Branch-and-Bound Algorithms for Variable Selection and Shortest Vector Problem

木村, 圭児

KYUSHU UNIVERSITY

Doctoral Thesis

Branch-and-Bound Algorithms for Variable Selection and Shortest Vector Problem

Keiji Kimura

A thesis submitted in fulfillment of the requirements
for the degree of Fuctional Mathematics

in the

Graduate School of Mathematics

February 15, 2019

Abstract

Overview: The availability and maturity of software for solving mixed integer nonlinear (MINLP) problems have increased significantly in the past 20 years. State-of-the-art optimization software, especially, is available for many optimization problems, and it is widely used by researchers and practitioners. However, the computational costs of MINLP problems are much higher than that of mixed integer linear problems even though such software is used. In addition, it does not make use of every property of a given optimization problem because it is regarded as general-purpose optimization software. On the other hand, an MINLP solver purpose-build for a specific problem occasionally achieves good computational performance. Herein, we propose effective branch-and-bound algorithms to solve two problems: variable selection and shortest vector problems. A branch-and-bound algorithm is a design paradigm for discrete and combinatorial optimization problems, and it consists of some components, such as relaxation, branching, and heuristic methods. We take advantage of the structure and properties of the problems to improve these components in terms of computational performance. We show numerical experiments pertaining to the proposed algorithms and examine how they outperform state-of-the-art optimization software.

Variable selection: Finding the best statistical model for a given dataset is one of the most important problems in statistical applications (e.g., linear and logistic regression). This problem is called variable selection, and solving it leads to various benefits. We focus on direct objective optimization which computes the best subsets of variables in terms of a given objective function. Computation of all candidates is impractical because the number of them is exponentially large. We formulate this optimization as an MINLP problem, which corresponds to ℓ_0-penalized variable selection. To solve it efficiently, we develop some components related to a branch-and-bound algorithm, for example, relaxation, a heuristic method, and branching rules. Moreover, we explain the application of the MINLP problem.

Shortest vector problem: The security of lattice-based cryptosystems is based on the hardness of problems on lattices. One of the most studied problems on lattices is the shortest vector problem (SVP), which searches for the shortest non-zero vector in a given lattice. This problem is known to be NP-hard, and it has been generally researched in the field of cryptology. Herein, we apply mathematical optimization to SVP. From the point of view of optimization software, SVP contains an intractable constraint, and it cannot directly be solved via optimization software. To handle the constraint, we develop ingenious branching and relaxation. In addition, we propose a heuristic method to find feasible solutions of subproblems. We examine computational performance of the proposed branch-and-bound algorithm. To this end, we propose two convex integer quadratic programming approaches using state-of-state-of-the-art optimization software.

Keywords: Mixed integer nonlinear programming, branch-and-bound algorithm, variable selection, shortest vector problem.

Contents

Chapter 1

Introduction

1.1 Mixed integer nonlinear programming

Mixed integer nonlinear programming (MINLP) can deal with integer variables and nonlinear functions and is the most flexible optimization from the viewpoint of formulation. Such optimization problems arise in many real world applications. However, this flexibility leads to numerical difficulties associated with the handling of nonlinear functions and challenges pertaining to optimization in the context of integrality. Nonetheless, many researchers and practitioners have shown interest in solving MINLP problems.

MINLP problems generally form a class of challenging optimization problems because they combine the difficulty of optimizing over integer variables with the handling of nonlinear functions. Even though an MINLP problem contains only linear functions, it becomes a mixed integer linear programming (MILP) problem, which is NP-hard [29]. In this study, we deal with convex MINLP problems, which contain convex functions in the objective function and the constraints. A convex MINLP problem are still NP-hard because it contains an MILP problem as a special case. Nevertheless, a convex MINLP problem can be solved much more efficiently than a general MINLP problem.

Several algorithms have been proposed for solving MINLP problems, for example, branch-and-bound algorithm, branch-and-cut algorithm, outer approximation, and Benders decomposition. See [15, 43, 56] for details about these algorithms. In this study, we employ a branch-and-bound algorithm. The idea of this algorithm is to successively divide a given problem into smaller subproblems until each of the subproblems is solvable. This algorithm is widely used to solve optimization problems, and it consists of some components, such as relaxation, branching, heuristics, and presolve. Chapter 2 summarizes this algorithm.

The availability and maturity of software for solving MINLP problems have increased significantly in the past 20 years. A number of open sources and commercial MINLP solvers are listed in [24]. Notably, state-of-the-art solvers, such as CPLEX [4], Gurobi [3], and FICO Xpress-Optimizer [1], are widely used. These are available for many optimization problems, for example, MILP, mixed integer quadratic programming (MIQP), and mixed integer second-order cone programming (MISOCP) problems. In other words, it is regarded as general-purpose optimization software. In this study, to examine computational performance of our branch-and-bound algorithms, we compare them with conventional algorithms employing CPLEX.

A customized MINLP solver for a specific application occasionally achieves good computational performance [21, 27]. In Chapter 3, we implement a few proposed techniques of a branch-and-bound algorithm by customizing the SCIP Optimization Suite [5]. This software toolbox comprises several parts, such as SCIP [9, 56] and UG [7]. SCIP is open source software, and it provides a branch-and-bound framework for solving MILP and MINLP problems. Additional plugins, such as branching rules, relaxation handlers, and heuristic methods, allow for an efficient solution process. UG provides a parallel extension of SCIP to employ multi-threaded parallel computation. These software applications have been developed by the Optimization Department at the Zuse Institute Berlin and its collaborators.

1.2 Variable selection

Finding the best statistical model for a given dataset is one of the most important problems in statistical applications (e.g., linear and logistic regression). This problem is called *variable selection,* and solving it leads to the following benefits: improvement in the prediction performance of a statistical model, development of faster and more cost-effective models in terms of computation, and better understanding of the essence of the statistical model behind a given dataset.

Methods for variable selection are essentially grouped into three methods: filter, wrapper, and embedded methods. Filter methods select subsets of variables independently to each other. For example, Torkkola [54] proposed a filter method using a mutual information criterion. The greatest advantage of filter methods is effective in computational time. However, filter methods tend to select redundant variables because they do not consider the relationships between variables. Hence, they are mainly employed as a pre-processing step. Wrapper methods, such as stepwise methods with forward selection and backward elimination, evaluate subsets of variables and consider the interaction between variables. The stepwise methods are implemented in standard statistical software (e.g., R [53]). In each step, the stepwise methods decide whether to add one variable to a statistical model or to remove it and evaluate the statistical model comprised of the selected variables. This process is repeated until no further improvement is possible. Although the stepwise methods are considered greedy algorithms, they often find good statistical models within a short time. Each embedded method performs its particular process and provides good statistical models as a result of the process, for example, ℓ_1-penalized regularization, and decision tree learning. See [31] for more details about variable selection.

In Chapter 3, we focus on *direct objective optimization*, which is a family of embedded methods. An objective function typically consists of two competing terms: the goodness-of-fit and the number of variables. For example, the Akaike information criterion (AIC) [12] and Bayesian information criterion (BIC) [50] are of the form of such an objective function. Direct objective optimization computes the best subsets of variables in terms of a given objective function. However, computation of all candidates is impractical because the number of them is exponentially large. Notably, most direct objective optimization under linear regression is known to be NP-hard [33].

Showing numerical experiments pertaining to linear and logistic regression, we briefly introduce work related to direct objective optimization for them as follows. ℓ_1-penalized linear and logistic regression [35, 41] is often employed because it provides sparse models and performs well even on large-scale instances. However, the models provided by this approach are not necessarily the best in terms of goodness-of-fit measures. Miyashiro and Takano proposed an MISOCP approach for variable selection based on some information criteria in linear regression [47]. This approach formulate the variable selection as an MISOCP problem, and standard optimization software such as CPLEX [4] and Gurobi [3], are available for the MISOCP problem. Bertsimas et al. [17] presented an MILP approach for variable selection in linear regression and developed a discrete extension of modern first order continuous optimization methods to find high quality feasible solutions. They used the feasible solutions as warm starts of optimization software. Sato et al. [48] formulated an MILP problem by employing a piecewise linear approximation to minimize goodness-of-fit measures for logistic regression. Although this approach might not arrive at the best logistic model, the results of their computational experiments indicated that this approach outperformed stepwise methods. Bertsimas and King [16] proposed an MINLP approach to constructing logistic models with the desired properties, for example, predictive power, interpretability, and sparsity. In addition, they proposed a tailored methodology using outer approximation techniques and dynamic constraint generation to solve the MINLP problem. The risk score problem [55] was optimized for feature selection, integer coefficient, and operational constraints. This problem was formulated as an MINLP problem that can be solved by using the cutting plane algorithm proposed in [55].

1.3 Shortest vector problem

An n-dimensional lattice is the set of all integral linear combinations of n linearly independent vectors in $\mathbb{R}^n$. Lattices have been known in number theory since the eighteenth century, and they already appeared when Lagrange, Gauss, and Hermite studied quadratic forms. Nowadays, lattices are widely used in the field of cryptology. One of the main problems in lattices is the *shortest vector problem* (SVP), which is searching for the shortest non-zero vector in a given lattice. SVP is known to be NP-hard under randomized reductions [11]. The security of lattice-based cryptosystems is based on the hardness of SVPs.

The fastest algorithm for solving SVP is the enumeration algorithm presented by Kannan [34] and by Fincke and Pohst [28]. Kannan's algorithm solves SVP in $2^{\mathcal{O}(n \log n)}$, where n is the lattice dimension. The variant used mostly in practice was presented by Schnorr and Euchner [49]. However, this algorithm also runs in exponential runtime. Roughly speaking, these enumeration algorithms perform a depth first search in a search tree that contains all vectors in a certain search space. So far, enumeration algorithms is applicable to SVPs with low dimensional lattice ($n \lesssim 60$). See the recent work of [25, 32] for details about limitations. To gain speed-up, this enumeration algorithm was parallelized [25, 32]. These parallel enumeration algorithms were faster than the then best public implementation.

The LLL algorithm [44] is the first algorithm with a polynomial runtime, although this algorithm searches for shorter vectors rather than the shortest vector. The LLL algorithm can be run in lattice dimension n up to 1000. In practice, the most promising algorithm for SVPs with high dimensional lattice is the BKZ algorithm [49], which mainly consists of two parts: an enumeration algorithm in low dimension and the LLL algorithm in high dimension. The BKZ algorithm finds shorter vectors than LLL algorithm and needs higher computational time. The LLL and BKZ algorithms transform a given lattice to a tractable lattice which consists of shorter basis. Enumeration algorithms is always executed on lattices to which either the LLL or BKZ algorithm is applied in a preprocessing step because these algorithms reduce the runtime significantly compared to the original lattice. The LLL and BKZ algorithms have been improved since these are developed (see, e.g., [46, 52]). The fplll [2] library includes algorithms to search for the shortest or shorter vectors, for example, the LLL, BKZ, and

Kannan's algorithm.

1.4 Contribution

Standard optimization software is convenient because of availability for many types of optimization problems (e.g., MILP and MIQP problems). In fact, state-of-the-art software, such as CPLEX [4] and Gurobi [3], is often used by many researchers and practitioners. Formulating a practical problem as an optimization problem which optimization software can handle, is one of the approaches to solving practical problems. Notably, state-of-the-art software provides good computational performance for MILP problems. However, a practical problem cannot always be formulated as an MILP problem. Even if an MINLP problem is reformulated as an MILP problem, the MILP problem may be poorly designed, and we may not achieve good computational performance. Generally, the computational costs of MINLP problems are much higher than that of MILP problems even though state-of-the-art software is employed. One of the techniques for solving MINLP problems is taking advantage of properties unique to a given optimization problem. Unfortunately, most optimization software does not make use of every such property because it is for solving general optimization problems, such as MILP, MIQP, and MISOCP problems. On the other hand, a customized MINLP solver for a specific problem occasionally achieves good computational performance by using properties and the structure of problems (see, e.g., [21, 27]).

Herein, we focus on two kinds of problems: variable selection and shortest vector problems. We propose an effective branch-and-bound algorithm for each of the problems. A branch-and-bound algorithm consists of some components, such as relaxation, branching, and heuristic methods. We take advantage of the structure and properties of the problems to improve these components in terms of computational performance. The results of computational experiments show how the proposed algorithms outperform state-of-the-art optimization software.

In Chapter 3, we focus on direct objective optimization in variable selection and formulate it as an MINLP problem. To solve the MINLP problem, we propose an effective branch-and-bound algorithm with a few techniques, for example, cost-effective relaxation, a heuristic method based on stepwise methods, and branching rules. These techniques are based on our previous papers [36, 37, 38]. We show that applications of the formulated MINLP problem include AIC-based variable selection in linear and logistic regression. Therefore, the proposed branch-and-bound algorithm can be applied to these applications. In numerical experiments pertaining to these applications, we show that, for small-scale and medium-scale instances, our solver is faster than conventional approaches using state-of-the-art optimization software. Conversely, for large-scale instances, our solver find better solutions compared to other approaches even if it cannot determine the optimal value within a realistic time.

In Chapter 4, we examine how large SVPs can be solved by applying integer quadratic programming (IQP). To this end, we propose an effective branch-and-bound algorithm for SVPs. This algorithm consists of presolve, branching, relaxation, and a heuristic method. In [40], we proposed the presolve to compute upper and lower bounds on variables. Herein, we propose the other components purpose-built for SVPs. In addition, we propose two convex IQP approaches using high standard optimization software (e.g., CPLEX [4]). We compare the proposed branch-and-bound algorithm with the two convex IQP approaches and shows that the branch-and-bound outperforms the other in terms of computational time.

The remainder of this paper is organized as follows. Chapter 2 presents some basic knowledge about MINLP: the definition of a MINLP problem, special cases related to this study, a branch-and-bound algorithm, and components of it. We discuss direct objective optimization (Chapter 3) and SVP (Chapter 4), respectively, and each of the chapters gives the overview in the first section. We summarize our conclusions in Chapter 5.

Chapter 2

Branch-and-Bound Algorithm

2.1 MINLP problems

This section presents the definition of an MINLP problem and special cases related to this study. The MINLP problem is expressed in the following form:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & x = (x_1, \ldots, x_n)^T \in X,\ x_i \in \mathbb{Z}\ (i \in I). \end{aligned} \tag{MINLP}$$

The set X is a subset of $\mathbb{R}^n$, for example, $X = \{x \in \mathbb{R}^n : g_j(x) \leq 0 \text{ for all } j = 1, \ldots, m\}$. Then, $g_j(x) \leq 0$ is called a *constraint*. The function $f : X \to \mathbb{R}$ is called the *objective function*. The *variables* $x_1, \ldots, x_n$ contain the *integer variables* x_i $(i \in I)$ and the *continuous variables* x_i $(i \in \{1, \ldots, n\} \setminus I)$. If x satisfies $x \in X$ and $x_i \in \mathbb{Z}$ for all $i \in I$, x is said to be *feasible* for (MINLP) or a *feasible solution*. Otherwise, x is said to be *infeasible* for it. The purpose of (MINLP) is to find an *optimal solution* x^* such that $f(x^*) \leq f(x)$ for any feasible solution x and to compute the *optimal value* $f(x^*)$. If no feasible solution exists, (MINLP) is said to be *infeasible*. Otherwise, (MINLP) is said to be *feasible*. For convenience, we assume that the objective function $f : X \to \mathbb{R}$ is bounded below if (MINLP) is feasible. In this paper, we may denote (MINLP) by

$$f(x^*) := \min_{x}\{f(x) : x \in X, x_i \in \mathbb{Z}\ (i \in I)\}.$$

There exists several special cases of MINLP. Herein, we briefly introduce ones related to this study as follows:

- *MILP:* If the objective function f is linear, and (MINLP) contains only linear constraints (i.e., the functions g_j appearing in X are linear), then it is regarded as a *mixed integer linear programming (MILP)* problem.
- *NLP:* If all integer constraints $x_i \in \mathbb{Z}$ are removed from (MINLP), it is consider as a *nonlinear programming (NLP)* problem. Then, the optimal value of the problem without the integer constraints is a lower bound of the optimal value of the original problem.
- *convex MINLP:* If f and all g_j appearing in X are convex, which means they satisfy the inequality

$$h(\alpha x + \beta y) \leq \alpha h(x) + \beta h(y),$$

where $h \in \{f, g_1, \dots, g_m\}$, and $\alpha, \beta \in \mathbb{R}$ with $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$, then (MINLP) is called a *convex MINLP* problem. Convexity leads to many benefits to solve optimization problems (see e.g., [18, 20]).

- *MIQP and IQP:* If $f(x) = x^T Qx + p^T x + r$, is a (convex) quadratic function of x, and there are only linear constraints on (MINLP), then it is known as a *(convex) mixed integer quadratic programming (MIQP)* problem. In addition, if all x_j are integer variables, (MINLP) is a *(convex) integer quadratic programming (IQP)* problem.
- *MISOCP:* If the objective function is linear, and all nonlinear constraints consist of the form $\|Ax+b\|_2 - c^T x \leq d$, then (MINLP) is a *mixed integer second-order cone programming (MISOCP)* problem.

2.2 Algorithm design

A branch-and-bound algorithm [8, 9, 15, 56] is a design paradigm for discrete and combinatorial optimization problems including (MINLP). The idea of this algorithm is to successively divide a given problem into smaller subproblems until each of the subproblems is solvable. A branch-and-bound algorithm for solving MILP problems was first proposed in [42]. Afterward, Dakin extended the branch-and-bound algorithm to MINLP problems [26]. Other early work related to branch-and-bound algorithms for solving MINLP problems include [19, 30]. All of the most successful optimization software is based on branch-and-bound algorithms or branch-and-cut algorithms [56] which employs cutting planes to tighten the set of feasible solutions. In this section, we briefly introduce a branch-and-bound algorithm. Let $P(X)$ be the problem (MINLP) and $\theta(X)$ the optimal value of (MINLP).

Branching: The branch-and-bound algorithm splits repeatedly a set of feasible solutions into a few sets by *branching*, for example, $X = X_1 \cup X_2$ $(X_1 \cap X_2 = \emptyset)$, and constructs a branch-and-bound tree of the nodes corresponding to the split sets. The problems $P(X_j)$ for both $j = 1, 2$ are called *subproblems* of $P(X)$. Then, the optimal value $\theta(X)$ of $P(X)$ is $\min\{\theta(X_1), \theta(X_2)\}$.

Relaxation: For any nonempty subset $Y \subseteq X$, a lower bound of the optimal value of subproblem $P(Y)$ is computed by *relaxation.* By relaxing the integrality of x_i $(i \in I)$, a standard *relaxation problem* to obtain the lower bound $\ell(Y)$ is formulated as follows:

$$\ell(Y) := \min_x \{f(x) : x \in Y\}. \tag{2.2.1}$$

Let $R(Y)$ be the problem (2.2.1). The optimal value $\ell(Y)$ of $R(Y)$ is a lower bound of $\theta(Y)$ because the optimal solution of $P(Y)$ is feasible for $R(Y)$.

Pruning: Given any feasible solution $\bar{x}$ of $P(X)$, the branch-and-bound algorithm tests whether nodes can be pruned from the tree. For example, if $\ell(X_1) \geq f(\bar{x})$, then $\theta(X) = \min\{f(\bar{x}), \theta(X_2)\}$ because of $\theta(X) = \min\{\theta(X_1), \theta(X_2)\}$. Hence, it is not necessary to solve the subproblem $P(X_1)$, and the node corresponding X_1 can be pruned from the branch-and-bound tree. Feasible solutions are obtained by two procedure. The first one is solving the relaxation problem $R(Y)$. The optimal solution of $R(Y)$ is possibly feasible for $P(Y)$. The second one is a *heuristic method* (e.g., local search).

In this way, the branch-and-bounds algorithm repeats the branching, relaxation, and pruning. We summarize this process in Algorithm 1.

Algorithm 1: Branch-and-bound algorithm

```
Input: An MINLP problem $P(X)$
Output: An optimal solution $x^*$ and the optimal value $\theta(X)$ of $P(X)$
$\mathcal{N} \leftarrow \{X\}$ and $\theta^* \leftarrow \infty$;
while $\mathcal{N} \neq \emptyset$ do
    Select $Y \in \mathcal{N}$ and $\mathcal{N} \leftarrow \mathcal{N} \setminus \{Y\}$;
    Compute a lower bound $\ell$ of $\theta(Y)$ by relaxation;
    if $\ell \geq \theta^*$ then continue;
    Apply a heuristic method optionally and let $\bar{x}$ be a feasible solution;
    if $f(\bar{x}) \leq \theta^*$ then $\theta^* \leftarrow \bar{x}$ and $x^* \leftarrow \bar{x}$;
    Split $Y$ into a few subsets $Y_1, \ldots, Y_k \subseteq Y$ and $\mathcal{N} \leftarrow \mathcal{N} \cup \{Y_1, \ldots, Y_k\}$;
end
$\theta(X) \leftarrow \theta^*$;
return $x^*$ and $\theta(X)$;
```

2.3 Basic components

In this section, we describe the following basic components of the branch-and-bound algorithm: relaxation, branching (rules), and heuristic methods. Moreover, we briefly introduce the functions of them and techniques.

Relaxation: One of the keys to algorithmic speedup is pruning. To this end, it is necessary to obtain good lower bounds of the optimal values of subproblems. Generally, the standard relaxation problem (2.2.1) is employed to compute a lower bound. Using the structure of a given problem, we may define another relaxation problem. If the optimal value of it is better than that of the standard relaxation problem, it is effective in pruning. For example, Buchheim et al. proposed effective relaxation, which provides better lower bounds for convex IQP problems [23]. There exist other procedures using cutting planes which reduce the sets of feasible solutions of subproblems (see, e.g., [8, Chapter 8]).

We need to consider computational costs of solving relaxation problem because most branch-and-bound algorithms spend much computational time on solving them. For example, we transformed a standard relaxation problem from a nonconvex NLP problem into a convex quadratic programming problem [38]. Though the nonconvex NLP problem was impractical, the transformed relaxation problem can be solved with a linear system.

Branching (rules): Most branch-and-bound algorithms employ the branching that uses an optimal solution $\tilde{x}$ of a standard relaxation problem $R(Y)$ and splits a set of feasible solutions into two sets. For any subproblem $P(Y)$, this standard branching selects an index $j \in I$ with a fractional value $\tilde{x}_j$ and splits Y into $Y_1 = \{x \in Y : x_j \leq \lfloor \tilde{x}_j \rfloor\}$ and $Y_2 = \{x \in Y : x_j \geq \lceil \tilde{x}_j \rceil\}$. A *branching rule* is the function for selection of index. Because branching is one of the cores of a branch-and-bound algorithm, it is important for solving MINLP problems to employ good branch rules. In [8, Chapter 5], many branching rules are listed and compared.

Heuristic methods: To prune branch-and-bound nodes from the tree, it is necessary to find good feasible solutions early. Most optimization software contain many heuristic methods for general optimization problems, such as MILP and MIQP problems. However, these methods do not always find feasible solutions. In that case, we should build heuristic methods using the structure of the problem.

Chapter 3

Direct Objective Optimization in Variable Selection

3.1 Overview

An objective function of variable selection typically consists of two competing terms (see, e.g., [31]): *the goodness-of-fit* and *the number of explanatory variables.* Given a dataset and a statistical model with p parameters, variable selection with this objective function can be formulated as the following MINLP problem:

$$\min_{\beta,z} \quad f(\beta) + \lambda \sum_{j=1}^{p} z_j \tag{3.1.1}$$

$$\text{s.t.} \quad z_j = 0 \Rightarrow \beta_j = 0 \ (j = 1, \ldots, p), \tag{3.1.2}$$

$$\beta_j \in \mathbb{R}, \ z_j \in \{0, 1\} \ (j = 1, \ldots, p), \tag{3.1.3}$$

where $\beta = (\beta_1, \ldots, \beta_p)^T$ represents the parameters in the given statistical model, and λ is a positive constant. The first term $f(\beta)$ of the objective function (3.1.1) corresponds to a goodness-of-fit, for example, a discrepancy between the given dataset and the statistical model. The second term $\lambda \sum_{j=1}^{p} z_j$ operates as a penalty for the number of variables. The constraints (3.1.2) represent indicator constraints, that is, β_j has to be zero if z_j is zero. This problem (3.1.1)–(3.1.3) is considered as ℓ_0-penalized variable selection.

We assume the following for $f(\beta)$ in the objective function (3.1.1):

Assumption 1. *For any nonempty subset $S \subseteq \{1, \ldots, p\}$, we can compute the optimal value and an optimal solution of the following optimization problem:*

$$\min_{\beta \in \mathbb{R}^p} \ f(\beta) \ \textit{s.t.} \ \beta_j = 0 \ (j \in \{1, \ldots, p\} \backslash S). \tag{3.1.4}$$

If f is a strongly convex function, the problem (3.1.4) becomes an unconstrained convex problem that can be solved by applying a gradient algorithm, for instance, the steepest descent method and Newton's method.

In this chapter, we focus on the MINLP problem (3.1.1)–(3.1.3) for variable selection. In Section 3.2, we develop an effective branch-and-bound algorithm to solve (3.1.1)–(3.1.3). This algorithm consists of the following techniques:

(Section 3.2.1) effective relaxation to compute lower bounds,

(Section 3.2.2) a heuristic method to find good feasible solutions,

(Section 3.2.3) most frequent branching and strong branching.

In Section 3.3, we explain the following applications of the MINLP problem (3.1.1)–(3.1.3):

(Section 3.3.1) AIC minimization for linear regression,

(Section 3.3.2) AIC minimization for logistic regression.

In other words, these minimization can be formulated as the form of (3.1.1)–(3.1.3). In Section 3.4, we show numerical experiments pertaining to AIC minimization for linear and logistic regression and compare the branch-and-bound algorithm developed in Section 3.2 with other approaches. Moreover, we compare the computational performance of the branching rules and examine which of the proposed techniques is effective.

3.2 Customized branch-and-bound algorithm

In [38, 36], we formulated AIC minimization for linear and logistic regression as MINLP problems and proposed branch-and-bound algorithms purpose-built for these problem. These algorithms consist of components related to effective relaxation, a heuristic method, and branching rules. In this subsection, we explain how these components are applicable to the MINLP problem (3.1.1)–(3.1.3). In Section 3.2.1, we explain the relaxation to compute lower bounds efficiently. Moreover, we show that a feasible solution of the subproblem can be obtained easily from an optimal solution of the proposed relaxation problem. In Sections 3.2.2, we describe the heuristic method based on stepwise method to find feasible solutions. In Sections 3.2.3, we explain most frequent branching proposed previously [38] and describe strong branching.

3.2.1 Relaxation

Branching fixes a binary variable z_j of the problem (3.1.1)–(3.1.3) to zero or one and generates two nodes repeatedly. For any node, we define the sets Z_0, Z_1, and Z as follows:

$$\begin{aligned} Z_1 &= \{j \in \{1,\dots,p\} : z_j \text{ is already fixed to } 1\},\\ Z_0 &= \{j \in \{1,\dots,p\} : z_j \text{ is already fixed to } 0\},\\ Z &= \{j \in \{1,\dots,p\} : z_j \text{ is not fixed}\}. \end{aligned}$$

Then, the subproblem of the problem (3.1.1)–(3.1.3) can be expressed as follows:

$$\min_{\beta,z} \quad f(\beta) + \lambda \sum_{j=1}^{p} z_j \tag{3.2.1}$$

$$\text{s.t.} \quad \beta_j \in \mathbb{R},\ z_j = 1\ (j \in Z_1),\ \beta_j = z_j = 0\ (j \in Z_0), \tag{3.2.2}$$

$$z_j = 0 \Rightarrow \beta_j = 0,\ \beta_j \in \mathbb{R},\ z_j \in \{0,1\}\ (j \in Z). \tag{3.2.3}$$

We denote the subproblem (3.2.1)–(3.2.3) by $Q(Z_1, Z_0, Z)$ because the subproblem can be specified uniquely by using Z_1, Z_0, and Z. By relaxing the integrality of the variables z_j, we obtain the following standard relaxation problem of $Q(Z_1, Z_0, Z)$:

$$\min_{\beta,z} \; f(\beta) + \lambda \sum_{j=1}^{p} z_j \tag{3.2.4}$$

$$\text{s.t.} \;\; \beta_j \in \mathbb{R},\; z_j = 1 \; (j \in Z_1),\; \beta_j = z_j = 0 \; (j \in Z_0), \tag{3.2.5}$$

$$z_j = 0 \Rightarrow \beta_j = 0,\; \beta_j \in \mathbb{R},\; 0 \le z_j \le 1 \; (j \in Z). \tag{3.2.6}$$

The optimal value of the problem (3.2.4)–(3.2.6) is the lower bound of the optimal value of $Q(Z_1, Z_0, Z)$. Instead of solving (3.2.4)–(3.2.6), we consider the following problem:

$$\min_{\beta} \; f(\beta) + \lambda \#(Z_1) \text{ s.t. } \beta_j = 0 \; (j \in Z_0),\; \beta_j \in \mathbb{R} \; (j \in Z_1 \cup Z), \tag{3.2.7}$$

where $\#(Z_1)$ stands for the number of elements in the set Z_1. This problem is arrived at by eliminating the indicator constraints and the variables z_j from the problem (3.2.4)–(3.2.6). Notably, the optimal value of the problem (3.2.7) is the lower bound of the optimal value of $Q(Z_1, Z_0, Z)$. In fact, the optimal value of (3.2.7) is smaller than or equal to the optimal value of (3.2.4)–(3.2.6) because any feasible solution (β, z) of (3.2.4)–(3.2.6) is also feasible for (3.2.7) and satisfies the following inequality:

$$f(\beta) + \lambda \sum_{j=1}^{p} z_j = f(\beta) + \lambda \left(\sum_{j \in Z} z_j + \#(Z_1) \right) \ge f(\beta) + \lambda \#(Z_1).$$

Hence, we employ (3.2.7) as a relaxation problem of the subproblem $Q(Z_1, Z_0, Z)$ to compute a lower bound of the optimal value of $Q(Z_1, Z_0, Z)$. We denote the relaxation problem (3.2.7) by $R(Z_1, Z_0, Z)$, which is an unconstrained convex problem. We can solve $R(Z_1, Z_0, Z)$ under Assumption 1.

The following lemma implies that the optimal value of $R(Z_1, Z_0, Z)$ is identical to the optimal value of the standard relaxation problem (3.2.4)–(3.2.6).

Lemma 3.2.1. *Let θ^* be the optimal value of $R(Z_1, Z_0, Z)$. Then, the optimal value of (3.2.4)–(3.2.6) is θ^*.*

Proof. Let β^* be an optimal solution of $R(Z_1, Z_0, Z)$. We construct a sequence $\{(\beta^N, z^N)\}_N^\infty$ as follows:

$$\beta^N = \beta^* \text{ and } z_j^N = \begin{cases} 1 & \text{if } j \in Z_1, \\ 1/N & \text{if } j \in Z, \\ 0 & \text{otherwise,} \end{cases} \quad (j = 1, \dots, p)$$

for all $N \ge 1$. (β^N, z^N) is feasible for (3.2.4)–(3.2.6) for all $N \ge 1$. It is sufficient to prove that the objective value θ^N of (3.2.4)–(3.2.6) at (β^N, z^N) converges to the optimal value θ^* of $R(Z_1, Z_0, Z)$ as N approaches infinity. Because we have $\theta^* = f(\beta^*) + \lambda \#(Z_1)$ and

$$\theta^* \le \theta^N = f(\beta^*) + \lambda \#(Z_1) + \frac{\lambda}{N} \#(Z) = \theta^* + \frac{\lambda}{N} \#(Z),$$

θ^N converges to θ^* as N approaches to infinity. This implies that the optimal value of $R(Z_1, Z_0, Z)$ is identical to the optimal value of (3.2.4)–(3.2.6). □

We can easily solve the relaxation problem of the subproblem obtained by fixing z_j to 1. By fixing the variable z_k, two subproblems $Q(Z_1 \cup \{k\}, Z_0, Z\backslash\{k\})$ and $Q(Z_1, Z_0 \cup \{k\}, Z\backslash\{k\})$ are generated from $Q(Z_1, Z_0, Z)$. The relaxation problem $R(Z_1 \cup \{k\}, Z_0, Z\backslash\{k\})$ can then be formulated as follows:

$$\min_{\beta} \ f(\beta) + \lambda\#(Z_1 \cup \{k\}) \text{ s.t. } \beta_j = 0 \ (j \in Z_0), \ \beta_j \in \mathbb{R} \ (Z_1 \cup Z).$$

Therefore, the optimal value of the relaxation problem $R(Z_1 \cup \{k\}, Z_0, Z\backslash\{k\})$ for any $k \in Z$ is $\theta^* + \lambda$, where θ^* is the optimal value of the relaxation problem $R(Z_1, Z_0, Z)$.

We explain a procedure to generate a feasible solution of the subproblem $Q(Z_1, Z_0, Z)$ from an optimal solution of $R(Z_1, Z_0, Z)$. Let $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)^T$ be the optimal solution of $R(Z_1, Z_0, Z)$. We define $\hat{z} = (\hat{z}_1, \ldots, \hat{z}_p)$ by $\hat{z}_j = 1$ if $\hat{\beta}_j \neq 0$, otherwise $\hat{z}_j = 0$. Clearly, $(\hat{\beta}, \hat{z})$ is feasible for $Q(Z_1, Z_0, Z)$.

3.2.2 Heuristic method based on stepwise methods

To prune shallow nodes from a branch-and-bound tree, it is necessary to find a good feasible solution early. To this end, we develop an effective heuristic method, which is based on stepwise methods with forward selection and backward elimination. The stepwise methods often find good statistical models via goodness-of-fit measures, such as AIC [12] and BIC [50], and these are implemented in statistical software (e.g., R [53]).

First, we briefly explain the stepwise methods. Generally, the stepwise method with forward selection starts with no explanatory variables in a given model and repeats the following steps until no further improvement is possible.

Step 1. Test the addition of each explanatory variable using a chosen goodness-of-fit measure.

Step 2. Add the variable whose inclusion gives the most significant improvement.

The stepwise method with backward elimination starts with all explanatory variables. Instead of adding the variable, it deletes the variable from the model. Although these methods are considered as local search algorithms, they often find good statistical models within a short time.

Next, we extend the capability of the stepwise methods to find feasible solutions of any subproblem $Q(Z_1, Z_0, Z)$. As a results, we expect that our heuristic methods will find good feasible solutions early. Given any subset $S \subseteq \{1, \ldots, p\}$ with $Z_1 \subseteq S \subseteq Z_1 \cup Z$, we define the vector $\bar{z}^S = (\bar{z}^S_1, \ldots, \bar{z}^S_p)^T \in \{0,1\}^p$ as follows:

$$\bar{z}^S_j := \begin{cases} 1 & \text{if } j \in S \\ 0 & \text{if } j \in \{1, \ldots, p\} \setminus S \end{cases} \quad \text{for all } j = 1, \ldots, p.$$

By substituting $\bar{z}^S$ for z in $Q(Z_1, Z_0, Z)$, it can be rewritten as

$$\min_{\beta \in \mathbb{R}^p} \ f(\beta) + \lambda\#(S) \text{ s.t. } \beta_j = 0 \ (j \in \{1, \ldots, p\} \setminus S), \tag{3.2.8}$$

where $\#(S)$ denotes the number of elements in S. An optimal solution of this problem (3.2.8) can be computed under Assumption 1. Let $\bar{\theta}^S$ and $\bar{\beta}^S$ be the optimal value and an optimal solution of (3.2.8), respectively. Then, $(\bar{\beta}^S, \bar{z}^S)$ is a feasible solution of $Q(Z_1, Z_0, Z)$. In fact, all the constraints of $Q(Z_1, Z_0, Z)$ is satisfied because of $Z_1 \subseteq S \subseteq Z_1 \cup Z$ and $Z_0 \cap S = \emptyset$. Our heuristic method improves repeatedly given feasible solutions by solving problems (3.2.8). We describe the heuristic method for $Q(Z_1, Z_0, Z)$ in Algorithm 2. We use $(\bar{\beta}^{Z_1}, \bar{z}^{Z_1})$ and $(\bar{\beta}^{Z_1 \cup Z}, \bar{z}^{Z_1 \cup Z})$ as the initial solutions (β^1, z^1) and (β^2, z^2), respectively, in our implementation.

Algorithm 2: Heuristic method based on the stepwise methods

```
Input: A subproblem $Q(Z_1, Z_0, Z)$ and two initial feasible solutions $(\beta^1, z^1)$ and
       $(\beta^2, z^2)$ of $Q(Z_1, Z_0, Z)$
Output: A feasible solution $(\beta, z)$ of $Q(Z_1, Z_0, Z)$
$S \longleftarrow \{j \in \{1, \ldots, p\} : z^1_j = 1\}$, $v^f \longleftarrow \infty$;
/* the stepwise method with forward selection                          */
while $\bar{\theta}^S < v^f$ do
    $v^f \longleftarrow \bar{\theta}^S$, $(\beta^f, z^f) \longleftarrow (\bar{\beta}^S, \bar{z}^S)$;
    Find $J = \operatorname*{arg\,min}_{j \in Z \setminus S} \{\bar{\theta}^{S \cup \{j\}} : \bar{z}^{S \cup \{j\}}$ is feasible for $Q(Z_1, Z_0, Z)\}$;
    if $J = \emptyset$ then break;
    Select $j \in J$ and $S \longleftarrow S \cup \{j\}$;
end
$S \longleftarrow \{j \in \{1, \ldots, p\} : z^2_j = 1\}$, $v^b \longleftarrow \infty$;
/* the stepwise method with backward elimination                       */
while $\bar{\theta}^S < v^b$ do
    $v^b \longleftarrow \bar{\theta}^S$, $(\beta^b, z^b) \longleftarrow (\bar{\beta}^S, \bar{z}^S)$;
    Find $J = \operatorname*{arg\,min}_{j \in Z \cap S} \{\bar{\theta}^{S \setminus \{j\}} : \bar{z}^{S \setminus \{j\}}$ is feasible for $Q(Z_1, Z_0, Z)\}$;
    if $J = \emptyset$ then break;
    Select $j \in J$ and $S \longleftarrow S \setminus \{j\}$;
end
if $v^f < v^b$ then return $(\beta^f, z^f)$;
else return $(\beta^b, z^b)$;
```

3.2.3 Most frequent branching and strong branching

In this section, we customize branching rules to early prune nodes of a branch-and-bound tree. We employ two branching rule: *most frequent branching* and *strong branching*, which are proposed previously [38]. The most frequent branching is based on two tendencies of (3.1.1)–(3.1.3): a few binary variables z_j $(j \in \{1, \ldots, p\})$ of good feasible solutions are often 1, and such variables z_j of an optimal solution are 1. The strong branching is based on a branching rule in [8, Section 5.4]. We define this branching suitable for (3.1.1)–(3.1.3).

First, we describe proposed relaxation problems of subproblems generated by branching variable z_k. For any subproblem $Q(Z_1, Z_0, Z)$, if z_k $(k \in Z)$ is chosen as a branching variable, two subproblems $Q(Z_1 \cup \{k\}, Z_0, Z \backslash \{k\})$ and $Q(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$ are generated. The relaxation problems of these subproblems are $R(Z_1 \cup \{k\}, Z_0, Z \backslash \{k\})$, that is,

$$\min_{\beta} \left\{ f(\beta) + \lambda \#(Z_1 \cup \{k\}) : \beta_j = 0 \ (j \in Z_0), \ \beta_j \in \mathbb{R} \ (j = 1, \ldots, p) \right\}, \tag{3.2.9}$$

and $R(Z_1, Z_0 \cup \{k\}, Z\backslash\{k\})$, that is,

$$\min_{\beta} \{ f(\beta) + \lambda \#(Z_1) : \beta_j = 0 \ (j \in Z_0 \cup \{k\}), \ \beta_j \in \mathbb{R} \ (j = 1, \ldots, p) \} . \tag{3.2.10}$$

The optimal value of (3.2.9) is $\theta^* + \lambda$ for any $k \in Z$ as described in Section 3.2.1, where θ^* is the optimal value of $R(Z_1, Z_0, Z)$. In other words, it does not depends on branching variable whether the corresponding node is pruned from a branch-and-bound tree. Hence, we focus on only the optimal value θ^*_k of (3.2.10) for all $k \in Z$. To prune the corresponding nodes early, we need to select a branching variable z_k $(k \in Z)$ with as large value θ^*_k as possible.

Strong branching [8, Section 5.4] tests which of the candidates of branching variable give the best progress in the lower bounds before actually branching on any of them. This test is done by temporarily branching and solving relaxation problems. Applying this strong branching to (3.1.1)–(3.1.3), we execute Algorithm 3. Because this rule means selecting the locally best branching variable, it is generally very effective in terms of the number of subproblems needed to solve the original problem. Unfortunately, the computation time per subproblem of the strong branching are expensive since it is necessary to solve a number of relaxation problems.

Algorithm 3: Strong branching

```
Input: A set $Z$ of indices of unfixed variables
Output: A branching variable $z_k$ $(k \in Z)$
for $j \in Z$ do
    Solve $R(Z_1, Z_0 \cup \{j\}, Z\backslash\{j\})$ and obtain the optimal value $\theta^*_j$;
end
return $z_k$ with $\theta^*_k = \max_{j \in Z}\{\theta^*_j\}$
```

Next, we explain the most frequent branching [38]. When the feasible solution set of a subproblem does not include an optimal solution and good feasible solutions of the original problem, the optimal value of the proposed relaxation problem of the subproblem might be large. As described in the beginning of this subsection, a few binary variables z_j of an optimal and good feasible solutions are often 1. Hence, selecting such a variable z_k as branching variable, we expect that the optimal value θ^*_k of (3.2.10) will be large. We describe this branching rule in Algorithm 4.

Algorithm 4: Most frequent branching

```
Input: A positive integer $N$, a set $Z$ of indices of unfixed variables, and the current
       pool of feasible solutions of (3.1.1)–(3.1.3)
Output: A branching variable $z_k$ $(k \in Z)$
Choose the top $N$ feasible solutions $(\beta^1, z^1), \ldots, (\beta^N, z^N)$ from the pool;
/* Here $(\beta^i, z^i)$ is a feasible solution with the $i$th lowest objective value
   in the pool.                                                                    */
for $j \in Z$ do
    Compute score value $s_j := \sum_{i=1}^{N} z^i_j$;
end
return $z_k$ with $s_k = \max_{j \in Z}\{s_j\}$
```

3.3 Applications

In variable selection, goodness-of-fit measures, for example, AIC [12] and BIC [50], are often employed for evaluation of statistical models. In variable selection based on AIC, the AIC value is computed for each model, and the model with the lowest AIC value is selected as the best statistical model. However, the computation of all candidates is not practical because the number of candidates of possible statistical models is exponentially large. We explain that AIC minimization for linear regression (Section 3.3.1) and logistic regression (Section 3.3.2) can be formulated as the form of the MINLP problem (3.1.1)–(3.1.3). Therefore, applying purpose-built branch-and-bound algorithm described in Section 3.2, we compute efficiently the statistical model with the lowest AIC value. Moreover, we explain techniques for fast implementation in each subsection and Section 3.3.3.

3.3.1 Application 1: AIC minimization for linear regression

We explain that AIC minimization for linear regression can be formulated as the form of the MINLP problem (3.1.1)–(3.1.3) in this subsection. To this end, we define firstly AIC and AIC minimization. Given a dataset with p *explanatory variables* and a statistical model with $p+k$ parameters, the parameter corresponding to the jth explanatory variable is denoted by $\beta_j \in \mathbb{R}$ for all $j = 1, \dots, p$, and is called a *coefficient parameter*. If a coefficient parameter β_j is nonzero, the statistical model becomes involved with the jth explanatory variable. A statistical model may have parameters except for the coefficient parameters. For instance, a linear regression model has the parameter such as the variance σ^2. Let $\{1, \dots, p\}$ be the set of indices of the given explanatory variables and S a subset of $\{1, \dots, p\}$. For any subset $S \subseteq \{1, \dots, p\}$, the AIC value of the statistical model with the jth explanatory variables ($j \in S$) is defined as follows:

$$\mathrm{AIC}(S) = -2 \max_{\beta} \{\ell(\beta) : \beta_j = 0 \ (j \in \{1, \dots, p\} \backslash S), \ \beta \in \mathbb{R}^{p+k}\} + 2(\#(S) + k), \qquad (3.3.1)$$

where $\ell(\beta)$ is the log-likelihood function, and $\#(S)$ stands for the number of elements in S. In variable selection based on AIC, the model with the lowest AIC value is selected as the best statistical model. Therefore, this variable selection can be formulated as follows:

$$\min_{S} \{\mathrm{AIC}(S) : S \subseteq \{1, \dots, p\}\}. \qquad (3.3.2)$$

This problem (3.3.2) is called *AIC minimization*. Unfortunately, it is practically difficult to solve (3.3.2) by computing the AIC values of all models because the number of model candidates is 2^p.

Secondly, we briefly introduce linear regression. Linear regression is the fundamental statistical tool, and it determines coefficient parameters $\beta_1, \dots, \beta_p \in \mathbb{R}$ of the following equation from a given dataset:

$$y = \beta_1 + \sum_{j=2}^{p} \beta_j x_j.$$

Here $x_2, \dots, x_p$ and y are called explanatory variables and a response variable, respectively. A given dataset is denoted by $(x_{i1}, \dots, x_{ip}, y_i) \in \mathbb{R}^p \times \mathbb{R} \ \ (i = 1, \dots, n)$ with $x_{i1} = 1$. Under the assumption that all the residual $\epsilon_i = y_i - \sum_{j=1}^p \beta_j x_{ij}$ are independent and normally distributed with the zero mean and variance σ^2, the log-likelihood function can be formulated as follows:

$$\ell(\beta, \sigma^2) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^n \left(y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

From (3.3.1), the AIC value is computed for any $S \subseteq \{1, \dots, p\}$ as follows:

$$\mathrm{AIC}(S) = -2 \max_{\beta\in\mathbb{R}^p, \sigma^2\in\mathbb{R}} \left\{ \ell(\beta, \sigma^2) : \beta_j = 0 \ (j \in \{1, \dots, p\} \setminus S) \right\} + 2(\#(S) + 1). \tag{3.3.3}$$

We focus on the first term to simplify (3.3.3). By substituting $\beta_j = 0$ $(j \in \{1, \dots, p\} \setminus S)$ to the objective function, the first term can be regarded as an unconstrained minimization. Thus minimum solutions satisfy the following equation:

$$\frac{d\ell}{d(\sigma^2)} = -\frac{n}{2\sigma^2} + \frac{1}{2(\sigma^2)^2}\sum_{i=1}^n \left(y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2 = 0.$$

From this equation, we obtain $\sigma^2 = \frac{1}{n}\sum_{i=1}^n \left(y_i - \sum_{j=1}^p \beta_j x_{ij}\right)^2$. Substituting the variance σ^2 to (3.3.3), we simplify (3.3.3) as follows:

$$\begin{aligned} \mathrm{AIC}(S) = \min_{\beta\in\mathbb{R}^p} &\left\{ n \log \sum_{i=1}^n \left(y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \ : \ \beta_j = 0 \ (j \in \{1, \dots, p\} \setminus S) \right\} \\ &+ 2(\#(S) + 1) + n\left(\log(2\pi/n) + 1\right). \end{aligned} \tag{3.3.4}$$

Finally, we formulate the minimization of AIC(S) over $S \subseteq \{1, \dots, p\}$ as the form of the MINLP problem (3.1.1)–(3.1.3). By eliminating the constant terms of (3.3.4) and introducing auxiliary binary variables $z_1, \dots, z_p \in \{0, 1\}$, AIC minimization for linear regression can be formulated as follows:

$$\min_{\beta, z} \ n \log \sum_{i=1}^n \left(y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + 2\sum_{j=1}^p z_j \tag{3.3.5}$$

$$\text{s.t.} \ z_j = 0 \Rightarrow \beta_j = 0 \ (j = 1, \dots, p), \tag{3.3.6}$$

$$\beta_j \in \mathbb{R}, \ z_j \in \{0, 1\} \ (j = 1, \dots, p), \tag{3.3.7}$$

From (3.2.7), the proposed relaxation problem $R(Z_1, Z_0, Z)$ of any subproblem $Q(Z_1, Z_0, Z)$ of (3.3.5)–(3.3.7) can be formulated as follows:

$$\min_{\beta} \ n \log \sum_{i=1}^n \left(y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + 2\#(Z_1) \ \text{s.t.} \ \beta_j = 0 \ (j \in Z_0), \ \beta_j \in \mathbb{R} \ (j = 1, \dots, p). \tag{3.3.8}$$

Although the objective function of (3.3.8) contains the logarithm function, we can freely remove the constant $2\#(Z_1)$ and the logarithm by the monotonicity of the logarithm function in (3.3.8), and thus obtain the following problem from (3.3.8):

$$\min_{\beta} \sum_{i=1}^n \left(y_i - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \ \text{s.t.} \ \beta_j = 0 \ (j \in Z_0), \ \beta_j \in \mathbb{R} \ (j = 1, \dots, p). \tag{3.3.9}$$

Because the problem (3.3.9) is regarded as the unconstrained minimization of a convex quadratic function, we can compute an optimal solution of (3.3.9) by solving a linear system. Thus Assumption 1 holds for variable selection based on AIC in linear regression. In our implementation, we call `dposv` function, which is a built-in function of LAPACK [13] for solving the linear system. We denote the optimal value of (3.3.9) by ξ^*. The optimal value of (3.3.8) is $n\log(\xi^*) + 2\#(Z_1)$, and this value is used as a lower bound of the optimal value of the subproblem.

3.3.2 Application 2: AIC minimization for logistic regression

In this section, we formulate AIC minimization for logistic regression as the form of the MINLP problem (3.1.1)–(3.1.3). First, we introduce a logistic regression model. Logistic regression is a fundamental statistical tool, and it estimates the probability of a binary response from a given dataset $(x_{i1}, \dots, x_{ip}, y_i) \in \mathbb{R}^p \times \{0,1\}$ with $x_{i1} = 1$ $(i = 1, \dots, n)$. We regard y_i as a class label of the ith data for all $i = 1, \dots, n$. Logistic regression determines coefficient parameters $\beta_1, \dots, \beta_p$ of the following logistic regression model which determines the probability of $y = 1$ for an input $x = (x_1, \dots, x_p)^T \in \mathbb{R}^p$,

$$P(y = 1 \mid x) = \frac{\exp\left(\sum_{j=1}^p \beta_j x_j\right)}{1 + \exp\left(\sum_{j=1}^p \beta_j x_j\right)}.$$

Here $x_1, \dots, x_p$ and y are explanatory variables and a response variable, respectively. The probability of $y = 0$ is obtained by simple calculation,

$$P(y = 0 \mid x) = 1 - P(y = 1 \mid x) = \frac{1}{1 + \exp\left(\sum_{j=1}^p \beta_j x_j\right)}.$$

Therefore, the probability of $y \in \{0,1\}$ can be written as

$$P(y \mid x) = \frac{\exp\left(y\sum_{j=1}^p \beta_j x_j\right)}{1 + \exp\left(\sum_{j=1}^p \beta_j x_j\right)}.$$

In logistic regression, the coefficient parameters $\beta_1, \dots, \beta_p$ can be determined by maximum likelihood estimation. In fact, the log-likelihood function ℓ is defined as

$$\ell(\beta) = \sum_{i=1}^n \log P(y_i \mid x^i) = -\sum_{i=1}^n \left(\log\left(1 + \exp\left(\beta^T x^i\right)\right) - y_i \beta^T x^i\right),$$

where $\beta = (\beta_1, \dots, \beta_p)^T$ and $x^i = (x_{i1}, \dots, x_{ip})^T$ for $i = 1, \dots, n$. From (3.3.1), for any subset $S \subseteq \{1, \dots, p\}$, the AIC value can be computed as follows:

$$\begin{aligned}\mathrm{AIC}(S) = 2\min_{\beta_j} &\left\{ \sum_{i=1}^n \left(\log\left(1 + \exp\left(\beta^T x^i\right)\right) - y_i \beta^T x^i\right) : \begin{array}{l} \beta_j = 0 \ (j \in \{1, \dots, p\} \setminus S) \\ \beta \in \mathbb{R}^p \end{array} \right\} \\ &+ 2\#(S). \end{aligned} \tag{3.3.10}$$

The objective function of the minimization in (3.3.10) is convex because its Hessian matrix is positive semidefinite. The minimization in (3.3.10) is solved for any subset $S \subseteq \{1, \dots, p\}$ by applying a gradient algorithm, for instance, the steepest descent method and Newton's method. In the same way as linear regression, the minimization of AIC(S) over $S \subseteq \{1, \dots, p\}$ can be formulated as the following MINLP problem:

$$\min_{\beta, z} \; 2\sum_{i=1}^{n} \left(\log\left(1 + \exp\left(\beta^T x^i\right)\right) - y_i \beta^T x^i\right) + 2\sum_{j=1}^{p} z_j \tag{3.3.11}$$

$$\text{s.t.} \;\; z_j = 0 \Rightarrow \beta_j = 0 \; (j = 1, \dots, p), \tag{3.3.12}$$

$$\beta_j \in \mathbb{R}, \; z_j \in \{0, 1\} \; (j = 1, \dots, p). \tag{3.3.13}$$

From (3.2.7), the proposed relaxation problem $R(Z_1, Z_0, Z)$ can be written as

$$\min_{\beta} \; 2\sum_{i=1}^{n} \left(\log\left(1 + \exp\left(\beta^T x^i\right)\right) - y_i \beta^T x^i\right) + 2\#(Z_1) \text{ s.t. } \beta_j = 0 \; (j \in Z_0), \; \beta_j \in \mathbb{R} \; (Z_1 \cup Z).$$

Assumption 1 holds for logistic regression analysis under the practical assumption. Therefore, the optimal value of this relaxation problem can be computed. However, Assumption 1 fails in a certain dataset. See Section 3.5 for more details.

Next, we explain an initial guess developed in [36] to solve the proposed relaxation problem (3.2.7) efficiently. We employ Newton's method for (3.2.7). This method is iterative, and it requires an initial feasible solution of the relaxation problem. We construct the initial feasible solution from an optimal solution of the relaxation problem of the parent node. To explain this procedure, we focus on two relaxation problems $R(Z_1 \cup \{k\}, Z_0, Z \backslash \{k\})$ and $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$, which are obtained by fixing the variable z_k. Then, the relaxation problem of the parent node is $R(Z_1, Z_0, Z)$. Let θ^* be the optimal value of $R(Z_1, Z_0, Z)$ and $\beta^* = (\beta_1^*, \dots, \beta_p^*)^T \in \mathbb{R}^p$ the optimal solution of $R(Z_1, Z_0, Z)$. In Section 3.2.1, we showed that the optimal value of $R(Z_1 \cup \{k\}, Z_0, Z \backslash \{k\})$ is $\theta^* + 2$. Hence, we construct only an initial feasible solution of the other relaxation problem $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$. Because $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$ is similar to $R(Z_1, Z_0, Z)$, we expect that the optimal solution of $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$ will be near β^*. Therefore, we construct the initial feasible solution $\beta^0 = (\beta_1^0, \dots, \beta_p^0)^T \in \mathbb{R}^p$ of $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$ as follows:

$$\beta_j^0 = \begin{cases} 0 & \text{if } j = k, \\ \beta_j^* & \text{otherwise,} \end{cases}$$

for all $j = 1, \dots, p$. Because β^* is feasible for $R(Z_1, Z_0, Z)$, β^0 is feasible for $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$.

3.3.3 Effective handling of data structure

In this section, we explain effective handling of data structure in AIC minimization for linear and logistic regression. This handling is developed previously [36]. Standard statistical textbooks often assume that datasets have linear independence; however, as it is some datasets in the UCI Machine Learning Repository [14]. As described in Section 3.3.1, we require solving linear systems for relaxation problems in AIC minimization for linear regression. Given that we apply Newton's method to relaxation problems in AIC minimization for logistic regression, it is necessary to solve linear systems as well. If a given dataset has linear dependence, the linear systems may have infinitely many solutions. Hence, we implement the processing of linear dependence in linear and logistic regression.

First, we explain the following proposition, which involves techniques for solving (3.1.1)–(3.1.3).

Proposition 3.3.1. *Let S be a nonempty subset of $\{1, \dots, p\}$. We assume that for any $s \in S$ and $\tilde{\beta} = (\tilde{\beta}_1, \dots, \tilde{\beta}_p)^T \in \mathbb{R}^p$, there exists $\hat{\beta} \in \mathbb{R}^p$ such that*

$$\hat{\beta}_j = \tilde{\beta}_j \ (j \in \{1, \dots, p\} \backslash S), \ \hat{\beta}_s = 0 \text{ and } f(\tilde{\beta}) = f(\hat{\beta}),$$

where the function f is the first term of the objective function in (3.1.1). Then, the following properties are satisfied:

1. *If $S \subseteq Z_1$, the subproblem $Q(Z_1, Z_0, Z)$ is pruned in the branch-and-bound tree, that is, the optimal value of $Q(Z_1, Z_0, Z)$ is larger than the optimal value of (3.1.1)–(3.1.3).*

2. *If $Z \cap S \neq \emptyset$ and $S \subseteq Z_1 \cup Z$, the optimal value of the relaxation problem $R(Z_1, Z_0, Z)$ is equal to the optimal value of the relaxation problem $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$ for any $k \in Z \cap S$.*

We prove Proposition 3.3.1 at the end of this subsection.

Remark. The first property of Proposition 3.3.1 implies that we can reduce the number of generated branch-and-bound nodes. The second property of Proposition 3.3.1 implies that we can reduce the computational cost of solving the relaxation problem. In fact, we can remove a continuous variable β_k $(k \in Z \cup S)$ from the relaxation problem, where the set S satisfies the assumption in Proposition 3.3.1. We apply this removal repeatedly. Therefore, we can efficiently solve (3.1.1)–(3.1.3) by using the properties of Proposition 3.3.1.

Next, we show that the assumption in Proposition 3.3.1 is satisfied if a given dataset has linear dependence in linear and logistic regression. To explain this, we define linear dependence in datasets and the function $f(\beta)$. For a given dataset $(x_{i1}, \dots, x_{ip}, y_i) \in \mathbb{R}^p \times \mathbb{R}$ with $x_{i1} = 1$ $(i = 1, \dots, n)$, we define the following vectors:

$$x_j = \begin{pmatrix} x_{1j} \\ \vdots \\ x_{nj} \end{pmatrix} \in \mathbb{R}^n \text{ for } j = 1, \dots, p.$$

If these vectors $x_1, \dots, x_p \in \mathbb{R}^n$ are linearly dependent, we say that the dataset has linearly dependent variables. The function $f(\beta)$, that is, the first term of the objective function in (3.1.1), is defined as

$$f(\beta) = \begin{cases} n \log \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 & \text{(if linear regression)} \\ 2 \sum_{i=1}^{n} \left(\log \left(1 + \exp \left(\beta^T x^i \right) \right) - y_i \beta^T x^i \right) & \text{(if logistic regression)} \end{cases}. \tag{3.3.14}$$

Lemmas 3.3.2 and 3.3.3 show that linear dependence in a given dataset corresponds to the assumption in Proposition 3.3.1. Hence, we can reduce the computational cost by applying Proposition 3.3.1.

Lemma 3.3.2. *If a given dataset has linearly dependent variables, there exists a nonempty set $S \subseteq \{1, \ldots, p\}$ such that*

$$\sum_{j \in S} \alpha_j x_j = 0 \text{ and } \alpha_j \neq 0 \text{ for all } j \in S. \tag{3.3.15}$$

Proof. If a given dataset has linearly dependent variables, there exists $\alpha \ (\neq 0) \in \mathbb{R}^p$ such that $\sum_{j=1}^{p} \alpha_j x_j = 0$. Then, the subset S is defined by $\{j \in \{1, \ldots, p\} : \alpha_j \neq 0\}$. It is readily apparent that S is nonempty. □

Lemma 3.3.3. *If a given dataset has linearly dependent variables, there exists a nonempty set $S \subseteq \{1, \ldots, p\}$ such that the S and f defined in (3.3.14) satisfy the assumption in Proposition 3.3.1.*

Proof. Let $\tilde{\beta}$ be $(\tilde{\beta}_1, \ldots, \tilde{\beta}_p)^T \in \mathbb{R}^p$ and I_p a set $\{1, \ldots, p\}$. From Lemma 3.3.2, there exists a nonempty set $S \subseteq \{1, \ldots, p\}$ such that (3.3.15). We consider the two cases: (i) $\#(S) = 1$ and (ii) $\#(S) > 1$.
(i). If S contains a single element (i.e., $S = \{s\}$), $x_s = 0$. We define $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)^T \in \mathbb{R}^p$ as follows:

$$\hat{\beta}_j = \begin{cases} \tilde{\beta}_j, & (j \in I_p \backslash \{s\}) \\ 0, & (j = s) \end{cases}$$

for all $j = 1, \ldots, p$. Because $\tilde{\beta}^T x^i = \hat{\beta}^T x^i$, $f(\tilde{\beta}) = f(\hat{\beta})$ is satisfied.
(ii). For any $s \in S$, there exist $\alpha'_j \neq 0$ $(j \in S \backslash \{s\})$ such that

$$x_{is} = \sum_{j \in S \backslash \{s\}} \alpha'_j x_{ij}$$

for all $i = 1, \ldots, n$. $\tilde{\beta}^T x^i$ $(i = 1, 2, \ldots, n)$ can be written as follows:

$$\begin{aligned} \tilde{\beta}^T x^i &= \sum_{j \in I_p \backslash \{s\}} \tilde{\beta}_j x_{ij} + \tilde{\beta}_s x_{is} \\ &= \sum_{j \in I_p \backslash \{s\}} \tilde{\beta}_j x_{ij} + \tilde{\beta}_s \sum_{j \in S \backslash \{s\}} \alpha'_j x_{ij} \\ &= \sum_{j \in I_p \backslash S} \tilde{\beta}_j x_{ij} + \sum_{j \in S \backslash \{s\}} (\tilde{\beta}_j + \tilde{\beta}_s \alpha'_j) x_{ij}. \end{aligned}$$

Here, we define $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_p)^T \in \mathbb{R}^p$ as follows:

$$\hat{\beta}_j = \begin{cases} \tilde{\beta}_j, & (j \in I_p \backslash S) \\ \tilde{\beta}_j + \tilde{\beta}_s \alpha'_j, & (j \in S \backslash \{s\}) \\ 0, & (j = s) \end{cases}$$

for all $j = 1, \ldots, p$. Because $\tilde{\beta}^T x^i = \hat{\beta}^T x^i$, $f(\tilde{\beta}) = f(\hat{\beta})$ is satisfied. □

As described at the start of this subsection, the linear systems for solving the relaxation problems $R(Z_1, Z_0, Z)$ have infinitely many solutions if the vectors $x_j (j \in Z \cup Z_1)$ are linearly dependent. Therefore, we transform $R(Z_1, Z_0, Z)$ to eliminate such linear dependence. To this end, we use the second property of Proposition 3.3.1. We describe the nonempty set $S \subseteq \{1, \dots, p\}$ of Lemma 3.3.2 as a linearly dependent set. Given any relaxation problem $R(Z_1, Z_0, Z)$ and a linearly dependent set $S \subseteq Z_1 \cup Z$ with $Z \cap S \neq \emptyset$, we select an index $k \in Z \cap S$ and solve $R(Z_1, Z_0 \cup \{k\}, Z \setminus \{k\})$ instead of $R(Z_1, Z_0, Z)$. Because the objective function of $R(Z_1, Z_0 \cup \{k\}, Z \setminus \{k\})$ does not contain the vector x_k, it is regarded as a problem without the linearly dependent set S. Hence, application of the second property of Proposition 3.3.1 corresponds to removal of the linearly dependent set from $R(Z_1, Z_0, Z)$.

To apply Proposition 3.3.1 at each branch-and-bound node, we must find linearly dependent sets. In Algorithm 5, we describe a process proposed in [38] to find a collection $\mathcal{C}(Z, Z_1)$ of the linearly dependent sets. This process ensures that Proposition 3.3.1 is available for any nonempty set $S \in \mathcal{C}(Z, Z_1)$. We state that the linear system (3.3.16) has a unique solution because the matrix $(x_k)_{k \in S}$ has full column rank. To save computational costs, we find $\mathcal{C}(\{1, \dots, p\}, \emptyset)$ in advance and reuse it. If the intersection of all linearly dependent sets of a given dataset is $\emptyset$, then it is sufficient to find $\mathcal{C}(\{1, \dots, p\}, \emptyset)$. In fact, it contains all linearly dependent sets in the given dataset. Otherwise, the linear system may yield infinitely many solutions even after application of the second property of Proposition 3.3.1 with $\mathcal{C}(\{1, \dots, p\}, \emptyset)$ to $R(Z_1, Z_0, Z)$. In this case, we alternate between executing Algorithm 5 and applying the second property.

Algorithm 5: An algorithm to find a collection of linearly dependent sets

```
Input: vectors $x_j$ ($j \in Z \cup Z_1$)
Output: A collection $\mathcal{C}(Z, Z_1)$ of linearly dependent sets
$\mathcal{C}(Z, Z_1) \longleftarrow \emptyset$, $S \longleftarrow \emptyset$;
for $j \in Z \cup Z_1$ do
    if the vectors $\{x_k : k \in S \cup \{j\}\}$ are linearly independent then
        $S \longleftarrow S \cup \{j\}$;
    else
        Solve the following linear system:
            $$\sum_{k \in S} \alpha_k x_k = x_j \qquad (3.3.16)$$
        $S' \longleftarrow \{k \in S : \alpha_k \neq 0\} \cup \{j\}$, $\mathcal{C}(Z, Z_1) \longleftarrow \mathcal{C}(Z, Z_1) \cup \{S'\}$;
    end
end
return $\mathcal{C}(Z, Z_1)$
```

Finally, we prove Proposition 3.3.1 as follows:

Proof. (*First property of Proposition 3.3.1*). Let m_Q be the optimal value of $Q(Z_1, Z_0, Z)$ and m_P the optimal value of the problem (3.1.1)–(3.1.3). It is sufficient to prove that $m_Q > m_P$. An optimal solution of $Q(Z_1, Z_0, Z)$ is denoted by $(\tilde{\beta}, \tilde{z}) \in \mathbb{R}^p \times \mathbb{R}^p$. Considering the assumption of this proposition, for $s \in S$, there exists $\hat{\beta} \in \mathbb{R}^p$ such that

$$\hat{\beta}_j = \tilde{\beta}_j \ (j \in \{1, \dots, p\} \setminus S), \ \hat{\beta}_s = 0 \ \textit{and} \ f(\tilde{\beta}) = f(\hat{\beta}).$$

We define $\hat{z} = (\hat{z}_1, \ldots, \hat{z}_p)^T \in \{0,1\}^p$ as follows:

$$\hat{z}_j = \begin{cases} \tilde{z}_j, & (\text{if } j \neq s) \\ 0, & (\text{if } j = s) \end{cases}$$

for all $j = 1, \ldots, p$. Because $\tilde{z}_s$ is one and $(\hat{\beta}, \hat{z})$ is feasible for (3.1.1)–(3.1.3),

$$m_Q = f(\tilde{\beta}) + \lambda \sum_{j=1}^{p} \tilde{z}_j > f(\hat{\beta}) + \lambda \sum_{j=1}^{p} \hat{z}_j \geq m_P.$$

(*Second property of Proposition 3.3.1*). Let m_R be the optimal value of the relaxation problem $R(Z_1, Z_0, Z)$ and m_{R_k} the optimal value of the relaxation problem $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$ for $k \in Z \cap S$. m_R and m_{R_k} are computed as follows:

$$\begin{aligned} m_R &= \min_{\beta} \{ f(\beta) + \lambda \#(Z_1) : \beta \in \mathbb{R}^p, \beta_j = 0 \ (j \in Z_0) \}, \\ m_{R_k} &= \min_{\beta} \{ f(\beta) + \lambda \#(Z_1) : \beta \in \mathbb{R}^p, \beta_j = 0 \ (j \in Z_0 \cup \{k\}) \}. \end{aligned}$$

Because an optimal solution of the relaxation problem $R(Z_1, Z_0 \cup \{k\}, Z \backslash \{k\})$ is feasible for the relaxation problem $R(Z_1, Z_0, Z)$, $m_{R_k} \geq m_R$ is satisfied. Let $\tilde{\beta}$ be an optimal solution of the relaxation problem $R(Z_1, Z_0, Z)$. Considering the assumption of this proposition, there exists $\hat{\beta} \in \mathbb{R}^p$ such that

$$\hat{\beta}_j = \tilde{\beta}_j \ (j \in \{1, \ldots, p\} \backslash S), \ \hat{\beta}_k = 0 \ \textit{and} \ f(\tilde{\beta}) = f(\hat{\beta}).$$

Because $\hat{\beta}$ is feasible for the relaxation problem $R(Z_1, Z_0, \cup\{k\}, Z \backslash \{k\})$, $m_R \geq m_{R_k}$ is satisfied. Hence $m_R = m_{R_k}$. □

3.4 Numerical experiments

In this section, we show numerical experiments pertaining to AIC minimization for linear regression (Application 1 described in Section 3.3.1) and logistic regression (Application 2 described in Section 3.3.2) and compare the branch-and-bound algorithm described in Section 3.2 with other approaches. We use benchmark datasets from the UCI Machine Learning Repository [14] and standardize the datasets to have zero mean and unit variance. The branch-and-bound algorithm for Application 1 compares with stepwise methods, an MISOCP approach (Section 3.4.1) and an MIQP approaches (Section 3.4.2). In Section 3.4.3, we compare the branching rules described in Section 3.2.3 with the inference branching [8, Section 5.8]. In Section 3.4.4, the branch-and-bound algorithm for Application 2 compares with stepwise methods, a piecewise linear approximation approach. In Section 3.4.5, we examine which of the proposed techniques is effective and how our heuristic method and branching rule influence changes in upper and lower bounds of the optimal value.

We describe tables and figures, which are numerical results of each numerical experiment, in Section 3.4.6. The columns labeled "*n*," "*p*," and "*k*" indicate the number of data points, candidates for explanatory variables, and selected explanatory variables, respectively. The column labeled "AIC" indicates the computed AIC value. The AIC values in bold font are the lowest among applied approaches. The column labeled "Time (sec)" indicates CPU time in seconds to compute the optimal value. “>5000” implies that the corresponding approach could not determine the optimal value within 5000 seconds. The column labeled “Nodes” indicates the number of generated branch-and-bound nodes. The column labeled “Gap (%)” indicates the optimality gap used in SCIP, and it is defined as

$$\text{Gap} = \frac{|\text{upper bound} - \text{lower bound}|}{\min\{|\text{upper bound}|, |\text{lower bound}|\}} \times 100.$$

The benchmark datasets with the mark “•” in Tables 3.1, 3.2 and 3.3, indicate that these datasets have linearly dependent variables.

The specifications of the computers used in each numerical experiment are as follows:

(Sections 3.4.1, 3.4.2 and 3.4.2) CPU: 3.5 GHz Intel Core i7; Memory: 16GB; and OS: OS X 10.9.5.

(Sections 3.4.4 and 3.4.5) CPU: Intel® Xeon® CPU E5–2687 @ 3.1GHz; Memory: 128GB; and OS: Ubuntu 16.04.3 LTS.

3.4.1 Application 1: Comparison with stepwise methods and MISOCP approach

In this subsection, we compare the branch-and-bound algorithm developed in Section 3.2 with stepwise method and an MISOCP approach proposed in [47]. The stepwise methods often find good statistical models via goodness-of-fit measures such as AIC, and these are implemented in statistical software (e.g., R [53]). In each step, the stepwise methods decide whether to add an explanatory variable to the statistical model or to remove it. Although these methods are considered as local search algorithms, they often find good statistical models within a short time. The MISOCP approach [47] transforms the problem (3.3.5)–(3.3.7) into a MISOCP problem, which is tractable by high standard optimization software (e.g., CPLEX [4]). See [47] for the details.

Table 3.1 in Section 3.4.6 shows a comparison of the performance of the following approaches:

- MINLP:
  - refers to the developed branch-and-bound algorithm for solving the MINLP problem (3.3.5)–(3.3.7),
  - employs SCIP [5, 8, 56] for the implementation,
  - adopts the most frequent branching for datasets with linearly dependent variables and the strong branching for datasets with no such variables,
  - uses a single thread.
- MISOCP:
  - refers to the MISOCP approach [47] with CPLEX [4],
  - employs 8 threads for parallel computation.
- SW_+:
  - refers to the stepwise method with forward selection,

  - starts with no explanatory variables,
  - is implemented in R [53].

- SW_-:
  - refers to the stepwise method with backward elimination,
  - starts with all explanatory variables,
  - is implemented in R [53].

We observe the following from Table 3.1.

- MINLP computed the optimal values much faster than MISOCP. MINLP found lower AIC values than MISOCP even when MINLP could not find the optimal values within 5000 seconds.
- MINLP outperformed the other approaches in terms of solution quality. In fact, each AIC value obtained by MINLP were lowest among the four values.
- Either SW_+ or SW_- found low AIC values expect for `automobile`.

3.4.2 Application 1: Comparison with MIQP approaches

In this subsection, we compare the branch-and-bound algorithm developed in Section 3.2 with MIQP approaches. AIC minimization for linear regression can be divided into $(p+1)$ MIQP problems by fixing the number of selected explanatory variables from 0 to p. In fact, the minimization can be equivalently reformulated as follows:

$$\min_{k=0,\dots,p} \min_{S\subseteq\{1,\dots,p\}} \{\text{AIC}(S) : \#(S) = k\}. \tag{3.4.1}$$

Each inner problem in (3.4.1) is defined by fixing $\sum_{j=1}^{p} z_j = k$ in (3.3.5)–(3.3.7). Because the logarithm function in the inner problem has the monotonicity, we can obtain the optimal value and an optimal solution of the inner problem by solving the following MIQP problem:

$$\begin{aligned}
\min_{\beta,z} \quad & \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 && (3.4.2)\\
\text{s.t.} \quad & \sum_{j=1}^{p} z_j = k && (3.4.3)\\
& z_j = 0 \Rightarrow \beta_j = 0 \ (j = 1,\dots,p), && (3.4.4)\\
& \beta_j \in \mathbb{R},\ z_j \in \{0,1\} \ (j = 1,\dots,p), && (3.4.5)
\end{aligned}$$

for all $k = 0,\dots,p$. We denote the optimal value of (3.4.2)–(3.4.5) by η_k^*. Then, the optimal value of the inner problem in (3.4.1) with k is $n\log(\eta_k^*) + 2k$. Therefore, we obtain the optimal value and an optimal solution of (3.4.1) by computing all optimal values of (3.4.2)–(3.4.5) for $k = 0,\dots,p$. We describe this naive procedure in Algorithm 6.

Algorithm 6: Naive algorithm for (3.3.5)–(3.3.7)

```
Input: AIC minimization for linear regression (3.3.5)–(3.3.7)
Output: The optimal value and an optimal solution of (3.3.5)–(3.3.7)
for $k \to 0$ to $p$ do
    Compute the optimal value $\eta_k^*$ and an optimal solution $(\beta_k^*, z_k^*)$ of (3.4.2)–(3.4.5)
      with $k$;
end
Find an index $K$ with $\theta_K^* = \min_{k=0,\dots,p} \{n \log(\eta_k^*) + 2k\}$;
return $\theta_K^*$ and $(\beta_K^*, z_K^*)$;
```

Next, we explain a faster algorithm than Algorithm 6. This algorithm was proposed previously [38], and it finds an upper bound of k. Hence, it is not necessary to solve all $p+1$ MIQP problems. The following lemma ensures that we can find an upper bound of k if we have a feasible solution of (3.3.5)–(3.3.7).

Lemma 3.4.1. *Let $\hat{\theta} \in \mathbb{R}^{p+1}$ be the optimal value of the following optimization problem:*

$$\min_{\beta \in \mathbb{R}^p} \; n \log \left(\sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right). \tag{3.4.6}$$

Given an upper bound $\bar{\theta}$ of an optimal solution of (3.3.5)–(3.3.7), any optimal solution (β^*, z^*) of (3.3.5)–(3.3.7) satisfies

$$\sum_{j=1}^{p} z_j^* \le \left\lfloor \frac{\bar{\theta} - \hat{\theta}}{2} \right\rfloor.$$

Proof. Let θ^* be the optimal value of (3.3.5)–(3.3.7) and (β^*, z^*) an optimal solution of (3.3.5)–(3.3.7). Then, we have

$$\bar{\theta} \ge \theta^* = n \log \left(\sum_{i=1}^{n} \left(y_i - \beta_0^* - \sum_{j=1}^{p} \beta_j^* x_{ij} \right)^2 \right) + 2 \sum_{j=1}^{p} z_j^* \ge \hat{\theta} + 2 \sum_{j=1}^{p} z_j^*,$$

and thus we have $\sum_{j=1}^{p} z_j^* \le (\bar{\theta} - \hat{\theta})/2$. Since z_j^* is integer, we obtain the desired result. $\square$

We describe the faster algorithm based on Lemma 3.4.1 in Algorithm 7.

```
Algorithm 7: Faster algorithm for (3.3.5)–(3.3.7) via MIQP
  Input: AIC minimization for linear regression (3.3.5)–(3.3.7)
  Output: The optimal value and an optimal solution of (3.3.5)–(3.3.7)
  Compute the optimal value $\hat{\theta}$ of (3.4.6);
  $\bar{\theta} \longleftarrow +\infty$;
  for $k \to 0$ to $p$ do
      if $k > \left\lfloor \frac{\bar{\theta}-\hat{\theta}}{2} \right\rfloor$ then
          break;
      end
      Find the optimal value $\eta_k^*$ and an optimal solution $(\beta_k^*, z_k^*)$ of (3.4.2)–(3.4.5) with $k$;
      if $\bar{\theta} \geq n\log(\eta_k^*) + 2k$ then
          $\bar{\theta} \longleftarrow n\log(\eta_k^*) + 2k$, $(\beta^*, z^*) \longleftarrow (\beta_k^*, z_k^*)$;
      end
  end
  return $\bar{\theta}$ and $(\beta^*, z^*)$;
```

We give details on our numerical experiments.

- □ We solve MIQP problems (3.4.2)–(3.4.5) with CPLEX [4]. In particular, since the constraints (3.4.4) are the logical relation between z_j and β_j, we use *indicator* implemented in CPLEX to represent these constraints.
- □ We also solve MIQP problems obtained from replacing the constraint $\sum_{j=1}^{p} z_j = k$ of (3.4.2)–(3.4.5) by $\sum_{j=1}^{p} z_j \leq k$. In Table 3.2, "Fast≤" indicates that we solve those obtained problems in Algorithm 7, while "Fast=" indicates that we solve (3.4.2)–(3.4.5). By this replacement, we can use an optimal solution of the MIQP problem with k as an initial upper bound for the MIQP problem with $k+1$.
- □ We terminate the corresponding approaches when they cannot find the lowest AIC value within 5000 seconds. Computational time in bold font indicates the shortest time among four approaches.

We show numerical results on the branch-and-bound algorithm (MINLP) described in Section 3.2, Algorithms 6 (Naive) and Algorithm 7 (Fast= and Fast≤) in Table 3.2 of Section 3.4.6. We observe the following from Table 3.2:

- MINLP outperformed the MIQP approaches. In particular, for larger p, MINLP obtains much better AIC values than MIQP approaches, although all the approaches cannot solve within 5000 seconds.
- The performance of Fast≤ is similar to Fast=, though Fast≤ uses an initial upper bound.

3.4.3 Application 1: Comparison of branching rules

Table 3.3 in Section 3.4.6 shows a comparison of the performance of the following branching rule:

- IB: the inference branching [8, Section 5.8] implemented in SCIP [5, 9, 56].

- MFB: the most frequent branching described in Section 3.2.3.
- SB: the strong branching described in Section 3.2.3.

The benchmark datasets with the mark "•" in Tables 3.3 indicate that these dataset have linearly dependent variables. The values in bold font are the best among the three branching rules. We observe the following from Table 3.3:

- The most frequent branching worked more effective than the others for the benchmark datasets with linearly dependent variables. In fact, except for `automobile`, MFB computed the optimal value the fastest of the three, and the numbers of the nodes generated by MFB were the smallest. In contrast, the strong branching rule was more effective than the others for the benchmark datasets with no linearly dependent variables.

This is why we adopted the most frequent branching for datasets with linearly dependent variables and the strong branching for datasets with no such variables in Tables 3.1 and 3.2.

3.4.4 Application 2: Comparison with stepwise methods and piecewise linear approximation approach

In this subsection, we show numerical experiments pertaining to AIC minimization for logistic regression and compare the branch-and-bound algorithm described in Section 3.2 with stepwise methods and a piecewise linear approximation approach [48]. This approach solve a MILP problems, and the greatest advantage of it is that commercial optimization software (e.g., CPLEX [4]) can be used to solve the MILP problem. This approach can be applied to AIC minimization for logistic regression (i.e., the problem (3.3.11)–(3.3.13)).

We briefly explain the piecewise linear approximation approach to solving the problem (3.3.11)–(3.3.13). For a given dataset $(x_{i1}, \dots, x_{ip}, y_i) \in \mathbb{R}^p \times \{0,1\}$ with $x_{i1} = 1$ $(i = 1, \dots, n)$, we define sets I_1 and I_2 as follows:

$$I_1 = \{i \in \{1, \dots, n\} : y_i = 1\} \text{ and } I_0 = \{i \in \{1, \dots, n\} : y_i = 0\}.$$

The function $F(\beta, z)$ denotes the objective function (3.3.11), and it can be rewritten as follows:

$$\begin{aligned} F(\beta, z) &:= 2\sum_{i=1}^{n} \left(\log\left(1 + \exp\left(\beta^T x^i\right)\right) - y_i \beta^T x^i\right) + 2\sum_{j=1}^{p} z_j \\ &= 2\sum_{i\in I_1} \left(\log\left(1 + \exp\left(\beta^T x^i\right)\right) - \beta^T x^i\right) + 2\sum_{i\in I_0} \log\left(1 + \exp\left(\beta^T x^i\right)\right) + 2\sum_{j=1}^{p} z_j \\ &= 2\sum_{i\in I_1} \log\left(1 + \exp\left(-\beta^T x^i\right)\right) + 2\sum_{i\in I_0} \log\left(1 + \exp\left(\beta^T x^i\right)\right) + 2\sum_{j=1}^{p} z_j. \end{aligned}$$

We define the function $g(v)$ as $g(v) := \log\left(1 + \exp\left(-v\right)\right)$ and rewrite it as

$$F(\beta, z) = 2\sum_{i\in I_1} g(\beta^T x^i) + 2\sum_{i\in I_0} g(-\beta^T x^i) + 2\sum_{j=1}^{p} z_j.$$

By introducing auxiliary variables $t_i (i = 1, \ldots, n)$, the problem (3.3.11)–(3.3.13) can be reformulated as follows:

$$\min_{\beta,z} \quad 2\sum_{i=1}^{n} t_i + 2\sum_{j=1}^{p} z_j \tag{3.4.7}$$

$$\text{s.t.} \quad t_i \geq g(\beta^T x^i) \ (i \in I_1), \ t_i \geq g(-\beta^T x^i) \ (i \in I_0), \tag{3.4.8}$$

$$z_j = 0 \Rightarrow \beta_j = 0, \ \beta_j \in \mathbb{R}, \ z_j \in \{0,1\} \ (j = 1, \ldots, p). \tag{3.4.9}$$

Given any set of points $V = \{v_1, \ldots, v_K\}$, we can construct a relaxation problem of (3.4.7)–(3.4.9) by using the convexity of g

$$\min_{\beta,z} \quad 2\sum_{i=1}^{n} t_i + 2\sum_{j=1}^{p} z_j \tag{3.4.10}$$

$$\text{s.t.} \quad t_i \geq g'(v_k)(\beta^T x^i - v_k) + g(v_k) \ (i \in I_1; \ v_k \in V), \tag{3.4.11}$$

$$t_i \geq -g'(v_k)(\beta^T x^i + v_k) + g(v_k) \ (i \in I_0; \ v_k \in V), \tag{3.4.12}$$

$$z_j = 0 \Rightarrow \beta_j = 0, \ \beta_j \in \mathbb{R}, \ z_j \in \{0,1\} \ (j = 1, \ldots, p). \tag{3.4.13}$$

The problem (3.4.10)–(3.4.13) is a mixed integer linear programming problem, and it can be solved by using standard optimization software. The optimal value $\bar{\theta}$ of (3.4.10)–(3.4.13) is a lower bound of the optimal value θ^* of (3.3.11)–(3.3.13). Let $(\bar{\beta}, \bar{z}, \bar{t})$ be an optimal solution of (3.4.10)–(3.4.13). We can construct the logistic regression model from the set of the selected explanatory variables $\bar{S} = \{j \in \{1, \ldots, p\} : \bar{z}_j = 1\}$. Then, the AIC value of the constructed model is $\text{AIC}(\bar{S})$. Hence, we obtain the following inequality:

$$\bar{\theta} \leq \theta^* \leq \text{AIC}(\bar{S}).$$

If $\text{AIC}(\bar{S}) - \bar{\theta}$ is small, the constructed model is guaranteed to be of good quality.

In the numerical experiments, we employ the following two sets as V,

$$V_1 = \{0, \pm 0.89, \pm 1.90, \pm 3.55, \pm\infty\},$$
$$V_2 = \{0, \pm 0.44, \pm 0.89, \pm 1.37, \pm 1.90, \pm 2.63, \pm 3.55, \pm 5.16, \pm\infty\}.$$

These sets can be computed by using the greedy algorithm proposed in [48].

Table 3.4 in Section 3.4.6 shows a comparison of the performance of the following approaches:

- MINLP:
  - refers to the developed branch-and-bound algorithm for solving the MINLP problem (3.3.11)–(3.3.13),
  - employs SCIP [5, 8, 56] and UG [7, 51] for the implementation,
  - adopts the most frequent branching described in Section 3.2.3,
  - uses 16 threads for parallel computation.
- SW_+:
  - refers to the stepwise method with forward selection,

  - starts with no explanatory variables,
  - is implemented by C++ and LAPACK [13].

- SW$_-$:
  - refers to the stepwise method with backward elimination,
  - starts with all explanatory variables,
  - is implemented by C++ and LAPACK [13].

- MILP(V):
  - refers to the piecewise linear approximation approach [48] with the point set V,
  - solves the mixed integer linear programming problem (3.4.10)–(3.4.13) with CPLEX [4],
  - employs the better of the two solutions of the stepwise methods as the initial solution,
  - employs 16 threads for parallel computation.

The column labeled "obj$_{\text{MILP}}$" in Table 3.4 presents the objective value of the computed solution of the mixed integer linear programming problem (3.4.10)–(3.4.13).

It can be inferred from Table 3.4 that MINLP outperforms MILP(V) in terms of computational time. In fact, for $p \leq 45$, MINLP was faster than both the MILP(V). Moreover, MINLP found the lowest AIC values of the five approaches on large-scale instances. However, for $p \geq 62$, even MINLP could not guarantee optimum within 5000 seconds.

3.4.5 Application 2: Computational performance of developed techniques

To examine which of the proposed techniques is effective, we present the computational performance of the following approaches:

- MINLP:
  - executes the most frequent branching described in Section 3.2.3,
  - executes the heuristic method described in Section 3.2.2,
  - constructs the initial feasible solution from an optimal solution of the relaxation problem of the parent node,
  - executes the procedure described in Section 3.3.2 to construct the initial guess for Newton's method.

- MINLP$_{\text{w/o-mfb}}$:
  - corresponds to MINLP without the most frequent branching,
  - executes the inference branching in SCIP.

- MINLP$_{\text{w/o-heur}}$: corresponds to MINLP without the heuristic method.

- MINLP$_{\text{w/o-guess}}$:
  - corresponds to MINLP without the initial guess,

– employs the zero vector as a initial feasible solution.

To indicate the effective techniques, we underline the highest values among all the methods in Table 3.5 of Section 3.4.6. We observe the following from Table 3.5:

- For $p \leq 45$, MINLP, that is, the developed solver incorporating all techniques, was the fastest among the four methods. This implies that the most frequent branching, the heuristic method based on the stepwise methods, and the initial guess are effective for solving (3.3.11)–(3.3.13).
- MINLP and $\text{MINLP}_{\text{w/o-guess}}$ could solve AIC minimization for `spectf` within 5000 seconds. However, $\text{MINLP}_{\text{w/o-mfb}}$ and $\text{MINLP}_{\text{w/o-heur}}$ could not solve the minimization within 5000 seconds. Hence, the most frequent branching and the heuristic method based on the stepwise methods are more effective than the initial guess in this instance.
- For $p \geq 62$, $\text{MINLP}_{\text{w/o-heur}}$ were the worst among the four methods in terms of solution quality. Hence, it is evident from this result that the heuristic method described in Section 3.2.2 is an important technique for large-scale instances.

Next, we examine how the heuristic method (Section 3.2.2) and the most frequent branching (Section 3.2.3) influence changes in the upper and lower bounds. Figure 3.1 shows the results of the upper bounds for `biodeg` and `spectf`. The solid and the broken lines correspond to our solver with and without the heuristic method based on the stepwise methods (i.e., MINLP and $\text{MINLP}_{\text{w/o-heur}}$), respectively. Our solver with the heuristic method immediately found good feasible solutions compared to the solver without the heuristic method. Figure 3.2 shows the results of the lower bounds for `biodeg` and `spectf`. The solid and the broken lines correspond to our solver with and without the most frequent branching (i.e., MINLP and $\text{MINLP}_{\text{w/o-mfb}}$), respectively. Our solver without the most frequent branching appears to stop increases in the lower bounds halfway. The benefit of using the most frequent branching can be confirmed from Figure 3.2.

3.4.6 Tables and figures of numerical experiments

Table 3.1: Comparison with stepwise methods (SW_+ and SW_-) and MISOCP approach

Name	n	p	Approaches	AIC	k	Time (sec)	Gap (%)
`housing`	506	13	MINLP	**776.21**	11	0.04	0.00
			MISOCP	**776.21**	11	7.96	0.00
			SW_+	**776.21**	11	0.35	—
			SW_-	**776.21**	11	0.10	—
•`auto-mpg`	392	25	MINLP	**332.88**	15	1.76	0.00
			MISOCP	**332.88**	15	303.83	0.00
			SW_+	334.73	16	0.49	—
			SW_-	337.96	18	0.32	—
•`solarflareC`	1066	26	MINLP	**2816.29**	9	10.49	0.00
			MISOCP	**2816.29**	9	304.51	0.00
			SW_+	**2816.29**	9	0.45	—
			SW_-	2821.61	12	1.08	—
•`solarflareM`	1066	26	MINLP	**2926.90**	7	3.99	0.00
			MISOCP	**2926.90**	7	255.02	0.00
			SW_+	**2926.90**	7	0.36	—
			SW_-	2930.91	9	1.16	—
•`solarflareX`	1066	26	MINLP	**2882.80**	3	0.92	0.00
			MISOCP	**2882.80**	3	19.39	0.00
			SW_+	**2882.80**	3	0.18	—
			SW_-	2891.56	9	1.20	—
`breastcancer`	194	32	MINLP	**508.40**	10	90.21	0.00
			MISOCP	508.62	10	>5000	3.72
			SW_+	509.50	8	0.24	—
			SW_-	509.96	14	0.60	—
•`forestfires`	517	63	MINLP	**1429.64**	12	>5000	0.77
			MISOCP	1431.32	12	>5000	6.44
			SW_+	**1429.64**	12	0.94	—
			SW_-	1447.36	21	7.43	—
•`automobile`	159	65	MINLP	**-61.28**	32	>5000	13.95
			MISOCP	-55.83	34	>5000	27.22
			SW_+	-28.55	21	1.12	—
			SW_-	-47.61	40	2.64	—
`crime`	1993	100	MINLP	**3410.25**	50	>5000	0.50
			MISOCP	3469.34	74	>5000	8.51
			SW_+	3430.19	37	17.03	—
			SW_-	**3410.25**	50	105.40	—

Table 3.2: Comparison with three MIQP approaches: Algorithm 6 (Naive), Algorithm 7 (Fast= and Fast≤)

Name	Approaches	AIC	k	Time (sec)
`housing`	MINLP	**776.21**	11	**0.04**
	Naive	**776.21**	11	2.54
	Fast=	**776.21**	11	2.15
	Fast≤	**776.21**	11	2.43
•`auto-mpg`	MINLP	**332.88**	15	**1.76**
	Naive	**332.88**	15	22.22
	Fast=	**332.88**	15	19.04
	Fast≤	**332.88**	15	14.45
•`solarflareC`	MINLP	**2816.29**	9	**10.49**
	Naive	**2816.29**	9	26.49
	Fast=	**2816.29**	9	18.17
	Fast≤	**2816.29**	9	15.03
•`solarflareM`	MINLP	**2926.90**	7	**3.99**
	Naive	**2926.90**	7	25.27
	Fast=	**2926.90**	7	8.15
	Fast≤	**2926.90**	7	7.24
•`solarflareX`	MINLP	**2882.80**	3	**0.92**
	Naive	**2882.80**	3	10.65
	Fast=	**2882.80**	3	2.25
	Fast≤	**2882.80**	3	2.40
`breastcancer`	MINLP	**508.40**	10	**90.21**
	Naive	**508.40**	10	420.44
	Fast=	**508.40**	10	402.64
	Fast≤	**508.40**	10	421.96
•`forestfires`	MINLP	**1429.64**	12	>5000
	Naive	1435.07	7	>5000
	Fast=	1435.07	7	>5000
	Fast≤	1435.07	7	>5000
•`automobile`	MINLP	**-61.28**	32	>5000
	Naive	52.84	8	>5000
	Fast=	52.84	8	>5000
	Fast≤	52.84	8	>5000
`crime`	MINLP	**3410.25**	50	>5000
	Naive	3646.35	4	>5000
	Fast=	3646.35	4	>5000
	Fast≤	3646.35	4	>5000

Table 3.3: Comparison of three branching rule: inference branching (IB), most frequent branching (MFB) and strong branching (SB)

Name	Branching rules	AIC	k	Time (sec)	Nodes	Gap (%)
`housing`	IB	**776.21**	11	0.05	55	0.00
	MFB	**776.21**	11	0.05	49	0.00
	SB	**776.21**	11	**0.04**	**27**	0.00
●`auto-mpg`	IB	**332.88**	15	4.06	18959	0.00
	MFB	**332.88**	15	**1.76**	**5723**	0.00
	SB	**332.88**	15	2.68	11586	0.00
●`solarflareC`	IB	**2816.29**	9	53.33	166639	0.00
	MFB	**2816.29**	9	**10.49**	**32261**	0.00
	SB	**2816.29**	9	23.13	79015	0.00
●`solarflareM`	IB	**2926.90**	7	40.03	117889	0.00
	MFB	**2926.90**	7	**3.99**	**11903**	0.00
	SB	**2926.90**	7	23.72	81899	0.00
●`solarflareX`	IB	**2882.80**	3	4.37	9737	0.00
	MFB	**2882.80**	3	**0.92**	**1519**	0.00
	SB	**2882.80**	3	3.40	7453	0.00
`breastcancer`	IB	**508.40**	10	505.70	3851×10^3	0.00
	MFB	**508.40**	10	478.66	3422×10^3	0.00
	SB	**508.40**	10	**90.21**	**550**$\times10^3$	0.00
●`forestfires`	IB	**1429.64**	12	>5000	7480×10^3	1.11
	MFB	**1429.64**	12	>5000	13179×10^3	**0.77**
	SB	**1429.64**	12	>5000	9938×10^3	0.95
●`automobile`	IB	-60.29	32	>5000	32192×10^3	**12.30**
	MFB	-61.28	32	>5000	29785×10^3	13.95
	SB	**-61.59**	33	>5000	15300×10^3	16.43
`crime`	IB	**3410.25**	50	>5000	10272×10^3	0.78
	MFB	**3410.25**	50	>5000	9753×10^3	0.52
	SB	**3410.25**	50	>5000	1904×10^3	**0.50**

Table 3.4: Comparison with stepwise methods and piecewise linear approximation approach

Name	n	p	Approaches	AIC	obj_{MILP}	k	Time (sec)	Gap (%)
`bumps`	2584	22	MINLP	**1097.11**	—	9	20.08	0.00
			SW_+	1097.37	—	9	0.92	—
			SW_-	1100.66	—	13	0.54	—
			MILP(V_1)	1098.12	1060.51	8	41.51	0.00
			MILP(V_2)	1099.98	1086.43	9	627.36	0.00
`breast-P`	194	34	MINLP	**147.04**	—	19	25.76	0.00
			SW_+	162.94	—	13	0.24	—
			SW_-	152.13	—	25	0.25	—
			MILP(V_1)	**147.04**	144.56	19	112.40	0.00
			MILP(V_2)	**147.04**	146.40	19	279.15	0.00
`biodeg`	1055	42	MINLP	**653.29**	—	23	221.54	0.00
			SW_+	654.79	—	25	2.01	—
			SW_-	**653.29**	—	23	2.25	—
			MILP(V_1)	**653.29**	640.75	23	>5000	0.93
			MILP(V_2)	**653.29**	649.62	23	>5000	2.39
`spectf`	267	45	MINLP	**168.33**	—	15	432.45	0.00
			SW_+	172.34	—	10	0.36	—
			SW_-	169.42	—	17	0.79	—
			MILP(V_1)	169.34	163.54	14	515.74	0.00
			MILP(V_2)	169.34	165.53	14	1603.12	0.00
`stat-G`	1000	62	MINLP	**958.15**	—	24	>5000	5.54
			SW_+	**958.15**	—	24	3.09	—
			SW_-	963.70	—	29	2.55	—
			MILP(V_1)	**958.15**	944.50	24	>5000	5.21
			MILP(V_2)	**958.15**	954.46	24	>5000	5.10
`musk`	6598	166	MINLP	**1706.89**	—	115	>5000	16.55
			SW_+	1733.56	—	120	292.18	—
			SW_-	**1706.89**	—	115	609.44	—
			MILP(V_1)	**1706.89**	1663.02	115	>5000	16.68
			MILP(V_2)	**1706.89**	1693.28	115	>5000	16.39
`madelon`	2000	500	MINLP	**2502.06**	—	105	>5000	20.76
			SW_+	2504.02	—	102	316.92	—
			SW_-	2905.58	—	422	>5000	—
			MILP(V_1)	2504.02	2471.93	102	>5000	20.20
			MILP(V_2)	2504.02	2493.70	102	>5000	22.85

Table 3.5: Computational performance of developed techniques

Name	n	p	Approaches	AIC	k	Time (sec)	Nodes	Gap (%)
`bumps`	2584	22	MINLP	1097.11	9	20.08	3.6×10^3	0.00
			MINLP$_{\text{w/o-mfb}}$	1097.11	9	44.99	2.2×10^4	0.00
			MINLP$_{\text{w/o-heur}}$	1097.11	9	28.68	$\underline{2.3 \times 10^4}$	0.00
			MINLP$_{\text{w/o-guess}}$	1097.11	9	$\underline{46.48}$	4.1×10^3	0.00
`breast-P`	194	34	MINLP	147.04	19	25.76	1.5×10^5	0.00
			MINLP$_{\text{w/o-mfb}}$	147.04	19	$\underline{554.07}$	$\underline{3.3 \times 10^6}$	0.00
			MINLP$_{\text{w/o-heur}}$	147.04	19	31.87	4.6×10^5	0.00
			MINLP$_{\text{w/o-guess}}$	147.04	19	27.38	1.5×10^5	0.00
`biodeg`	1055	42	MINLP	653.29	23	221.54	1.7×10^5	0.00
			MINLP$_{\text{w/o-mfb}}$	653.29	23	$\underline{>5000}$	$\underline{8.8 \times 10^6}$	$\underline{4.53}$
			MINLP$_{\text{w/o-heur}}$	653.29	23	1018.83	2.5×10^6	0.00
			MINLP$_{\text{w/o-guess}}$	653.29	23	586.45	1.9×10^5	0.00
`spectf`	267	45	MINLP	168.33	15	432.45	1.1×10^6	0.00
			MINLP$_{\text{w/o-mfb}}$	168.33	15	$\underline{>5000}$	$\underline{1.1 \times 10^7}$	29.89
			MINLP$_{\text{w/o-heur}}$	$\underline{171.80}$	17	$\underline{>5000}$	$\underline{1.1 \times 10^7}$	$\underline{34.53}$
			MINLP$_{\text{w/o-guess}}$	168.33	15	574.13	1.5×10^5	0.00
`stat-G`	1000	62	MINLP	958.15	24	>5000	7.7×10^6	5.54
			MINLP$_{\text{w/o-mfb}}$	958.15	24	>5000	6.5×10^6	6.11
			MINLP$_{\text{w/o-heur}}$	$\underline{978.67}$	30	>5000	5.5×10^6	$\underline{7.61}$
			MINLP$_{\text{w/o-guess}}$	958.15	24	>5000	$\underline{8.9 \times 10^6}$	4.62
`musk`	6598	166	MINLP	1706.89	115	>5000	3.5×10^4	16.55
			MINLP$_{\text{w/o-mfb}}$	1705.01	111	>5000	5.7×10^4	16.87
			MINLP$_{\text{w/o-heur}}$	$\underline{1774.54}$	161	>5000	$\underline{6.4 \times 10^5}$	$\underline{20.18}$
			MINLP$_{\text{w/o-guess}}$	1706.89	115	>5000	2.1×10^4	17.19
`madelon`	2000	500	MINLP	2502.06	105	>5000	1.0×10^6	20.76
			MINLP$_{\text{w/o-mfb}}$	2503.58	105	>5000	1.1×10^6	21.15
			MINLP$_{\text{w/o-heur}}$	$\underline{3028.85}$	455	>5000	$\underline{2.4 \times 10^6}$	$\underline{46.70}$
			MINLP$_{\text{w/o-guess}}$	2502.06	105	>5000	8.3×10^5	20.76

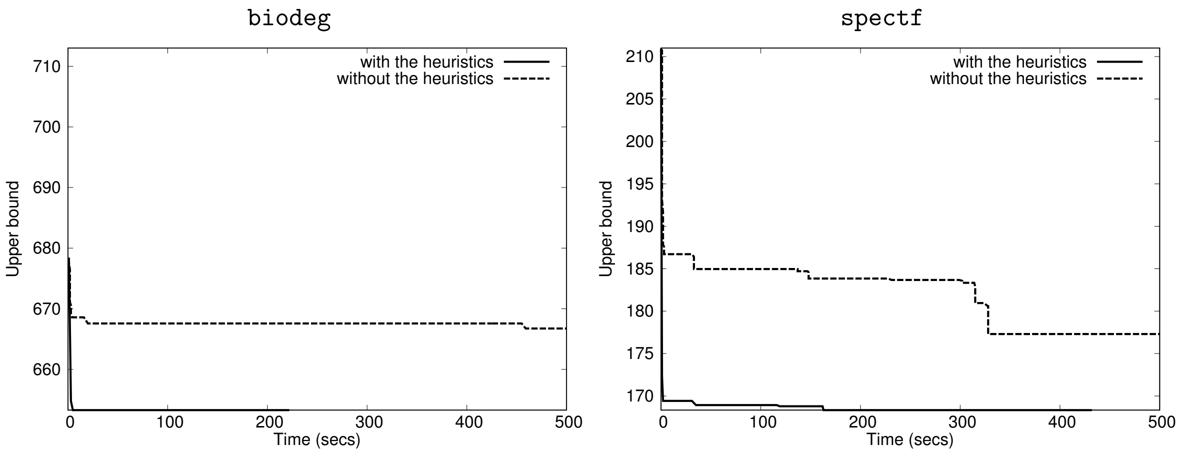

Figure 3.1: Evolution of upper bounds in the first 500 seconds, for `biodeg` and `spectf` when using our solver with and without our heuristic method

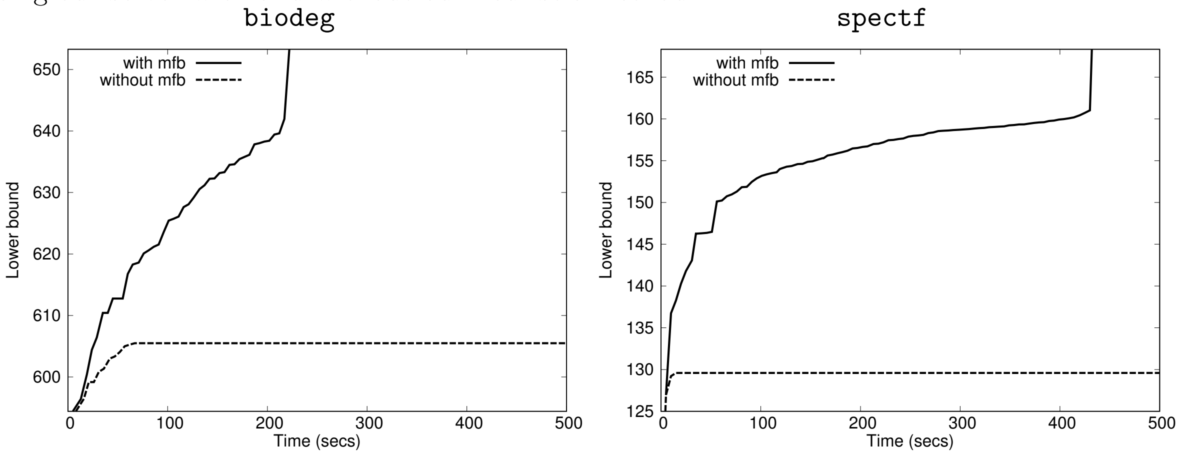

Figure 3.2: Evolution of lower bounds in the first 500 seconds, for `biodeg` and `spectf` when using our solver with and without the most frequent branching

3.5 Appendix

As mentioned in Section 3.3.2, the MINLP formulation (3.3.11)–(3.3.13) for logistic regression may not satisfy Assumption 1. Therefore, here, we provide a necessary and sufficient condition to ensure that the MINLP formulation (3.3.11)–(3.3.13) for logistic regression satisfies Assumption 1. First, we introduce notation and symbols. For a dataset $(x^i, y_i) \in \mathbb{R}^p \times \{0,1\}$ ($i = 1,\dots,n$), we define the sets I_1 and I_0 as

$$I_1 = \{i \in \{1,\dots,n\} : y_i = 1\} \text{ and } I_0 = \{i \in \{1,\dots,n\} : y_i = 0\}.$$

We rewrite the objective function of the minimization (3.1.4) as

$$f(\beta) = \sum_{i\in I_0} \log\left(1+\exp(\beta^T x^i)\right) + \sum_{i\in I_1} \log\left(1+\exp(-\beta^T x^i)\right).$$

For $\beta \in \mathbb{R}^p$, we define the sets $J_+(\beta), J_-(\beta)$, and $J_0(\beta)$ as

$$J_+(\beta) = \{i \in \{1,\dots,n\} : \beta^T x^i > 0\},\ J_-(\beta) = \{i \in \{1,\dots,n\} : \beta^T x^i < 0\} \text{ and}$$
$$J_0(\beta) = \{i \in \{1,\dots,n\} : \beta^T x^i = 0\}.$$

Then, we have $J_\bullet(\gamma\beta) = J_\bullet(\beta)$ for $\gamma > 0$ and $\bullet \in \{+,-,0\}$. For any $\gamma > 0$ and $\beta \in \mathbb{R}^p$, we have

$$\begin{aligned} f(\gamma\beta) &= \sum_{i\in I_0} \log\left(1+\exp(\gamma\beta^T x^i)\right) + \sum_{i\in I_1} \log\left(1+\exp(-\gamma\beta^T x^i)\right) \\ &= \sum_{i\in I_0\cap J_+(\beta)} \log\left(1+\exp(\gamma\beta^T x^i)\right) + \sum_{i\in I_0\cap J_-(\beta)} \log\left(1+\exp(\gamma\beta^T x^i)\right) \\ &+ \sum_{i\in I_1\cap J_+(\beta)} \log\left(1+\exp(-\gamma\beta^T x^i)\right) + \sum_{i\in I_1\cap J_-(\beta)} \log\left(1+\exp(-\gamma\beta^T x^i)\right) \\ &+ \#(J_0(\beta))\log(2). \end{aligned} \tag{3.5.1}$$

It follows from the following theorem that Assumption 1 holds when the necessary and sufficient condition in the theorem holds.

Theorem 3.5.1. *The minimization (3.1.4) has an optimal solutions for any nonempty subset $S \subseteq \{1,\dots,p\}$ if and only if for any $\beta \in \mathbb{R}^p \setminus \{0\}$, $I_0 \cap J_+(\beta)$ or $I_1 \cap J_-(\beta)$ is nonempty.*

Proof. For simplicity, we fix $S = \{1,\dots,p\}$ for (3.1.4). First, we prove the if part. We fix $\beta \in \mathbb{R}^p$ so that $\|\beta\| = 1$. Then, by taking $\gamma \to \infty$, each term in (3.5.1) satisfies

$$\sum_{i\in I_0\cap J_+(\beta)} \log\left(1+\exp(\gamma\beta^T x^i)\right) \to +\infty, \quad \sum_{i\in I_0\cap J_-(\beta)} \log\left(1+\exp(\gamma\beta^T x^i)\right) \to 0,$$
$$\sum_{i\in I_1\cap J_+(\beta)} \log\left(1+\exp(-\gamma\beta^T x^i)\right) \to 0, \quad \sum_{i\in I_1\cap J_-(\beta)} \log\left(1+\exp(-\gamma\beta^T x^i)\right) \to +\infty.$$

Because we have assumed that $I_0 \cap J_+(\beta)$ or $I_1 \cap J_-(\beta)$ is nonempty, there exists $M > 0$ such that the objective function $f(\beta)$ takes sufficiently large values for all β so that $\|\beta\| > M$. Hence, the minimum solution of (3.1.4) is in the circle $\|\beta\| \leq M$. Therefore, (3.1.4) has an optimal solution.

Next, we prove the only-if part. We assume that there exists $\beta \in \mathbb{R}^p \setminus \{0\}$ such that both $I_0 \cap J_+(\beta)$ and $I_1 \cap J_-(\beta)$ are empty. It is sufficient to prove that (3.1.4) has a finite optimal value but no optimal solutions. It follows from the definition of $f(\beta)$ that $f(\beta) > \#(J_0(\beta)) \log(2)$ for all $\beta \in \mathbb{R}^p \setminus \{0\}$. In addition, from the proof of the if-part, by taking $\gamma \to \infty$, we have $g(\gamma\beta) \to \#(J_0(\beta)) \log(2)$. This is the desired result. $\square$

Chapter 4

Shortest Vector Problem

4.1 Overview

Given n linearly independent integer vectors $b_1, \ldots, b_n \in \mathbb{Z}^n$, we define the $n \times n$ matrix $B = (b_1, \ldots, b_n)$ and formulate the shortest vector problem (SVP) mathematically as follows:

$$\theta_0 := \min_x \left\{ \|Bx\|_2^2 : x = (x_1, \ldots, x_n)^T \in \mathbb{Z}^n, x \neq 0 \right\}, \tag{SVP}$$

where $\|\cdot\|_2$ is the 2-norm. SVP has been generally researched in the field of cryptology. In this chapter, we apply a branch-and-bound algorithm, which has been developed in the field of mathematical optimization, to SVP, and propose integer quadratic programming (IQP) approaches using high standard optimization software (e.g., CPLEX [4]) to solve SVP efficiently. Moreover, we combine the branch-and-bound algorithm with techniques developed in cryptology, for example, the BKZ algorithm [49] and a property of the Gram-Schmidt orthogonalization [22, Section 3.3]. We examine how large SVPs can be solved experimentally with the proposed algorithm and approaches.

In Section 4.2, we customize a branch-and-bound algorithm to solve (SVP) efficiently. This algorithm consists of the following techniques:

(Section 4.2.1) computation of bounds on variables in (SVP),

(Section 4.2.2) branching that splits a feasible region into three sets,

(Section 4.2.3) relaxation using the Gram-Schmidt orthogonalization,

(Section 4.2.4) heuristic method to find good feasible solutions.

We cannot directly solve (SVP) via standard optimization software (e.g., CPLEX [4]) because it has the constraint $x \neq 0$. In Section 4.3, we propose the following IQP approaches using high standard optimization software to solve (SVP):

(Section 4.3.1) convex 0 - 1 IQP approach,

(Section 4.3.2) Partitioning approach.

In Section 4.4, we show numerical experiments of the proposed algorithm and approaches and compare the computational performance of them.

4.2 Customized branch-and-bound algorithm

In this section, we proposed a branch-and-bound algorithm purpose-built for (SVP). In Section 4.2.1, we explain presolve to compute upper and lower bounds on variables in (SVP). In Section 4.2.2, we describe branching for (SVP). In Section 4.2.3, we compute a lower bound of an optimal value of a subproblem with the constraint $x \neq 0$ by using the Gram-Schmidt orthogonalization. In Section 4.2.4, we propose a heuristic method for finding better feasible solutions of a subproblem.

4.2.1 Computation of bounds on variables

Presolve (including root and node presolve) for MILP and MINLP is one of important components of a branch-and-bound algorithm, and it includes the following techniques: tightening a given feasible region and removing redundant constraints and variables. Such various techniques has been proposed to accelerate the speed of a branch-and-bound algorithm. See [8, 10, 15] for more details. In [39, 40], we proposed root presolve, which computes upper and lower bounds on variables in (SVP). In this subsection, we explain root presolve based on [39, 40].

Given m linearly independent integer vectors $\bar{b}_1, \dots, \bar{b}_m \in \mathbb{Z}^n$ $(n \geq m)$, we consider the following problem:

$$\bar{\theta}_0 := \min_x \left\{ \left\| \bar{B}x \right\|_2^2 : x \neq 0, x \in \mathbb{Z}^m \right\}, \tag{4.2.1}$$

where $\bar{B} \in \mathbb{Z}^{n \times m}$ is denoted by $(\bar{b}_1, \dots, \bar{b}_m)$. Note that (4.2.1) is the form of (SVP) if $n = m$. Let $M(> 0) \in \mathbb{Z}$ be an upper bound of the optimal value $\bar{\theta}_0$ of (4.2.1). For instance, we can employ $\min\{\|\bar{b}_1\|_2^2, \dots, \|\bar{b}_m\|_2^2\}$ as M because any standard basis vector e_i of $\mathbb{R}^m$, which has a single nonzero entry with the value 1, is feasible for (4.2.1). Because an optimal solution $x^* \in \mathbb{Z}^m$ of (4.2.1) satisfies $\|\bar{B}x^*\|_2^2 \leq M$, upper and lower bounds on variables x_i can be computed as follows:

$$\min_{x \in \mathbb{Z}^m} \left\{ x_i : \left\| \bar{B}x \right\|_2^2 \leq M \right\} \leq x_i \leq \max_{x \in \mathbb{Z}^m} \left\{ x_i : \left\| \bar{B}x \right\|_2^2 \leq M \right\}, \tag{4.2.2}$$

for all $i = 1, \dots, m$. Because these problems contain integer variables, the computational cost for solving them is high. Therefore, we compute the following optimization problems instead of (4.2.2):

$$\min_{x \in \mathbb{R}^m} \left\{ x_i : \left\| \bar{B}x \right\|_2^2 \leq M \right\} \leq x_i \leq \max_{x \in \mathbb{R}^m} \left\{ x_i : \left\| \bar{B}x \right\|_2^2 \leq M \right\}, \tag{4.2.3}$$

for all $i = 1, \dots, m$. We can compute these problems by applying the following lemma.

Lemma 4.2.1. *Given a scalar $M(> 0) \in \mathbb{R}$, vectors $t \in \mathbb{R}^m$, $d \in \mathbb{R}^n$, and a full column matrix $A \in \mathbb{R}^{n \times m} (n \geq m)$, the following equations hold:*

$$\min_{x \in \mathbb{R}^m} \left\{ t^T x : \|Ax + d\|_2^2 \leq M \right\} = -\sqrt{M} \left\| \tilde{A}^T t \right\|_2 - t^T \tilde{A} d,$$
$$\max_{x \in \mathbb{R}^m} \left\{ t^T x : \|Ax + d\|_2^2 \leq M \right\} = \sqrt{M} \left\| \tilde{A}^T t \right\|_2 - t^T \tilde{A} d,$$

where $\tilde{A}$ is defined as $(A^T A)^{-1} A^T$.

Proof. By defining $y \in \mathbb{R}^n$ as $(1/\sqrt{M})(Ax+d)$, we can rewrite above problems as follows:

$$\min_{x\in\mathbb{R}^m}\left\{t^Tx : \|Ax+d\|_2^2 \le M\right\} = -t^T\tilde{A}d + \sqrt{M}\min_{y\in\mathbb{R}^n}\left\{t^T\tilde{A}y : \|y\|_2^2 \le 1\right\},$$
$$\max_{x\in\mathbb{R}^m}\left\{t^Tx : \|Ax+d\|_2^2 \le M\right\} = -t^T\tilde{A}d + \sqrt{M}\max_{y\in\mathbb{R}^n}\left\{t^T\tilde{A}y : \|y\|_2^2 \le 1\right\}.$$

From the Cauchy-Schwarz inequality, we have $-\|\tilde{A}^Tt\|_2 \le t^T\tilde{A}y \le \|\tilde{A}^Tt\|_2$ for any $y \in \mathbb{R}^n$ with $\|y\|_2^2 \le 1$. Here, $y^- \in \mathbb{R}^n$ denotes $-(1/\|\tilde{A}^Tt\|_2)\tilde{A}^Tt$, and $y^+ \in \mathbb{R}^n$ denotes $(1/\|\tilde{A}^Tt\|_2)\tilde{A}^Tt$. Then, y^- and y^+ are the unit vectors. Because $t^T\tilde{A}y^- = -\|\tilde{A}^Tt\|_2$ and $t^T\tilde{A}y^+ = \|\tilde{A}^Tt\|_2$, y^- and y^+ are optimal solutions of above problems, respectively. □

Applying Lemma 4.2.1, we can compute (4.2.3) as follows:

$$-\sqrt{M}\left\|\bar{B}\left(\bar{B}^T\bar{B}\right)^{-1}e_i\right\|_2 \le x_i \le \sqrt{M}\left\|\bar{B}\left(\bar{B}^T\bar{B}\right)^{-1}e_i\right\|_2, \tag{4.2.4}$$

for all $i = 1,\dots,m$. In addition, by using the integrality of variables x_i, we can tighten the bounds (4.2.4) as follows:

$$-\left\lfloor\sqrt{M}\left\|\bar{B}\left(\bar{B}^T\bar{B}\right)^{-1}e_i\right\|_2\right\rfloor \le x_i \le \left\lfloor\sqrt{M}\left\|\bar{B}\left(\bar{B}^T\bar{B}\right)^{-1}e_i\right\|_2\right\rfloor,$$

for all $i = 1,\dots,m$.

We apply this presolve to the root node (i.e., (SVP)). An upper bound on variable x_i in (SVP) is denoted by

$$c_i = \left\lfloor\sqrt{M}\left\|B^{-T}e_i\right\|_2\right\rfloor$$

for any $i = 1,\dots,m$. Then, (SVP) with bounds on variables is written as

$$\theta_0 := \min_x\left\{\|Bx\|_2^2 : -c_i \le x_i \le c_i\ (i=1,\dots,n), x \ne 0, x \in \mathbb{Z}^n\right\}. \tag{SVP$'$}$$

4.2.2 Branching

We explain branching for (SVP′) in this subsection, and it is used in our branch-and-bound algorithm. By selecting index k from among $\{1,\dots,n\}$ and adding trivial constraints to (SVP′), we obtain the following subproblems:

$$\theta_k^0 := \min_x\left\{\|Bx\|_2^2 : -c_i \le x_i \le c_i\ (i=1,\dots,n), x_k = 0, x \ne 0, x \in \mathbb{Z}^n\right\}, \tag{4.2.5}$$
$$\theta_k^+ := \min_x\left\{\|Bx\|_2^2 : -c_i \le x_i \le c_i\ (i=1,\dots,n), 1 \le x_k, x \ne 0, x \in \mathbb{Z}^n\right\}, \tag{4.2.6}$$
$$\theta_k^- := \min_x\left\{\|Bx\|_2^2 : -c_i \le x_i \le c_i\ (i=1,\dots,n), x_k \le -1, x \ne 0, x \in \mathbb{Z}^n\right\}. \tag{4.2.7}$$

It is clear that the optimal value of θ_0 is equal to $\min\{\theta^0,\theta^+,\theta^-\}$. The following lemma implies that we do not need to solve the problem (4.2.7). Therefore, it is enough to focus on (4.2.5) and (4.2.6).

Lemma 4.2.2. $\theta_0 = \min\{\theta_k^0,\theta_k^+\}$ *for any* $k \in \{1,\dots,n\}$.

Proof. Let x^+ be an optimal solution of (4.2.6) and x^- an optimal solution of (4.2.7). $-x^+$ is feasible for (4.2.7) and we obtain $\|B(-x^+)\|_2^2 = \|Bx^+\|_2^2 = \theta_k^+ \ge \theta_k^-$. $-x^-$ is feasible for (4.2.6) and we obtain $\|B(-x^-)\|_2^2 = \|Bx^-\|_2^2 = \theta_k^- \ge \theta_k^+$. Hence we obtain $\theta_k^+ = \theta_k^-$. □

First, we consider branching for (4.2.5). As in branching for (SVP'), we select index k' from among $\{1,\dots,n\} \setminus \{k\}$ and generate two subproblems: (4.2.5) with the equality $x_{k'} = 0$ and (4.2.5) with the inequality $1 \le x_{k'}$. Next, we consider branching for (4.2.6). The constraint $x \neq 0$ in (4.2.6) is redundant. By removing $x \neq 0$, we can transform (4.2.6) into the following problem:

$$\theta_k^+ = \min_x \left\{ \|Bx\|_2^2 : -c_i \le x_i \le c_i, 1 \le x_k, x \in \mathbb{Z}^n \right\}. \tag{4.2.8}$$

This problem is a convex IQP problem. A relaxation problem which is obtained by relaxing integrality of all variables x_i, can be solved by applying conventional algorithms (e.g., the interior point method and the active set method). Therefore, we execute standard branching for (4.2.8). Let $\bar{x}$ be an optimal solution of the relaxation problem of (4.2.8). If $\bar{x}$ is an integer point, $\bar{x}$ is also an optimal solution of (4.2.8). Then, we do not need to branch for (4.2.8). Otherwise, we select an index k' with a fractional value $\bar{x}_{k'}$, and generate two subproblems: (4.2.8) with the inequality $x_{k'} \le \lfloor \bar{x}_{k'} \rfloor$ and (4.2.8) with the inequality $\lceil \bar{x}_{k'} \rceil \le x_{k'}$.

We summarize branching for any subproblem in Algorithm 8. Any subproblem is written as

$$\min_x \left\{ \|Bx\|_2^2 : \ell_i \le x_i \le u_i \ (i = 1,\dots,n), x \neq 0, x \in \mathbb{Z}^n \right\}, \tag{4.2.9}$$

where ℓ_i and u_i for all $i = 1,\dots,n$ are integers.

Algorithm 8: Branching

```
Input: A feasible subproblem (4.2.9)
Output: Two subproblems
$S \leftarrow \{i \in \{1,\dots,n\} : \ell_i \neq 0$ or $u_i \neq 0\}$;
if $\ell_i < 0$ and $u_i > 0$ for all $i \in S$ then
    // Then, we apply branching for (4.2.5).
    Select an index $k \in S$ and return two subproblems:
        • (4.2.9) with $x_k = 0$
        • (4.2.9) with $x_k \ge 1$
else
    // Then, we apply branching for (4.2.6).
    // Let $\bar{x}$ be an optimal solution of the relaxation problem of (4.2.9)
    $S \leftarrow \{i \in \{1,\dots,n\} : \bar{x}_i$ is a fractional value$\}$;
    Select index $k \in S$ and return two subproblems:
        • (4.2.9) with $x_k \le \lfloor \bar{x}_k \rfloor$
        • (4.2.9) with $x_k \ge \lceil \bar{x}_k \rceil$
end
```

4.2.3 Relaxation

We explain a procedure to compute a lower bound of the optimal value of any subproblem (4.2.9). If the constraint $x \neq 0$ of a given subproblem (4.2.9) is redundant, the relaxation problem obtained by relaxing the integrality is convex quadratic programming problem, which can be solved by applying conventional algorithms, (e.g., the interior point method and the active set method). Our branch-and-bound algorithm applies the active set method to such a subproblem and employs an optimal solution of the parent subproblem as an initial solution. If the constraint $x \neq 0$ of (4.2.9) is not redundant, the set of feasible solutions of the relaxation problem is nonconvex, and it is intractable. In this case, instead of the relaxation problem, we employ a proposition described in [22, Section 3.3] to compute a lower bound of the optimal value of the subproblem.

We consider the following problem:

$$\theta := \min_{x} \left\{ \left\| \bar{B}x \right\|_2^2 : x \neq 0,\ x \in \mathbb{Z}^m \right\}, \tag{4.2.10}$$

where $\bar{B} \in \mathbb{Z}^{n\times m}$ is a submatrix of $B \in \mathbb{Z}^{n\times n}$ in (SVP′). Let $\bar{b}_1, \ldots, \bar{b}_m$ be the column vectors of $\bar{B}$. The Gram-Schmidt orthogonalization $b_1^*, \ldots, b_m^*$ of $\bar{b}_1, \ldots, \bar{b}_m$ is computed as follows:

$$\begin{aligned} b_1^* &= \bar{b}_1, \\ b_i^* &= \bar{b}_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^* \ (i = 2, \ldots, m), \quad \mu_{ij} = \frac{\bar{b}_i \cdot b_j^*}{b_j^* \cdot b_j^*} \ (1 \leq j < i \leq m), \end{aligned}$$

where dot notation "·" denotes the dot product of vectors. We do not normalize the vectors. By using the following proposition described in [22, Section 3.3], we can compute a lower bound of the optimal value of (4.2.10).

Proposition 4.2.3. $\theta \geq \min\{\|b_j^*\|_2^2 : j = 1, \ldots, m\} > 0$

Proof. Let x^* be an optimal solution of (4.2.10) and k the largest index with $x_k^* \neq 0$. Then, the optimal value of (4.2.10) is $\|\bar{B}x^*\|_2^2$. First, we prove $|\bar{B}x^* \cdot b_k^*| \geq \|b_k^*\|_2^2$. Using the definition of the Gram-Schmidt orthogonalization, we obtain

$$\bar{B}x^* \cdot b_k^* = \sum_{i=1}^{k} \left(\bar{b}_i \cdot b_k^*\right) x_i^* = \left(\bar{b}_k \cdot b_k^*\right) x_k^* = \left(\left(b_k^* + \sum_{j=1}^{k-1} \mu_{kj} b_j^* \right) \cdot b_k^* \right) x_k^* = x_k^* \|b_k^*\|_2^2.$$

Since x_k^* is a nonzero integer, we have $|x_k^*| \geq 1$, and so

$$\left|\bar{B}x^* \cdot b_k^*\right| = |x_k^*| \|b_k^*\|_2^2 \geq \|b_k^*\|_2^2.$$

From the Cauchy-Schwarz inequality, we obtain

$$\left\|\bar{B}x^*\right\|_2 \|b_k^*\|_2 \geq \left|\bar{B}x^* \cdot b_k^*\right| \geq \|b_k^*\|_2^2.$$

Therefore, the following equation holds:

$$\left\|\bar{B}x^*\right\|_2 \geq \|b_k^*\|_2 \geq \min\left\{\|b_j^*\|_2 : j = 1, \ldots, m\right\} > 0.$$

□

We can compute a lower bound of the optimal value of any subproblem by applying Proposition 4.2.3 even if one contains the constraint $x \neq 0$. Any subproblem is written as

$$\min_{x} \left\{ \|Bx\|_2^2 : \ell_i \leq x_i \leq u_i \ (i = 1, \ldots, n), x \neq 0, x \in \mathbb{Z}^n \right\},$$

where ℓ_i and u_i for all $i = 1, \dots, n$ are integers. If there exist $i \in \{1, \dots, n\}$ such that $\ell_i = u_i = 0$, we substitute $x_i = 0$ with all such i into the subproblem. Then, the subproblem is rewritten as

$$\min_x \left\{ \left\|\bar{B}x\right\|_2^2 : \ell_i \le x_i \le u_i \ (i = 1, \dots, m), x \neq 0, x \in \mathbb{Z}^m \right\},$$

where $\bar{B} \in \mathbb{Z}^{n \times m}$ is a submatrix of $B \in \mathbb{Z}^{n \times n}$. By relaxing the bound constraints $\ell_i \le x_i \le u_i$ for all $i = 1, \dots, m$, this subproblem is of the form of (4.2.10). Therefore, we can compute a lower bound of the optimal value of the subproblem. However, since the bound constraints are relaxed, the computed lower bounds may not be good value.

4.2.4 Heuristic method

We can find a feasible solution of a subproblem (4.2.9) by rounding an optimal solution of the relaxation problem. However, such a solution does not always give a low objective value. In this section, we propose a heuristic method for finding a better feasible solution of the following subproblem:

$$\min_x \left\{ \|Bx\|_2^2 : \ell_i \le x_i \le u_i \ (i = 1, \dots, n), \ x \in \mathbb{Z}^n \right\}, \tag{4.2.11}$$

where ℓ_i and u_i $(i = 1, \dots, n)$ are integers. We assume that the zero vector is infeasible for (4.2.11). Given a feasible solution $\hat{x}$ of (4.2.11), we improve repeatedly the feasible solution $\hat{x}$ by solving the following problems:

$$\min_t \left\{ \|B\left(\hat{x} + te_i\right)\|_2^2 : \ell_i \le \hat{x}_i + t \le u_i, t \in \mathbb{Z} \right\}, \tag{4.2.12}$$

for all $i = 1, \dots, n$, where $e_i \in \mathbb{R}^n$ is the n-dimensional ith standard basis vector. Let t_i^* $(i = 1, \dots, n)$ be optimal solutions of these problems (4.2.12), respectively. This optimization is to find the best solution $\hat{x} + t_i^* e_i$ with respect to the ith element of the solution $\hat{x}$. We solve (4.2.12) for all $i = 1, \dots, n$ and select an index i^* so that the objective value is the lowest in all the optimal values. If the value is less than $\|B\hat{x}\|_2^2$, then we change from $\hat{x}$ to $\hat{x} + t_i^* e_i$ as the current solution. We repeat this procedure until the objective value is not improved. We summarize this process in Algorithm 9. This heuristic method can be regarded as the coordinate descent method for an IQP problem (4.2.11). See [45, Section 8.9] for the detail of the coordinate descent method.

Algorithm 9: Heuristic method for subproblems (4.2.11)

```
Input: A subproblem (4.2.11) and an initial feasible solution $x$ of (4.2.11)
Output: A feasible solution $\hat{x}$ of (4.2.11)
$\hat{x}_{new} \longleftarrow x$;
do
    $\hat{x} \longleftarrow \hat{x}_{new}$;
    for $i \to 1$ to $n$ do
        Solve (4.2.12) and $t_i^*$ denotes an optimal solution of (4.2.12);
        $v_i \longleftarrow \|B(\hat{x} + t_i^* e_i)\|_2^2$;
    end
    $i^* \longleftarrow \arg\min\{v_i : i = 1, \dots, n\}$, $\hat{x}_{new} \longleftarrow \hat{x} + t_{i^*}^* e_{i^*}$;
while $\|B\hat{x}_{new}\|_2^2 < \|B\hat{x}\|_2^2$;
return $\hat{x}$;
```

We can easily solve the problems (4.2.12). To explain this, we define $\hat{\ell}_i$ and $\hat{u}_i$ as follows:

$$\hat{\ell}_i = \ell_i - \hat{x}_i \text{ and } \hat{u}_i = u_i - \hat{x}_i.$$

By removing the constant term $\|B\hat{x}\|_2^2$ from the objective function we simplify (4.2.12) as follows:

$$\min_t \{t^2 \|Be_i\|_2^2 + 2te_i{}^T B^T B\hat{x} : \hat{\ell}_i \le t \le \hat{u}_i, t \in \mathbb{Z}\} \tag{4.2.13}$$

The following lemma ensures that an optimal solution of (4.2.13) is provided with the closed-form expression. By using this lemma, we can find t^* in Algorithm 9 efficiently.

Lemma 4.2.4. *Let $\hat{t}$ be $-\hat{x}^T B^T Be_i / \|Be_i\|_2^2$. An optimal solution t^* of* (4.2.13) *is provided as follows.*

Case1 *If $\hat{\ell}_i \le \hat{t} \le \hat{u}_i$, then t^* is either $\lceil \hat{t} \rceil$ or $\lfloor \hat{t} \rfloor$.*

Case2 *If $\hat{t} < \hat{\ell}_i$, then $t^* = \hat{\ell}_i$.*

Case3 *If $\hat{t} > \hat{u}_i$, then $t^* = \hat{u}_i$.*

Proof. The objective function of (4.2.13) is denoted by $g_i(t)$. $\hat{t}$ is the optimal solution of the minimization of $g_i(t)$ over $t \in \mathbb{R}$. Since $g_i(t)$ is convex, we have the following monotonicity:

$$g_i(t) \ge g_i(\lfloor \hat{t} \rfloor) \ge g_i(\hat{t}) \ (t \le \lfloor \hat{t} \rfloor \le \hat{t}) \text{ and } g_i(\hat{t}) \le g_i(\lceil \hat{t} \rceil) \le g_i(t) \ (t \ge \lceil \hat{t} \rceil \ge \hat{t}).$$

In Case1, since $\hat{\ell}_i$ and $\hat{u}_i$ are both integer, $\hat{\ell}_i \le \lfloor \hat{t} \rfloor \le \lceil \hat{t} \rceil \le \hat{u}_i$. In addition, there is no integer between $\lfloor \hat{t} \rfloor$ and $\lceil \hat{t} \rceil$. Thus either $\lfloor \hat{t} \rfloor$ or $\lceil \hat{t} \rceil$ is optimal for (4.2.13) in Case1. We can prove Case2 and Case3 by using this monotonicity on $g_i(t)$. □

4.3 IQP approaches using optimization software

In Section 4.2, we proposed the branch-and-bound algorithm purpose-built for (SVP). To examine the computational performance of this algorithm, we propose two convex IQP approaches using high standard optimization software (e.g., CPLEX [4]) in this section. The optimization software cannot directly be applied to (SVP) because it contains the constraint $x \neq 0$. Hence, we need to cope with $x \neq 0$. The first approach reformulates (SVP) as a convex 0 - 1 IQP problem. The constraint $x \neq 0$ is transformed into equality and inequality constraints in Section 4.3.1. The second approach is a combination of Proposition 4.2.3 and high standard optimization software. We explain the process in Section 4.3.2. In Section 4.4, we compare these approaches with the branch-and-bound algorithm proposed in Section 4.2.

4.3.1 Convex 0 - 1 IQP approach

We proposed a convex 0 - 1 IQP formulation for SVP in [40]. In this subsection, we introduce firstly this formulation. Next, we propose another formulation that contains less auxiliary variables. In both of the formulations, we apply the techniques proposed in Section 4.2.1 to compute upper and lower bounds on variables x_i in (SVP). Hence, we focus on (SVP) with the bounds, that is,

$$\theta_0 := \min_x \left\{ \|Bx\|_2^2 : -c_i \le x_i \le c_i \ (i = 1, \ldots, n), x \neq 0, x \in \mathbb{Z}^m \right\}, \tag{SVP'}$$

in this subsection.

By adding binary variables y_{ij} and constraints to (SVP), we transform $x \neq 0$ into equality and inequality constraints as follows:

$$\theta^0 = \min_{x,y} \left\{ \|Bx\|_2^2 : \begin{array}{l} x_i = \sum_{j=-c_i}^{c_i} j y_{ij},\ \sum_{j=-c_i}^{c_i} y_{ij} = 1\ (i = 1, \ldots, n), \\ \sum_{i=1}^{n} y_{i0} \leq n-1, \\ x \in \mathbb{Z}^n, y_{ij} \in \{0,1\}\ (i = 1, \ldots, n; j = -c_i, -c_i + 1 \ldots, c_i) \end{array} \right\}. \tag{4.3.1}$$

The constraint $\sum_{j=-c_i}^{c_i} y_{ij} = 1$ means that the only one variable y_{ij} is 1 and that the others are 0. Thus, $x_i = k$ holds if $y_{ik} = 1$. The constraint $\sum_{i=1}^{n} y_{i0} \leq n-1$ ensures that any feasible solution x is not the zero vector.

Next, we explain another formulation for SVP. Given (SVP'), we define $k_i \in \mathbb{Z}$ for all $i = 1, \ldots, n$ as

$$k_i = \min\{k \in \mathbb{Z} : c_i \leq 2^k, k \geq 0\},$$

where c_i for all $i = 1, \ldots, n$ are the bounds in (SVP') and positive integers. By introducing integer and binary variables, (SVP') can be reformulated as follows:

$$\theta^0 = \min_{x,x^\pm,y^\pm,z^\pm} \left\{ \|Bx\|_2^2 : \begin{array}{l} x_i = x_i^+ - x_i^-\ (i = 1, \ldots, n), \\ x_i^+ = \sum_{j=0}^{k_i} 2^j y_{ij}^+,\ x_i^- = \sum_{j=0}^{k_i} 2^j y_{ij}^-\ (i = 1, \ldots, n), \\ \sum_{i=1}^{n} (x_i^+ + x_i^-) \geq 1, \\ z_i^+ + z_i^- = 1\ (i = 1, \ldots, n), \\ -c_i \leq x_i \leq c_i\ (i = 1, \ldots, n), \\ 0 \leq x_i^+ \leq c_i z_i^+,\ 0 \leq x_i^- \leq c_i z_i^-\ (i = 1, \ldots, n), \\ x, x^+, x^- \in \mathbb{Z}^n, z^+, z^- \in \{0,1\}^n, \\ y_{ij}^+, y_{ij}^- \in \{0,1\}\ (i = 1, \ldots, n;\ j = 1, \ldots, k_i) \end{array} \right\}. \tag{4.3.2}$$

The variables x_i^+ and x_i^- correspond to positive and negative components of x_i, respectively. The constraints $z_i^+ + z_i^- = 1$ ensure that either z_i^+ or z_i^- is 0. If $z_i^+ = 0$, then $x_i^+ = 0$ because of $x_i^+ \leq c_i z_i^+$. The constraint $\sum_{i=1}^{n} (x_i^+ + x_i^-) \geq 1$ corresponds to $x \neq 0$. In fact, this constraints is unsatisfied only if $x = 0$.

These problems (4.3.1) and (4.3.2) are convex IQP problems, and standard optimization software (e.g., CPLEX [4]) is available for them. Here, we compare the number of auxiliary variables in each problem. The problem (4.3.1) needs $\sum_{i=1}^{n} (2c_i + 1)$ auxiliary variables. On the other hand, the number of the auxiliary variables of (4.3.2) is $\sum_{i=1}^{n} (2k_i + 4)$ because x^+ and x^- can be eliminated from (4.3.2). For benchmark problems [6] with the lattice dimension $n \geq 40$, most of the c_i are larger than 3. Hence, we expect that (4.3.2) will contain a small number of variables compared to the other formulation (4.3.1). In Section 4.4, we show numerical experiments pertaining to (4.3.2).

4.3.2 Partitioning approach

In this subsection, we propose an effective convex IQP approach without auxiliary variables to solve (SVP'). This approach splits a problem into two problems. The first problem does not contain the constraint $x \neq 0$, and optimization software is available for it. On the other hand, the second problem contains the constraint $x \neq 0$, and optimization software cannot directly be applied to it. Instead of solving the second problem, we use Proposition 4.2.3 for computation of a lower bound of the optimal value of it. The second problem is repeatedly split.

First, we define two problems for any subset $K \subset \{1,\dots,n\}$ and $k' \in \{1,\dots,n\} \setminus K$ as follows:

$$\theta^0\left(K,k'\right) := \min_x \left\{ \|Bx\|_2^2 : \begin{array}{l} -c_i \le x_i \le c_i \ (i=1,\dots,n), \\ x_k = 0 \ (k \in K), \ x_{k'} = 0, \\ x \neq 0, \ x \in \mathbb{Z}^n \end{array} \right\}, \qquad \mathsf{P}(K,k')$$

$$\theta^+\left(K,k'\right) := \min_x \left\{ \|Bx\|_2^2 : \begin{array}{l} -c_i \le x_i \le c_i \ (i=1,\dots,n), \\ x_k = 0 \ (k \in K), \ x_{k'} \ge 1, \\ x \in \mathbb{Z}^n \end{array} \right\}. \qquad \mathsf{Q}(K,k')$$

By applying the branching described in Section 4.2.2, we rewrite (SVP') as

$$\begin{aligned}
\theta_0 &= \min_x \left\{ \|Bx\|_2^2 : -c_i \le x_i \le c_i \ (i=1,\dots,n), x \neq 0, x \in \mathbb{Z}^n \right\} \\
&= \min\left\{\theta^+\left(\emptyset,k_1'\right), \theta^0\left(\emptyset,k_1'\right)\right\} && (4.3.3) \\
&= \min\left\{\theta^+\left(\emptyset,k_1'\right), \min\left\{\theta^+\left(\left\{k_1'\right\},k_2'\right), \theta^0\left(\left\{k_1'\right\},k_2'\right)\right\}\right\} \\
&= \min\left\{\theta^+\left(\emptyset,k_1'\right), \theta^+\left(\left\{k_1'\right\},k_2'\right), \min\left\{\theta^+\left(\left\{k_1',k_2'\right\},k_3'\right), \theta^0\left(\left\{k_1',k_2'\right\},k_3'\right)\right\}\right\} \\
&\ \vdots \\
&= \min\left\{\theta^+\left(\emptyset,k_1'\right), \theta^+\left(\left\{k_1'\right\},k_2'\right), \theta^+\left(\left\{k_1',k_2'\right\},k_3'\right), \dots, \theta^+\left(\left\{k_1',k_2',\dots,k_{n-1}'\right\},k_n'\right)\right\}, && (4.3.4)
\end{aligned}$$

where $k_i' \in \{1,\dots,n\} \setminus \{k_j' : 0 < j < i\}$ for all $i = 1,\dots,n$. Because all problems $\mathsf{Q}(K,k')$ do not contain $x \neq 0$, they can be solved via optimization software. Hence, we can compute the optimal value θ_0 and an optimal solution of (SVP').

The equation (4.3.4) contains n convex IQP problems. We make good use of Proposition 4.2.3 and show that it is not necessary to solve all the problems. Let x^* be an interim solution. Applying Proposition 4.2.3 to $\mathsf{P}(K,k')$, we can obtain a lower bound of $\theta^0\left(K,k'\right)$. For example, if a lower bound ℓ of $\theta^0\left(\emptyset,k_1'\right)$ is higher than $\|Bx^*\|_2^2$ (i.e., $\theta^0\left(\emptyset,k_1'\right) \ge \ell > \|Bx^*\|_2^2$), then $\theta_0 = \min\{\theta^+\left(\emptyset,k_1'\right), \|Bx^*\|_2^2\}$ because of (4.3.3). In this way, the proposed approach splits repeatedly a problem into $\mathsf{P}(K,k')$ and $\mathsf{Q}(K,k')$ and computes the optimal value $\theta^+\left(K,k'\right)$ of $\mathsf{Q}(K,k')$ and a lower bound of the optimal value $\theta^0\left(K,k'\right)$ of $\mathsf{P}(K,k')$. Finally, we summarize the proposed approach in Algorithm 10.

Algorithm 10: Convex IQP approach for (SVP')

```
Input: $(\mathsf{SVP}')$, i.e. $B := (b_1, \dots, b_n) \in \mathbb{Z}^{n \times n}$ and $c_i (\geq 0) \in \mathbb{Z}$ $(i = 1, \dots, n)$
Output: The optimal value and an optimal solution of $(\mathsf{SVP}')$
$K \leftarrow \emptyset$;
$\theta^* \leftarrow \min\{\|b_i\|_2^2 : i = 1, \dots, n\}$ and $x^* \leftarrow e_j$ where $j \in \arg\min\{\|b_i\|_2^2 : i = 1, \dots, n\}$;
for $i \rightarrow 1$ to $n - 1$ do
    Select an index $k \in \{1, \dots, n\} \setminus K$;
    Compute the optimal value $\theta^+(K, k)$ of $\mathsf{Q}(K, k)$ via optimization software;
    Let $\tilde{x}$ be an optimal solution of $\mathsf{Q}(K, k)$;
    if $\theta^* \geq \theta^+(K, k)$ then $\theta^* \leftarrow \theta^+(K, k)$ and $x^* \leftarrow \tilde{x}$;
    if $i < n - 1$ then
        Compute lower bound $\ell$ of $\theta^0(K, k)$ by applying Proposition 4.2.3;
        if $\ell > \theta^*$ then break;
    end
    $K \leftarrow K \cup \{k\}$;
end
return $\theta^*$ and $x^*$;
```

4.4 Numerical experiments

In this section, we show numerical experiments of the proposed approaches, which are the branch-and-bound algorithm described in Section 4.2 and the IQP approaches proposed in Section 4.3. We use benchmark problems obtained from generator in [6] and apply the BKZ algorithm [49] with blocksize 20 to these problems. We employed this BKZ algorithm from the library of fplll [2]. The specifications of the computer used in the numerical experiments were as follows: CPU: Intel® Xeon® CPU E5–2687 @ 3.1GHz; Memory: 128GB; and OS: Ubuntu 16.04.3 LTS.

We solve SVPs with the dimension $n = 40, 43, 46, 49$, and these problems are obtained from generator with the seed $= 0, 1, \dots, 4$. The time limit for solving each SVP is 86400 seconds (= 1 day). Figure 4.1 shows computational time of the following approaches:

- BaB:
    - refers to the branch-and-bound algorithm proposed in Section 4.2,
    - is implemented by C++,
    - employs 32 threads for parallel computation.
- BIQP:
    - solves 0 - 1 convex IQP problem (4.3.2) via CPLEX [4],
    - employs 32 threads for parallel computation.
- IQP:
    - refers to Algorithm 10 proposed in Section 4.3.2,
    - employs CPLEX [4] to solve $\mathsf{Q}(K, k')$,

– employs 32 threads for parallel computation.

In Figure 4.1, the vertical line shows the computational time and horizontal one seed. If the benchmark problem could not be solved within 86400 seconds, the computational time of the corresponding approach is not plotted. It can be inferred from Figure 4.1 that BaB, that is, the branch-and-bound algorithm proposed in Section 4.2, outperforms the others in terms of computational time. In fact, for all the benchmark problem, BaB was the fastest among the three approaches. The second fastest approach was IQP, that is, the convex IQP approach proposed in Section 4.3.2. IQP and BIQP could not solve some of the benchmark problems with $n = 49$ within 86400 seconds.

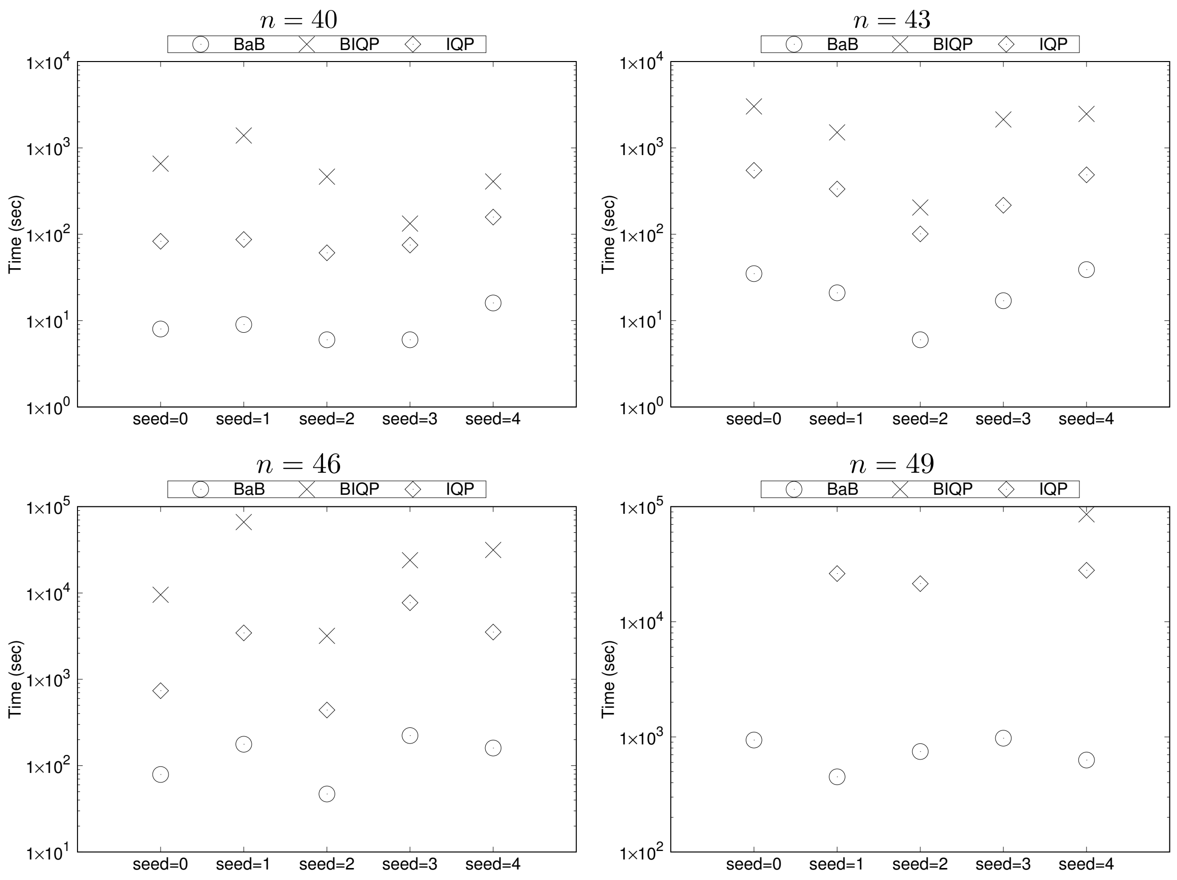

Figure 4.1: Comparison of computational time on benchmark problems with dimension $n \in \{40, 43, 46, 49\}$

Next, we show the computational performance of BaB for benchmark problems with larger dimension in Table 4.1. The column labeled "Time" indicates CPU time in seconds to compute the optimal value. ">86400 s" implies that the branch-and-bound algorithm could not determine the optimal value within 86400 seconds. The column labeled "α" indicates *the approximation factor* defined as

$$\alpha := \frac{\sqrt{\pi}\|Bx^*\|_2}{\Gamma(n/2+1)^{1/n}|\det B|^{1/n}},$$

where x^* is the computed solution, and $\Gamma(n/2+1)$ stands for the value of gamma function at $(n/2+1)$. Notably, the approximation factor is used as a measure of the quality of the computed solution. In [6], it is necessary to find feasible solutions with $\alpha \leq 1.05$.

It can be inferred Table 4.1 that the proposed branch-and-bound algorithm can solve SVPs with the dimension $n \leq 55$ within 86400 seconds. In fact, each of the SVP with $n = 55$ is solved within 4 hours. Although the proposed algorithm could not solve the SVPs with $n = 58$ and seed $= 0, 1$ within 86400 seconds, it could find good feasible solutions.

Table 4.1: Computational performance of our proposed branch-and-bound algorithm

	$n = 52$		$n = 55$		$n = 58$	
seed	Time	α	Time	α	Time	α
0	3542 s	1.02	14063 s	1.02	>86400 s	1.03
1	1966 s	0.99	10841 s	1.01	>86400 s	1.04
2	1739 s	0.99	4163 s	0.95	84087 s	1.03
3	4263 s	1.03	10806 s	1.01	59794 s	1.01
4	2828 s	1.01	11149 s	1.00	63357 s	0.99

Chapter 5

Conclusion

We proposed branch-and-bound algorithms to solve two problems: variable selection and SVP. To achieve good computational performance, we applied the structure and properties of the problems to components related to the branch-and-bound algorithms, such as relaxation, branching (rules), heuristic methods. We showed numerical experiments pertaining to the proposed algorithms and examined how they outperform approaches using state-of-the-art optimization software.

In Chapter 3, we focus on direct objective optimization and formulate it as an MINLP problem. This problem contains a flexible objective function, and applications of it include AIC-based variable selection in linear and logistic regression. To solve the MINLP problem efficiently, we developed several components related to the branch-and-bound algorithm, for example, the cost-effective relaxation, the heuristic method based on stepwise methods, and the branching rules. In the numerical experiments, we implement the branch-and-bound with these components by using SCIP and UG, which are open source software for a flexible framework of an branch-and-bound algorithm and parallel computation. For small-scale and medium-scale instances, our solver was faster than approaches using state-of-the-art optimization software. Conversely, for large-scale instances, our solver could find better solutions compared to the other approaches even if it could not determine the optimal value within 5000 seconds. Nonetheless, there is room for improvement in the numerical performance of our solver. The computational cost of our heuristic method based on the stepwise methods appears to be high for large-scale instances. In fact, SW_+ and SW_- (i.e., the stepwise methods) required considerably more computational time for solving `musk` and `madelon` in Table 3.4. Hence, further study is to reduce the computational time of our heuristic method, for example, by applying discrete first order algorithms [17].

In Chapter 4, we proposed the effective branch-and-bound algorithm purpose-built for SVPs. This algorithm consists of the following components: presolve, branching, relaxation, and a heuristic method. In addition, we proposed two convex IQP approaches using state-of-the-art optimization software to solve SVPs. We compared the proposed branch-and-bounds algorithm with the proposed convex IQP approaches in the numerical experiments. It can be inferred from the numerical experiments that the proposed branch-and-bound algorithm outperforms the others in terms of computational time. To examine difficulty in solving SVPs, we need to solve SVPs with larger dimension n. To this end, we will improve parallel computation of branch-and-bound algorithm (see, e.g., [51]) and incorporate techniques proposed in [23] for convex IQP problems.

Acknowledgements

First and foremost, I would like to show my greatest appreciation to Associate Professor Hayato Waki, who gave me continuing helpful comments and words of encouragement. I would also like to express my gratitude to Professor Katsuki Fujisawa, who lent me high-spec computers for some preliminary computational experiments and provided helpful comments and opportunities for a variety of researches. I would also like to thank Associate Professor Masaya Yasuda for helpful discussions. I would like to offer my special thanks to Doctor Yuji Shinano and Doctor Ambros Gleixner. They let me study in ZIB and supported me in Berlin life. I want to thank past and present members of Fujisawa's laboratory for spending fun time with me. Now lastly, I would like to express my gratitude to my family for their support and warm encouragements.

Bibliography

[1] FICO Xpress-Optimizer, FICO. Home page: `http://www.fico.com/xpress`

[2] fplll, FPLLL development team. fplll home page: `http://github.com/fplll/fplll`

[3] Gurobi, Gurobi Optimization. Gurobi home page: `http://www.gurobi.com`

[4] IBM ILOG CPLEX Optimizer 12.8.0, IBM ILOG. CPLEX home page: `https://www.ibm.com/products/ilog-cplex-optimization-studio`

[5] SCIP Optimization Suite, Zuse Institute Berlin. SCIP home page: `http://scip.zib.de/`

[6] SVP CHALLENGE. Home page: `https://www.latticechallenge.org/svp-challenge/`

[7] Ubiquity Generator framework, Zuse Institute Berlin. UG home page: `http://ug.zib.de/`

[8] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universität Berlin, 2007.

[9] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[10] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve reductions in mixed integer programming. *ZIB Report*, pages 16–44, 2016.

[11] M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 10–19. ACM, 1998.

[12] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[13] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and A. McKenney. *LAPACK Users' guide.* SIAM, 1999.

[14] K. Bache and M. Lichman. UCI machine learning repository, 2013. Home page: `http://archive.ics.uci.edu/ml`

[15] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

[16] D. Bertsimas and A. King. Logistic regression: From art to science. *Statistical Science*, 32(3):367–384, 2017.

[17] D. Bertsimas, A. King, and R. Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, 2016.

[18] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[19] B. Borchers and J. E. Mitchell. An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers & Operations Research*, 21(4):359–367, 1994.

[20] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[21] C. Bragalli, C. D'Ambrosio, J. Lee, A. Lodi, and P. Toth. On the optimal design of water distribution networks: a practical MINLP approach. *Optimization and Engineering*, 13(2):219–246, 2012.

[22] M. R. Bremner. *Lattice Basis Reduction: An Introduction to the LLL Algorithm and Its Applications*. Chapman & Hall Pure and Applied Mathematics. Taylor & Francis, 2011.

[23] C. Buchheim, A. Caprara, and A. Lodi. An effective branch-and-bound algorithm for convex quadratic integer programming. *Mathematical Programming*, 135(1-2):369–395, 2012.

[24] M. R. Bussieck and S. Vigerske. MINLP solver software. In *Wiley Encyclopedia of Operations Research and Management Science*. Wiley, Chichester, 2010.

[25] Ö. Dagdelen and M. Schneider. Parallel enumeration of shortest lattice vectors. In *European Conference on Parallel Processing*, pages 211–222. Springer, 2010.

[26] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965.

[27] T. Farkas, B. Czuczai, and Z. Lelkes. New MINLP model and modified outer approximation algorithm for distillation column synthesis. *Industrial & Engineering Chemistry Research*, 47(9):3088–3103, 2008.

[28] U. Fincke and M. Pohst. A procedure for determining algebraic integers of given norm. In *European Conference on Computer Algebra*, pages 194–202. Springer, 1983.

[29] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of npcompleteness. *Computers and Intractability*, 340, 1979.

[30] O. K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[31] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003.

[32] J. Hermans, M. Schneider, J. Buchmann, F. Vercauteren, and B. Preneel. Parallel shortest lattice vector enumeration on graphics cards. In *International Conference on Cryptology in Africa*, pages 52–68. Springer, 2010.

[33] X. Huo and X. Ni. When do stepwise algorithms meet subset selection criteria? *The Annals of Statistics*, pages 870–887, 2007.

[34] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 193–206. ACM, 1983.

[35] J. Kim, Seung, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):606–617, 2007.

[36] K. Kimura. Application of a mixed integer nonlinear programming approach to variable selection in logistic regression. *Journal of the Operations Research Society of Japan*, 62(1), to appear.

[37] K. Kimura and H. Waki. Mixed integer nonlinear program for minimization of Akaike's information criterion. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – ICMS 2016*, pages 292–300, Cham, 2016. Springer International Publishing.

[38] K. Kimura and H. Waki. Minimization of Akaike's information criterion in linear regression analysis via mixed integer nonlinear program. *Optimization Methods and Software*, 33(3):633–649, 2018.

[39] K. Kimura and H. Waki. A mixed integer quadratic formulation for the shortest vector problem. In *Mathematical Modelling for Next-Generation Cryptography*, pages 239–255. Springer, 2018.

[40] K. Kimura, H. Waki, and M. Yasuda. Application of mixed integer quadratic program to shortest vector problems. *JSIAM Letters*, 9:65–68, 2017.

[41] K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale ℓ_1-regularized logistic regression. *Journal of Machine Learning Research*, 8(Jul):1519–1555, 2007.

[42] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[43] J. Lee and S. Leyffer, editors. *Mixed Integer Nonlinear Programming*, volume 154. Springer Science & Business Media, 2011.

[44] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[45] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming.* International Series in Operations Research & Management Science. Springer US, 2008.

[46] D. Micciancio and M. Walter. Practical, predictable lattice basis reduction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 820–849. Springer, 2016.

[47] R. Miyashiro and Y. Takano. Mixed integer second-order cone programming formulations for variable selection in linear regression. *European Journal of Operational Research*, 247(3):721–731, 2015.

[48] T. Sato, Y. Takano, R. Miyashiro, and A. Yoshise. Feature subset selection for logistic regression via mixed integer optimization. *Computational Optimization and Applications*, 64(3):865–880, 2016.

[49] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1-3):181–199, 1994.

[50] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[51] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, and T. Koch. ParaSCIP: a parallel extension of SCIP. In *Competence in High Performance Computing 2010*, pages 135–148. Springer, 2011.

[52] D. Stehlé. Floating-point LLL: theoretical and practical aspects. In *The LLL Algorithm*, pages 179–213. Springer, 2009.

[53] R Development Core Team. R: A language and environment for statistical computing. 2013. R home page: `https://www.r-project.org/`

[54] K. Torkkola. Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research*, 3(Mar):1415–1438, 2003.

[55] B. Ustun and C. Rudin. Learning optimized risk scores on large-scale datasets. *arXiv preprint arXiv:1610.00168*, 2016.

[56] S. Vigerske and A. Gleixner. SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. *Optimization Methods and Software*, 33(3):563–593, 2018.