicability of the proposed algorithm for high-dimensional data, we considered the leukemia dataset (downloaded from the accompanying supplementary material to [12]). This dataset consists of 3571 predictors and 72 observations with a binary response y, classifying a given sample into *ALL* or *AML* type leukemia.

We generate a synthetic continuous response y by choosing β^0 as in example 2 with $k_0 = 10$. After choosing ε from $N(0, \sigma^2)$ where σ^2 is chosen for a given SNR, we find $y = X\beta^0 + \varepsilon$. Table 5 shows the average CPU time for BBCC and MIO to solve the (BBSS) problem with k ranging from 1 to 10 by generating y using 4 SNR values. MIO solving the (BBSS) in dimension n using the special formulation for $p \gg n$ case, provided the optimal solution for every instance in approximately one minute. On the other hand, BBCC also found the optimal solution for every instance, but stopped due to the soft stopping criterias incorported within the algorithm without providing the certificate of optimality.

k	BBCC	MIO
10	>2.8167	**0.6061**
9	>2.7557	**0.7829**
8	>2.6810	**0.7112**
7	>2.5924	**0.7312**
6	>2.7074	**0.4848**
5	>2.4945	**1.0940**
4	>2.4691	**1.0535**
3	>2.4288	**1.0106**
2	>2.4013	**1.5431**
1	>2.2713	**0.8654**

Tab. 5. Average CPU time in minutes for leukemia data set with 4 SNR values and $k \in \{1, \ldots, 10\}$.

4.3. Numerical results for cardinality-constrained portfolio optimization problem

In this section, we present numerical results to solve the (CCPO) problem for the randomly generated instances along with a real dataset. We test the following two algorithms.

BBCC The branch-and-bound algorithm for cardinality constraint following Algorithm 2 with appropriate modifications as discussed in section 3.4;

MIO The mixed integer portfolio optimization problem solved by GUROBI.

Implementation issues To solve (CCPO) using the MIO approach, we construct a model based on (MIPO) and provide it to `GUROBI` solver. Algorithm 2 with the appropriate modifications as discussed in section 3.4 is used to solve (CCPO) directly. The initial feasible point in step 2 is computed following the procedure in Algorithm 4 and using a custom BVSM subroutine to solve (LP2). Note that in Algorithm 4, for a working box V, if m is the number of indices in the set J, the number of possible sets $I_i \subseteq J$ with k indices is $\binom{m}{k}$, which can be a large number even for small m and k. Hence, in our implementation, to find a feasible point for a working box V using Algorithm 4, we check only one set I_1 of k indices corresponding to the largest magnitude of u. If we cannot find a feasible point corresponding to the index set I_1, we skip the checking of bound-based deletion and add the box to the list for further exploration. The $lb\ f(\cdot)$ in step 16 of Algorithm 2 is computed using `quadprog` solver in `MATLAB` in the reduced space determined by the working box V, with all the parameters set to their default values. We tested both algorithms with a maximum CPU time limit of 15 minutes to solve an instance. For BBCC we provide a maximum iteration limit of $2 \cdot 10^6$ and set a soft stop by terminating the algorithm prematurely if $\tilde{f}$ in Algorithm 2 does not improve for 10^5 iterations. Our early testing results revealed that the DFS selection strategy to select a new box in step 8 of Algorithm 2 works best to solve (CCPO) using BBCC. So

we use DFS to select a new box from the list to process further in BBCC. The selection of partition direction η is done the same way as in (BBSS) problem.

Generating the test sets We consider the five mean-variance portfolio dataset based on stocks from five major capital market indices worldwide. Weekly price data for these stocks were obtained from DATASTREAM, covering the period from March 1992 to September 1997, including the Hang Seng (Hong Kong), DAX 100 (Germany), FTSE 100 (UK), S&P 100 (USA), and Nikkei 225 (Japan). The five dataset are publicly available in `OR-Library` using [2]. We select the expected portfolio return $R > 0$ following the idea used in [11], which is to take $R = R_{\min} + 0.3(R_{\max} - R_{\min})$ with $R_{\min} = r^T x^*$, $R_{\max} = r^T \bar{x}$ where

$$x^* = \arg\min_x \{x^T \Sigma\, x : e^T x = 1, x \geq 0\}$$

$$\bar{x} = \arg\max_x \{r^T x : e^T x = 1, x \geq 0\}.$$

The upper bound u has been sampled from the Gaussian distribution $N(0.5, 0.01)$ with mean 0.5 and variance 0.01. To further test our method on synthetic random data, we follow the procedure given in [19]. We generate random covariance matrices Σ and the mean return vector r following the `Python` implementation given in [28]. The expected return R and upper bound u are computed as described above.

Discussion We first discuss the results to solve (CCPO) with $k \in \{1, 2, 3, 4, 5\}$ for the five real datasets. Table 6 shows the objective function values and CPU time comparing BBCC and MIO. For all 25 instances, both BBCC and MIO found the optimal solutions for the given stopping parameters or marked the problem infeasible. For 15 out of 20 feasible instances, BBCC stopped after processing all the boxes, providing the guarantee of the solutions being optimal. In 5 of the instances, BBCC stopped prematurely, returning the current best solution as the final solution. On the other hand, for most of the instances, MIO finds the optimal solution within less than a minute.

Table 7 shows the results for synthetic random test problems with $p \in \{20, 40, 60, 80\}$ and $k \in \{1, 2, 3, 4, 5\}$. Both methods provide the optimal solutions for all 20 instances tested or conclude that the problem is infeasible. BBCC stopped prematurely for 4 instances, returning the current best solution as the final output. Like every global solver, BBCC is able to find the optimal solution quickly and spends much of the CPU time to provide the certificate of optimality for those solutions. In terms of CPU time, MIO is clearly faster. However, for a few instances with $k = 2$, BBCC is faster than MIO.

5. CONCLUSION

In this paper, we introduced a special branch-and-bound algorithm to solve the cardinality-constrained quadratic optimization problem. The proposed algorithm finds a globally optimal solution using a special enumeration to go over all the possible subsets and discard those that cannot contain an optimal solution using certain deletion conditions. The numerical results show that our algorithm scales well to solve the best subset selection problem with dimensions in the 1000s and can solve the cardinality-constrained

			$f(\beta)$		CPU time	
Instance	p	k	BBCC	MIO	BBCC	MIO
port1	31	5	0.3967	0.3967	>0.5412	**0.0168**
		4	0.4046	0.4046	0.1575	**0.0107**
		3	0.4466	0.4466	**0.0178**	0.0205
		2	0.6570	0.6570	**0.0057**	0.0183
		1	Infeasible	Infeasible	**0.0002**	0.0025
port2	85	5	0.2133	0.2133	3.0698	**0.6042**
		4	0.2528	0.2528	4.4125	**0.3898**
		3	0.3301	0.3301	0.7205	**0.1402**
		2	0.4935	0.4935	**0.0243**	0.0398
		1	Infeasible	Infeasible	**0.0003**	0.0027
port3	89	5	0.2732	0.2732	2.0917	**0.5232**
		4	0.3075	0.3075	1.1273	**0.4433**
		3	0.3625	0.3625	0.3412	**0.1288**
		2	0.5091	0.5091	**0.0383**	0.0435
		1	Infeasible	Infeasible	**0.0002**	0.0027
port4	98	5	0.2045	0.2045	>12.1155	**4.1295**
		4	0.2291	0.2291	>10.7818	**1.1103**
		3	0.2933	0.2933	1.7493	**0.5610**
		2	0.4842	0.4842	**0.1243**	0.1260
		1	Infeasible	Infeasible	**0.0003**	0.0027
port5	225	5	0.3835	0.3835	>15	**0.5048**
		4	0.4108	0.4108	>15	**0.5688**
		3	0.4541	0.4541	10.7118	**0.6033**
		2	0.5459	0.5459	1.2103	**0.2807**
		1	Infeasible	Infeasible	**0.0008**	0.0032

Tab. 6. $f(\beta)$ and CPU time in minutes to solve CCPO problem for the five real dataset with $k \in \{1, 2, 3, 4, 5\}$.

portfolio optimization problem with dimensions in the 100s, in a practical amount of time. To solve the cardinality-constrained quadratic optimization problem, we compared our algorithm with other state-of-the-art solvers. The numerical results show that the proposed algorithm is at least competitive with other solvers. It has been demonstrated that our algorithm can handle additional linear constraints in addition to a cardinality-constraint. The algorithm has the desired property of finding high-quality candidate solutions early on in the process, so that even if it gets terminated prematurely, we still get a near-optimal solution. We can further include more acceleration strategies within our algorithm to make it faster while keeping the global convergence of the algorithm. The proposed algorithm does not need the convexity of the objective function as an assumption, as long as we are using tight lower bounds within the algorithm.

		$f(\beta)$		CPU time	
p	k	BBCC	MIO	BBCC	MIO
20	5	0.00 2136	0.00 2136	0.2732	**0.0503**
	4	0.00 2182	0.00 2182	0.1270	**0.0373**
	3	0.00 2261	0.00 2261	**0.0342**	0.0508
	2	0.00 2432	0.00 2432	**0.0037**	0.0275
	1	Infeasible	Infeasible	**0.0002**	0.0028
40	5	0.00 2084	0.00 2084	1.8985	**0.3935**
	4	0.00 2130	0.00 2130	0.6958	**0.1227**
	3	0.00 2210	0.00 2210	0.1588	**0.0613**
	2	0.00 2383	0.00 2383	**0.0138**	0.0468
	1	Infeasible	Infeasible	**0.0002**	0.0025
60	5	0.00 2102	0.00 2102	>1.9830	**1.5723**
	4	0.00 2143	0.00 2143	>2.6330	**0.5738**
	3	0.00 2210	0.00 2210	0.4175	**0.2823**
	2	0.00 2403	0.00 2403	**0.0275**	0.0517
	1	Infeasible	Infeasible	**0.0000**	0.0023
80	5	0.00 2088	0.00 2088	>**8.9858**	>15
	4	0.00 2133	0.00 2133	>7.9843	**2.9293**
	3	0.00 2214	0.00 2214	1.8727	**0.5947**
	2	0.00 2377	0.00 2377	0.0942	**0.0783**
	1	Infeasible	Infeasible	**0.0000**	0.0025

Tab. 7. $f(\beta)$ and CPU time in minutes to solve CCPO problem with $p \in \{20, 40, 60, 80\}$ and $k \in \{1, 2, 3, 4, 5\}$.

ACKNOWLEDGEMENT

We would like to thank Ward Jaeger for his contribution in implementing the heap data structure and anonymous reviewers for their valuable comments that improved this manuscript. We also thank The University of Alabama and the Office of Information Technology for providing high-performance computing resources and support that have contributed to these research results.

(Received April 18, 2025)

REFERENCES

[1] A. d'Aspremont, F. Bach, and L. El Ghaoui: Optimal solutions for sparse principal component analysis. J. Mach. Learn. Res. *9* (2008), 7, 1269–1294.

[2] J. E. Beasley: OR-Library: distributing test problems by electronic mail. J. Oper. Res. Soc. *41* (1990), 11, 1069–1072. DOI:10.1057/jors.1990.166

[3] D. Bertsimas and R. Shioda: Algorithm for cardinality-constrained quadratic optimization. Comput. Optim. Appl. *43* (2009), 1, 1–22. DOI:10.1007/s10589-007-9126-9

[4] D. Bertsimas, A. King, and R. Mazumder: Best subset selection via a modern optimization lens. Ann. Statist. *44* (2015), 2, 813–852. DOI:10.1214/15-AOS1388

[5] D. Bienstock: Computational study of a family of mixed-integer quadratic programming problems. Math. Program. *74* (1996), 121–140. DOI:10.1007/BF02592208

[6] R. E. Bixby: Implementing the simplex method: the initial basis. ORSA J. Comput. *4* (1992), 3, 267–284. DOI:10.1287/ijoc.4.3.267

[7] T. Blumensath and M. E. Davies: Iterative thresholding for sparse approximations. J. Fourier Anal. Appl. *14* (2008), 629–654. DOI:10.1007/s00041-008-9035-z

[8] P. Bonami and M. A. Lejeune: An exact solution approach for portfolio optimization problems under stochastic and integer constraints. Oper. Res. *57* (2009), 3, 650–670.

[9] M. Branda, M. Bucher, M. Červinka, and A. Schwartz: Convergence of a Scholtes-type regularization method for cardinality-constrained optimization problems with an application in sparse robust portfolio optimization. Comput. Optim. Appl. *70* (2018), 2, 503–530. DOI:10.1007/s10589-018-9985-2

[10] P. Bühlmann and S. van de Geer: Statistics for High-Dimensional Data: Methods, Theory and Applications. Springer–Verlag, Berlin 2011. DOI:10.1007/978-3-642-20192-9

[11] C. Moreira, D. Kreber, and M. Schmidt: An alternating method for cardinality-constrained optimization: a computational study for the best subset selection and sparse portfolio problems. INFORMS J. Comput. *34* (2022), 6, 2968–2988. DOI:10.1287/ijoc.2022.1211

[12] J. Friedman, T. Hastie, and R. Tibshirani: Regularization paths for generalized linear models via coordinate descent. J. Stat. Softw. *33* (2010), 1, 1–22. DOI:10.18637/jss.v033.i01

[13] K. Fukunaga: Introduction to Statistical Pattern Recognition. Second edition. Academic Press, Inc., San Diego 1990.

[14] J. Gao and D. Li: Optimal cardinality constrained portfolio selection. Oper. Res. *61* (2013), 3, 745–761. DOI:10.1287/opre.2013.1170

[15] C. Gatu and E. J. Kontoghiorghes: Branch-and-bound algorithms for computing the best-subset regression models. J. Comput. Graph. Statist. *15* (2006), 1, 139–156. DOI:10.1198/106186006X100290

[16] Gurobi Optimization, LLC: Gurobi optimizer reference manual. 2023.

[17] H. Eldon: Global optimization using interval analysis—the multi-dimensional case. Numer. Math. *34* (1980), 3, 247–270.

[18] T. Hastie, R. Tibshirani, and R. J. Tibshirani: Extended comparisons of best subset selection, forward stepwise selection, and the lasso. (2017) arXiv:1707.08692.

[19] M. Hirschberger, Y. Qi, and R. E. Steuer: Randomly generating portfolio-selection covariance matrices with specified distributional characteristics. European J. Oper. Res. *177* (2007), 3, 1610–1625. DOI:10.1016/j.ejor.2005.10.014

[20] Y. Jin, R. Qu, and J. Atkin: Constrained portfolio optimisation: the state-of-the-art Markowitz models. In: Proc. 5th International Conference on Operations Research and Enterprise Systems – ICORES (2016), pp. 388–395. DOI:10.5220/0005758303880395

[21] H. Markowitz: Portfolio Selection*. J. Finance *7* (1952), 1, 77–91. DOI:10.1111/j.1540-6261.1952.tb01525.x

[22] A. Miller: Subset Selection in Regression. Second edition. Chapman and Hall/CRC, New York 2002.

[23] R. E. Moore: Interval Analysis. Prentice-Hall, Englewood Cliffs, New Jersey 1966.

[24] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell: Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optim. *19* (2016), 79–102. DOI:10.1016/j.disopt.2016.01.005

[25] P. M. Narendra and K. Fukunaga: A branch and bound algorithm for feature subset selection. IEEE Trans. Comput. *26* (1977), 9, 917–922.

[26] B. K. Natarajan: Sparse approximate solutions to linear systems. SIAM J. Comput. *24* (1995), 2, 227–234. DOI:10.1137/S0097539792240406

[27] H. Ratschek and J. Rokne: New Computer Methods for Global Optimization. Halsted Press, Chichester 1988.

[28] S. B. Çay: Random Portfolio Dataset Generator. DOI:10.5281/zenodo.53204

[29] P. Somol, P. Pudil, and J. Kittler: Fast branch & bound algorithms for optimal feature selection. IEEE Trans. Patt. Anal. Machine Intell. *26* (2004), 7, 900–912. DOI:10.1109/tpami.2004.28

[30] R. Tibshirani: Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B. Stat. Methodol. *58* (1996), 1, 267–288. DOI:10.1111/j.2517-6161.1996.tb02080.x

[31] A. M. Tillman, D. Bienstock, A. Lodi, and A. Schwartz: Cardinality minimization, constraints, and regularization: a survey. SIAM Rev. *66* (2024), 3, 403–477. DOI:10.1137/21M142770X

[32] D. S. Watkins: Fundamentals of Matrix Computations. Third edition. John Wiley and Sons, Hoboken, New Jersey 2010.

[33] B. Yu and B. Yuan: A more efficient branch and bound algorithm for feature selection. Pattern Recognit. *26* (1993), 6, 883–889. DOI:10.1016/0031-3203(93)90054-Z

[34] Y. Zhao and X. Huo: A survey of numerical algorithms that can solve the Lasso problems. Wiley Interdiscip. Rev. Comput. Stat. *15* (2023), 4, e1602. DOI:10.1002/wics.1602

[35] J. Zhu, C. Wen, J. Zhu, H. Zhang, and X. Wang: A polynomial algorithm for best-subset selection problem. Proc. Natl. Acad. Sci. USA *117* (2020), 52, 33117–33123. DOI:10.1073/pnas.2014241117

[36] J. Zhu, X. Wang, L. Hu, J. Huang, K. Jiang,Y. Zhang, S. Lin, and J. Zhu: abess: A fast best-subset selection library in python and R. J. Mach. Learn. Res. *23* (2022), 202, 1–7. DOI:10.32614/cran.package.abess

Vikram Singh, Department of Mathematics, Texas A&M University, Texarkana, Texas, 75503. United States.
e-mail: vsingh@tamut.edu

Min Sun, Department of Mathematics, The University of Alabama, Tuscaloosa, Alabama, 35487. United States.
e-mail: msun@ua.edu