Computers & Chemical Engineering, Volume 32, Issue 10, 17 October 2008, Pages 2306-2319

# Bidirectional Branch and Bound for Controlled Variable Selection Part I: Principles and Minimum Singular Value Criterion

Yi Cao[‡]*and Vinay Kariwala[†]

[‡]School of Engineering, Cranfield University, Cranfield, Bedford MK43 0AL, UK

[†] Division of Chemical & Biomolecular Engineering,

Nanyang Technological University, Singapore 637459

This version: November 12, 2007

**Abstract**

The minimum singular value (MSV) rule is a useful tool for selecting controlled variables (CVs) from the available measurements. However, the application of the MSV rule to large-scale problems is difficult, as all feasible measurement subsets need to be evaluated to find the optimal solution. In this paper, a new and efficient branch and bound (BAB) method for selection of CVs using the MSV rule is proposed by posing the problem as a subset selection problem. In traditional BAB algorithms for subset selection problems, pruning is performed downwards (gradually decreasing subset size). In this work, the branch pruning is considered in both upward (gradually increasing subset size) and downward directions simultaneously so that the total number of subsets evaluated is reduced dramatically. Furthermore, a novel bidirectional branching strategy to dynamically branch solution trees for subset selection problems is also proposed, which maximizes the number of nodes associated with the branches to be pruned. Finally, by replacing time-consuming MSV calculations with novel determinant based conditions, the efficiency of the bidirectional BAB algorithm is increased further. Numerical examples show that with these new approaches, the CV selection problem can be solved incredibly fast.

*Keywords*: Branch and bound, Control structure design, Controlled variables, Combinatorial optimization, Minimum singular value, Self-optimizing control.

*Corresponding Author: Tel: +44-1234-750111; Fax: +44-1234-754685; E-mail:y.cao@cranfield.ac.uk

# Nomenclature

| | |
|---|---|
| $\mathbf{a}$ | column vector (lower case bold face letter) |
| $\mathbf{A}$ | matrix (upper case bold face letter) |
| $B$ | best available lower bound on $J$ |
| $C$ | candidate set of a node |
| $\mathcal{C}_m^n$ | binomial coefficient of $m$ choose $n$ |
| $F$ | fixed set of a node |
| $\mathbf{G}$ | full $m \times n$ matrix for row selection |
| $\mathbf{G}_X$ | sub-matrix of $\mathbf{G}$ consisting of rows with indices in set $X$ |
| $J$ | selection criterion (to be maximized) |
| $\overline{J}_n(X)$ | upper bound on $J$ for all $n$-element subsets of $X$ |
| $m$ | total number of candidate variables |
| $n$ | target size of the subset to be selected |
| $\mathbf{Q}$ | defined as $\mathbf{Q} = \mathbf{G}\mathbf{G}^T$ |
| $\mathbf{Q}_{X,Y}$ | sub-matrix of $\mathbf{Q}$ consisting of rows and columns with indices in sets $X$ and $Y$, respectively |
| $S$ | union of the sets $F$ and $C$, i.e. $S = F \cup C$ |
| $\mathcal{S}$ | a two-tuple, $\mathcal{S} = (F, C)$ represents a node in the search tree |
| $X_t$ | subscript $t$ represents the size of the set $X$ |
| $X^i$ | superscript $i$ represents the index of the sub or super set obtained from $X$ |
| $\alpha_i^S$ | $i^{th}$ downwards pruning index based on set $S$ |
| $\beta_i^F$ | $i^{th}$ upwards pruning index based on fixed set $F$ |
| $\lambda_i$ | $i^{th}$ largest eigenvalue of a square matrix |
| $\underline{\lambda}$ | minimum eigenvalue of a square matrix |
| $\sigma_i$ | $i^{th}$ largest singular value of a matrix |
| $\underline{\sigma}$ | minimum singular value of a matrix |

## 1 Introduction

Control structure design (CSD) deals with the selection of controlled, manipulated and measured variables, and the interconnections linking them. During the past few decades, a number of tools have been developed for various problems arising in CSD; see *e.g.* [22]. Though extremely useful, a common feature of most of these tools is that they require evaluation of all the possible alternatives. As the number of alternatives grows rapidly with the problem size, in many practical situations it is impossible to do an exhaustive search to find the optimal control structure. Thus, effective algorithms are required to find the globally optimal solution without enumerating all the possible alternatives.

In this paper, we consider the selection of controlled variables (CVs) from the available set of measurements. For this problem, a survey of most of the available methods is given by Van de Wal and de Jager [24]. Recently, Skogestad [21] introduced the concept of self-optimizing control, which is useful for selecting CVs. The central idea is to select CVs such that in presence of disturbances, the loss incurred in implementing the operational policy by holding these CVs at constant setpoints is minimal, as compared to the use of an online optimizer. The choice of CVs based on the general non-linear formulation of self-optimizing control can be time-consuming. To quickly pre-screen the alternatives, Skogestad and Postlethwaite [22] presented the approximate minimum singular value (MSV) rule, where the CVs are selected such that the MSV of the scaled gain matrix is maximum among different alternatives. The MSV rule has recently been revisited in [11]. Though efficient for evaluation of a single alternative in comparison with the nonlinear formulation, the MSV rule still suffers from computational intractability as the MSV need to be evaluated for every feasible subset of measurements in order to find the optimal solution. For example, the selection of 5 CVs from 50 measurements requires $\mathcal{C}_{50}^{5} \approx 2.2 \times 10^{6}$ MSV evaluations. This drawback of the MSV rule motivates the present work.

Kariwala and Skogestad [14] have demonstrated that the scaling of the gain matrix, required for application of the MSV rule, can be performed prior to CV selection. In this sense, CV selection based on MSV rule can be seen as selection of the rows of a matrix such that the MSV of the resulting square matrix is maximum among all the alternatives. For this problem, an early sub-optimal solution was proposed by Businger and Golub [3] based on QR factorization. Other possibilities for obtaining sub-optimal solutions include the use of non-square relative gain array [5] and a sequential approach, where one row is selected at a time such that the least non-zero singular value is maximum, until the resulting matrix is square [14].

The problem of MSV maximization by row selection can also be seen as a subset selection problem, for which many different algorithms are available in literature [8]. Among these available methods, however, only the exhaustive search and branch and bound (BAB) method guarantee global optimality [8]. In comparison to exhaustive search, a BAB approach gains its efficiency by pruning certain branches without evaluating nodes (subsets) associated with those branches. To solve subset selection problems, a BAB algorithm was first described by Narendra and Fukunaga [18] in the context of pattern recognition. This algorithm has been subsequently improved by several researchers; see *e.g.* [8, 23, 25]. This algorithm is powerful because any subset selection problem with a monotonic relationship between the subset size and selection criterion can be solved using this method.

The usefulness of BAB method for solving mixed integer linear and nonlinear programming (MILP and MINLP) problems is well-known [9, 20]. Luyben and Floudas [16] and, Kookos and Perkins [15] have applied the BAB algorithm to problems arising in CSD using the general MINLP and MILP frameworks, respectively. In these approaches, the lower and upper bounds on the objective functions are computed by relaxing the integer variables as continuous variables. The optimization problems resulting upon integer relaxation can be non-convex and thus global optimality cannot be guaranteed. In comparison, the use of the BAB algorithm of Narendra and Fukunaga [18] for subset selection problems is advantageous, as the need for solving the relaxed optimization problems is avoided. Furthermore, the BAB algorithm can also be easily tailored to suit the specific problems in CSD as demonstrated in [6, 7, 14], where the globally optimal solution is obtained without exhaustive search.

That MSV satisfies monotonicity requirement and renders the application of BAB method has been shown by Cao et al. [7]. Similar to most of the problems dealing with subset selection, Cao et al. [7] use downwards pruning (gradually decreasing subset size). Logically, pruning can also be conducted upwards (gradually increasing subset size). Recently, Kariwala and Skogestad [14] showed that MSV satisfies upward monotonicity and proposed a new BAB method. For maximization of MSV, we introduce the following novel concepts in this paper:

(a) *Bidirectional pruning*: Intuitively, the upwards pruning based BAB method is more efficient than the downwards pruning based BAB method, when only a few variables are to be selected from a large number of measurements and *vice versa*. We show that the solution tree for upward (downward) search contains a hidden downward (upward) structure. This observation enables us to combine the two pruning strategies such that nodes that cannot lead to an optimal solution are discarded more quickly and fewer nodes need to be evaluated.

(b) *Bidirectional branching*: The efficiency of a BAB method also depends strongly on the branching strategy. By analyzing advantages and disadvantages of the upwards and downwards branching principles, a novel bidirectional branching strategy is developed. The proposed branching approach dynamically decides upon the best search direction (upwards or downwards) and ordering of sub- or super-nodes such that the number of nodes associated with a branch to be pruned is maximized.

(c) *Determinant condition*: BAB method spends most of its time in evaluation of non-optimal nodes. The number of floating point operations required for MSV calculation increases polynomially (circa $\mathcal{O}(n^3)$ for $n \times n$ matrix [10]). We present a computationally inexpensive condition based on determinants, which can be used to quickly decide upon whether expansion of a node can lead to the optimal solution.

These three novel features greatly improve the efficiency of the BAB method in finding the optimal set of CVs using the MSV rule. The improvement in computational efficiency is so substantial that for the benchmark Hydrodealkylation of Toluene (HDA) process [2], which requires selection of 8 CVs among 129 measurements, the proposed method requires less than 0.1 seconds on a Windows XP SP2 notebook with an Intel ®Core$^{\text{TM}}$ Duo Processor T2500 (2.0 GHz, 2MB L2 Cache, 667 MHz FSB) using MATLAB® R2006b. In comparison, the available methods [7, 14] require more than an hour for solving the same problem. Hence, the concepts of bidirectional pruning, branching and determinant condition reduce the solution time by 4 orders of magnitude for the HDA process. We also show that the proposed BAB algorithm scales well with problem dimensions using several randomly generated matrices. In this sense, the BAB algorithm proposed in this paper enables the practicing engineer to efficiently select CVs for large-scale processes using the MSV rule.

Though in this paper, the problem of MSV maximization is considered for selection of individual measurements as CVs, this problem has many other engineering applications. For example, in the local self-optimizing control framework, the designer needs to select a specified number of measurements, whose linear combinations found using null space method can be used as CVs. Alstad [1] has shown that these measurements can be reasonably selected by squaring a matrix such that the MSV is maximized. Similarly, based on controllability arguments, MSV maximization through selection of columns of the gain matrix (or rows of its transpose) is useful for identifying appropriate manipulated variables for CSD [22]. Mavroidis et al. [17] have shown that the sensor locations for long reach manipulator systems can be optimally selected by MSV maximization. Hoog and Mattheij [12] discuss the use of MSV maximization for least squares problems for ill-conditioned matrices. In summary, the computationally efficient method for MSV maximization presented in this paper is applicable to a wide variety of engineering problems.

The rest of the paper is organized as follows. The general principles of unidirectional BAB methods for subset selection are illustrated through upward and downward solution trees in Section 2. Then, the novel bidirectional pruning and branching concepts are proposed in Section 3. These concepts are applied to the MSV rule and efficient BAB algorithms are developed in Section 4. Developed algorithms are tested with several numerical examples in Section 5 and the work is concluded in Section 6.

## 2 Unidirectional branch and bound

In this section, we formally define the problem of subset selection and briefly introduce the principles of unidirectional BAB methods for solving the same.

### 2.1 Subset selection

Let $X_m = \{x_i \mid i = 1, 2, \cdots, m\}$ be an $m$-element set and $X_n$ be an $n$-element subset selected from $X_m$. A subset selection problem with criterion $J$ is to find the globally optimal $n$-element subset, $X_n^*$ such that

$$J(X_n^*) = \max_{X_n \subset X_m} J(X_n). \tag{1}$$

Since there are $\mathcal{C}_m^n = \frac{m!}{(m-n)!n!}$ candidate $n$-element subsets available for selection, the total number of candidates grows very rapidly as $m$ and $n$ increase and thus exhaustive search becomes unviable for large $m$ and $n$. BAB is one of the efficient approaches, which are able to find the globally optimal subset without exhaustive search. BAB search can be conducted either downwards or upwards, where the elements are gradually removed from or added to the subsets, respectively, until the desired subset size is reached.

### 2.2 Downward branch and bound

To present the downward BAB approach, let $\overline{J}_n(X_s)$, $s > n$, be a downwards upper bound on $J$ over all $n$-element subsets of $X_s$,

$$\overline{J}_n(X_s) \geq \max_{X_n \subseteq X_s} J(X_n) \tag{2}$$

Further, let $B$ be a lower bound of $J(X_n^*)$,

$$B \leq J(X_n^*) \tag{3}$$

Then, it follows that

$$J(X_n) < J(X_n^*), \quad \forall X_n \subseteq X_s, \quad \text{if } \overline{J}_n(X_s) < B \tag{4}$$

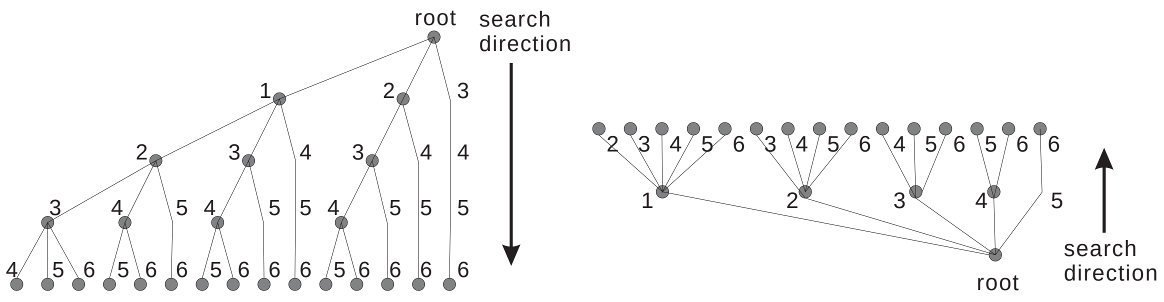
Condition (4) indicates that none of subsets of $X_s$ can be the optimal subset. Therefore, $X_s$ and its subsets can be discarded without further evaluation.

**Remark 1 (Downward monotonicity)** *The above BAB principle requires an estimation of a downwards upper bound on the selection criterion $J$ over all $n$-element subsets of a given set,* i.e. (2). *However, if $J$ satisfies downward monotonicity, i.e.,*

$$J(X_s) \le J(X_t), \quad \textit{if} \quad X_s \subseteq X_t \tag{5}$$

*then a simple downwards upper bound estimation can be adopted as,*

$$\overline{J}_n(X_s) = J(X_s) \tag{6}$$



(a) downward search (b) upward search

Figure 1: Solution trees for $m = 6$ and $n = 2$ selection

The first downward BAB subset selection algorithm based on an asymmetrical solution tree was proposed by Narendra and Fukunaga [18] and further improved by several researchers [8, 23, 25]. To remove subset redundancy, the tree is constructed asymmetrically, *i.e.* nodes at the same level have different number of branches. An example of a downwards solution tree for selecting a 2-element subset from a 6-element set is shown in Figure 1(a). In this tree, the top node is the root of the tree, which represents the set containing all the available elements. From the top to the bottom, at each level, a node represents a subset obtained by eliminating one element from its parent set. Labels at nodes denote the elements discarded there. The bottom nodes of the tree are terminal nodes which represent all possible $n$-element subsets. For selecting an $n$-element subset from an $m$-element set, $(m - n)$ elements are to be discarded. Hence, the downward search tree has $(m - n + 1)$ levels. All nodes at the same level have the same subset size.

For simple representation of the asymmetrical tree and recursive implementation of downwards BAB algorithm, Cao et al. [6, 7] introduced the concepts of fixed and candidate sets of a node. In this paper, these concepts are instrumental in derivation of bidirectional pruning and branching strategies discussed in Section 3.

**Definition 1 (Downwards fixed set $F$)** *The downwards fixed set, $F \subseteq X$ is the largest subset of $X$, whose elements are always included in all the sub-nodes of $X$.*

**Definition 2 (Downwards candidate set $C$)** *The downwards candidate set, $C \subseteq X$ is the set of elements which can be freely removed from $X$, i.e. $C = X \setminus F$, where the symbol $\setminus$ denotes exclude.*

Fixed set and candidate set are complementary, *i.e.* $s = f + c$ if $X_s = F_f \cup C_c$. The downward solution tree is branched according to the following branching rule.

**Definition 3 (Downwards branching rule)** *The node $X_s = F_f \cup C_c$ has $(n - f + 1)$ branches (sub-nodes), which are denoted as $X^i_{s-1} = F^i_{f_i} \cup C^i_{c_i} = X_s \setminus x_i$, $i = 1, 2, \cdots, (n - f + 1)$, arranged from the left to the right. Fixed sets and candidate sets of sub-nodes are defined as:*

$$F^i_{f_i} = F_f \cup \{x_1, \cdots, x_{i-1}\} \tag{7}$$

$$C^i_{c_i} = C_c \setminus \{x_1, \cdots, x_i\} \tag{8}$$

Cao and Saha [6] have shown that the branching rule of Definition 3 is complete without any redundancies, *i.e.* each $n$-element subset belongs to one and only one branch.

**Remark 2 (Hidden upward structure)** *According to (7), for the sub-nodes originating from the same parent node, elements removed from the candidate sets of the left sub-nodes are passed as fixed elements to the right sub-nodes [6, 7]. Therefore, the fixed set of the $i^{th}$ sub-node has $(f + i - 1)$ elements, i.e. $f_i = f + i - 1$ and $F^i_{f_i} = F^i_{f+i-1}$. More importantly, the fixed sets of all the sub-nodes derived from the same parent satisfy the following relationship:*

$$F^1_f \subset F^2_{f+1} \subset \cdots \subset F^{n-f+1}_n \tag{9}$$

*where $F^1_f$ is the fixed set of the leftmost sub-node and $F^{n-f+1}_n$ is the fixed set of the rightmost sub-node. This relationship indicates that there is a hidden upwards (fixed sets expand from left to right) structure in the downwards search tree. This property is exploited to derive a bidirectional pruning strategy in Section 3.1.*

## 2.3 Upward branch and bound

Logically, subset selection can also be performed upwards although this has not been discussed in the literature up to best of our knowledge. An upward search starts from an empty set, gradually expands the superset element-by-element, until it reaches the required subset size.

To present the principle of upward BAB, let $B$ be a lower bound of $J(X_n^*)$ as defined in (3) and $\overline{J}_n(X_s)$, $s < n$, be an upwards upper bound of $J$ over all $n$-element supersets of $X_s$,

$$\overline{J}_n(X_s) \geq \max_{X_n \supseteq X_s} J(X_n) \tag{10}$$

Then,

$$J(X_n) < J(X_n^*), \quad \forall X_n \supseteq X_s, \quad \text{if} \ \ \overline{J}_n(X_s) < B \tag{11}$$

Condition (11) ensures that none of supersets of $X_s$ can be globally optimal. Thus, $X_s$ and its supersets can be pruned without further consideration.

**Remark 3 (Upward monotonicity)** *If the selection criterion satisfies the upward monotonicity*

$$J(X_s) \leq J(X_t), \quad \textit{if} \quad X_s \supseteq X_t \tag{12}$$

*then a simple upwards upper bound estimation can be adopted as*

$$\overline{J}_n(X_s) = J(X_s) \tag{13}$$

The upward BAB method can also be represented using an asymmetric tree. An example of such a tree for selecting 2 elements from a 6-element set is shown in Figure 1(b). In an upward tree, the bottom node is the root of the tree, which represents an empty set. A node represents a superset obtained by adding one element to its parent set. Labels at nodes denote the elements added there. At each level, the sizes of the supersets increase by one until the top level is reached, where the terminal nodes have $n$ elements. Thus, from bottom to top, the tree has $(n + 1)$ levels. The fixed and candidate sets for upward BAB are defined as follows.

**Definition 4 (Upwards fixed set $F$)** *The upwards fixed set, $F$ has the same elements as the node itself,* i.e. $F = X$.

**Definition 5 (Upwards candidate set $C$)** *The upwards candidate set, $C$ is the set of elements which are not in $X$, but can be freely selected to append $X$.*

The upward solution tree is branched according to the following branching rule.

**Definition 6 (Upwards branching rule)** *A node $X_f$ with candidate set $C_c$ has $(f+c-n+1)$ branches (super-nodes), which are denoted as $X^i_{f+1} = X_f \cup x_i$, $i = 1, 2, \cdots, (f+c-n+1)$ and are arranged from the left to the right. Here, $F^i_{f+1} = X^i_{f+1}$ and the candidate set of the $i^{th}$ super-node, $C^i_{c_i}$ is defined by the following branching rule:*

$$C^i_{c_i} = C_c \setminus \{x_1, \cdots, x_i\} \tag{14}$$

*In words, $C^i_{c_i}$ is constructed by eliminating those elements of $C_c$, which have been fixed in its left super-nodes.*

Similar to the downward BAB method, the branching rule for upward BAB method is also complete without redundancies, as shown next.

**Proposition 1 (Upward branch completeness)** *The branching rule in Definition 6 is complete without any redundancies,* i.e. *each $n$-element subset belongs to one and only one branch.*

*Proof*: Let $\mathcal{F}(X_f)$ denote all the supersets generated by the node $X_f$. Consider, $1 \le i < j \le (f+c-n+1)$. Then, according to the branching rule in Definition 6, $x_i \in X_s$, for all $X_s \in \mathcal{F}(X^i_{f+1})$. However, $x_i \notin X_t$, for all $X_t \in \mathcal{F}(X^j_{f+1})$. Thus, $\mathcal{F}(X^i_{f+1}) \cap \mathcal{F}(X^j_{f+1}) = \emptyset$, *i.e.* no superset belongs to more than one branch. This proves that each $n$-element subset belongs to only one branch.

Now, as the solution tree is properly branched so that there is no redundancy in the terminal nodes, it suffices to prove that the numbers of $n$-element subsets that can be reached from any node and its super-nodes are the same. We note that $X_f$ has $\mathcal{C}^{n-f}_c$ valid $n$-element supersets, whilst $\mathcal{C}^{n-f-1}_{c-i}$ valid $n$-element supersets belong to its $i^{th}$ super-node $X^i_{f+1}$. According to the identity $\mathcal{C}^n_m + \mathcal{C}^{n-1}_m = \mathcal{C}^n_{m+1}$, the total number of $n$-element supersets that belong to the two rightmost super-nodes of $X_f$ is

$$\mathcal{C}^{n-f-1}_{n-f-1} + \mathcal{C}^{n-f-1}_{n-f} = \mathcal{C}^{n-f}_{n-f} + \mathcal{C}^{n-f-1}_{n-f} = \mathcal{C}^{n-f}_{n-f+1} \tag{15}$$

Similarly, the the three rightmost super-nodes of $X_f$ contain

$$\mathcal{C}^{n-f}_{n-f+1} + \mathcal{C}^{n-f-1}_{n-f+1} = \mathcal{C}^{n-f}_{n-f+2} \tag{16}$$

$n$-element supersets. By recursively applying these arguments, the number of $n$-element supersets that belong to at least one of super-nodes $X^i_{f+1}$, $i = 1, 2, \cdots, (f+c-n+1)$ is given as

$$\sum_{k=1}^{f+c-n+1} \mathcal{C}^{n-f-1}_{c-k} = \mathcal{C}^{n-f}_c \tag{17}$$

which is same as the number of $n$-element supersets of $X_f$ and thus shows the upward branch completeness.

■

**Remark 4 (Hidden downward structure)** *According to (14), $c_i = c - i$ and for the super-nodes derived from the same parent set, the candidate set of a left super-node is a superset of the candidate sets of all the super-nodes on its right,*

$$C^1_{c-1} \supset C^2_{c-2} \supset \cdots \supset C^{f+c-n+1}_{n-f-1} \tag{18}$$

*Similarly, the unions of the fixed and candidate sets satisfy the following relationship*

$$\left(F^1_{f+1} \cup C^1_{c-1}\right) \supset \left(F^2_{f+1} \cup C^2_{c-2}\right) \supset \cdots \supset \left(F^{c+f-n+1}_{f+1} \cup C^{f+c-n+1}_{n-f-1}\right) \tag{19}$$

*The above relationship shows that there is a hidden downward structure (from left to right) in an upwards search tree, where the candidate sets and, the unions of fixed and candidate sets shrink from left to right. Similar to Remark 2, this property is used to derive a bidirectional pruning strategy in Section 3.1.*

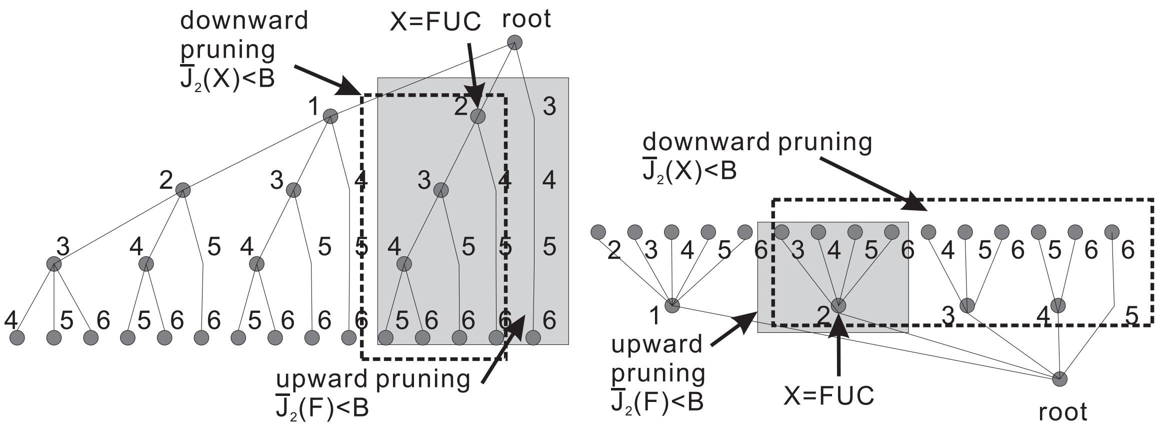
## 3 Bidirectional branch and bound

The this section, the attractive features of the upward and downward approaches are combined to improve the overall efficiency of a BAB method.

### 3.1 Bidirectional pruning

The efficiency of a BAB approach depends on its pruning strategy. Downward and upward search approaches provide two independent pruning strategies, as given in (4) and (11), respectively. According to the Remark 2, there is a hidden upward structure in the downward tree, where the size of the fixed sets of the sub-nodes of the same branch expand upwards from left to right. Similarly, following Remark 4, there is a hidden downward structure in the upward tree, where the size of the union of super-nodes and its candidate sets in the same branch shrink downwards from left to right. Therefore, it is possible to adopt the upwards pruning strategy in a downward solution tree and the downwards pruning strategy in an upward solution tree so that non-optimal nodes can be discarded more quickly and fewer nodes need to be evaluated. In this way, the efficiency of BAB search can be significantly improved.

More specifically, consider a node $X = F \cup C$ in a downward tree. If $\overline{J}_n(X) < B$, then, according to (4) downwards pruning is performed, *i.e.* all the sub-nodes of $X$ are pruned, as shown in the dashed box of Figure 2(a). On the other hand, if $\overline{J}_n(F) < B$, then according to (11) and the hidden upward structure, an upwards pruning can be conducted, *i.e.* the sub-nodes and all the nodes to the right of $X$ are removed, as shown in the gray box of Figure 2(a).



(a) bidirectional pruning in downward tree (b) bidirectional pruning in upward tree

Figure 2: Bidirectional pruning in solution trees

The bidirectional pruning strategy is also applicable to an upward tree. For the node $X = F$ with the candidate set $C$, if $\overline{J}_n(F) < B$, normal upwards pruning is conducted, as shown in the gray box of Figure 2(b). In addition, if the downwards pruning condition $\overline{J}_n(F \cup C) < B$ is satisfied, according to (4) and the hidden downward structure, the super-nodes and all the nodes to the right of $X$ are pruned, as shown in the dashed box of Figure 2(b). The findings of this section are summarized by the following proposition, where a node in a solution tree is represented by a two-tuple, $\mathcal{S} = (F, C)$.

**Proposition 2 (Bidirectional pruning)** *A node $\mathcal{S} = (F, C)$ may contain the globally optimal subset,* i.e. $X_n^* \subset (F \cup C)$ *only if both of the requirements $J_n(F) > B$ and $\overline{J}_n(F \cup C) > B$ are satisfied.*

In comparison to unidirectional BAB methods, Proposition 2 is more efficient for pruning non-optimal nodes at an early stage of the search. The key requirement for using Proposition 2 is the availability of both upwards and downwards upper bounds on $J$. As shown in Section 4.1, these two upper bounds for MSV can be obtained based on monotonicity. Finding the upwards and downwards upper bounds for other commonly used selection criteria is currently under research.

For a node $\mathcal{S} = (F, C)$, downwards pruning is an operation to reduce the size of $C$, whilst upwards pruning is the manipulation to increase the fixed set size by moving an element from $C$ to $F$. The downwards and upwards pruning strategies are uniformly stated in the following lemmas, which are applicable to both the solution trees simultaneously. Instead of pruning a node, as is done traditionally, the proposed results discuss pruning on the sub- and super-nodes. This is not only more efficient in pruning but also facilitates the development of efficient upper bound estimation algorithms as discussed in Section 4.

**Lemma 1 (Downwards subset pruning)** *Let $B$ be the best available lower bound of $J$. Consider a* *node, $\mathcal{S} = (F_f, C_c)$ with $\overline{J}_n(F_f \cup C_c) > B$. For $x_i \in C_c$, if*

$$\overline{J}_n(F_f \cup C_c \setminus x_i) < B \tag{20}$$

*then downwards pruning is performed by replacing $\mathcal{S}$ with $\mathcal{S}^i$ defined as*

$$\mathcal{S}^i = (F_f \cup x_i, C_c \setminus x_i) \tag{21}$$

*Proof*: Based on the branching rule in Definition 3, the terminal nodes of $(F_f, C_c)$ are the same as the union of the terminal nodes of $(F_f, C_c \setminus x_i)$ and $(F_f \cup x_i, C_c \setminus x_i)$. When (20) holds, any terminal node of $(F_f, C_c \setminus x_i)$ cannot be the optimal solution and thus it suffices to evaluate only the node $(F_f \cup x_i, C_c \setminus x_i)$ further. ■

**Lemma 2 (Upwards superset pruning)** *Let $B$ be the best available lower bound of $J$. Consider a node, $\mathcal{S} = (F_f, C_c)$ with $\overline{J}_n(F_f) > B$. For $x_i \in C_c$, if*

$$\overline{J}_n(F_f \cup x_i) < B \tag{22}$$

*then upwards pruning is performed by replacing $\mathcal{S}$ with $\mathcal{S}^i$ defined as*

$$\mathcal{S}^i = (F_f, C_c \setminus x_i) \tag{23}$$

The proof of Lemma 2 is similar to the proof of Lemma 1 and is omitted.

The pruning approaches described in Lemmas 1 and 2 are equivalent to placing the sub- and super-node, respectively, at the leftmost position of the solution tree before pruning it. Therefore, in general, they are more efficient than the traditional strategy of pruning directly from a structured tree. For example, downwards pruning element 2 in the downward tree removes sub-nodes shown in the dashed box of Figure 2(a), which eliminates only 4 terminal nodes. On the other hand, with the approach given in Lemma 1, $\mathcal{C}_6^2 - \mathcal{C}_5^1 = 10$ terminal nodes are removed.

At a specific node, $\mathcal{S} = (F, C)$ with given $B$, the downwards pruning only depends on $F \cup C$. Therefore, all elements in $C$ can be simultaneously checked against (20). Similarly, the upwards pruning only depends on $F$. Hence, the upwards pruning condition (22) can also be simultaneously determined on all elements in $C$. These insights lead to Proposition 3.

**Proposition 3 (Multiple pruning)** *If there are $k$ elements, $x_1, \cdots, x_k$, which satisfy (20), then these $k$ elements can be simultaneously pruned downwards by replacing $\mathcal{S}$ with*

$$\mathcal{S}' = (F_f \cup \{x_1, \cdots, x_k\}, C_c \setminus \{x_1, \cdots, x_k\}) \tag{24}$$

*Similarly, if $k$ elements, $x_1, \cdots, x_k$ satisfy (22), then all elements can be simultaneously pruned upwards by replacing $\mathcal{S}$ with*

$$\mathcal{S}' = (F_f, C_c \setminus \{x_1, \cdots, x_k\}) \tag{25}$$

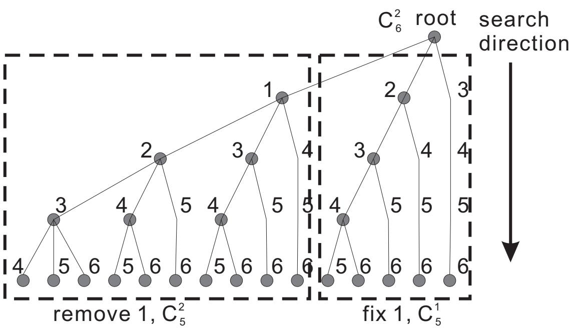
When successful upwards pruning in (25) is conducted, the set $F \cup C$ shrinks. Thus the downwards pruning check can be performed again on the new set $F \cup C$. Similarly, after successful downwards pruning in (24), the set $F$ expands and the upwards pruning check can be conducted again on the new set $F$. By applying these two pruning strategies alternately, the minimal candidate set can be obtained. An algorithm that shows implementation of this idea is given in Section 4.2.

## 3.2 Bidirectional branching

The efficiency of a BAB approach is also strongly related to its branching strategy. This happens as the search tree is asymmetric, *i.e.* nodes on the left of the search tree have more sub-nodes (super-nodes) than those on the right. There is no compulsory rule for a given node $\mathcal{S} = (F, C)$ to be on the left or on the right of a search tree. However, it is desired that a node to be pruned has as many sub-nodes or super-nodes as possible so that the total number of nodes to be evaluated is minimized. In bidirectional BAB, this issue is addressed by dynamically selecting the search direction (upwards or downwards) and using a simplest-first plus best-first search strategy so that the efficiency of bidirectional BAB is maximized.

A node that satisfies the conditions in Proposition 2 needs to be evaluated further either by expanding it in the upward direction or by shrinking it in the downward direction. To decide upon the search direction, we organize the solution tree bidirectionally using the concepts of fixed and candidate sets. As shown in Figure 3(a), subsets of the root node, which has $\mathcal{C}_6^2$ 2-element subsets, can be divided into two groups. The left group with $\mathcal{C}_5^2$ candidates is obtained by removing element 1 from the root, whilst the right group with $\mathcal{C}_5^1$ terminal nodes is derived by fixing element 1. Similarly, in the upward tree, the left group is the one with element 1 fixed whilst the right group is the one with element 1 removed.

In general, the operation of discarding an element from the candidate set or fixing this element by moving this element from the candidate set to the fixed set, represents a binary branching, which can be applied to any node of a solution tree. Such a solution tree for a $\mathcal{C}_6^2$-selection problem is shown in Figure 4. In the binary tree, the decision element for each node is denoted underneath the node. The left sub-node is the one with the decision element removed whilst the right sub-mode is constructed with the decision element fixed. The total number of terminal nodes of a sub-node is marked in the circle. All terminal nodes are


(a) binary branching in downward tree

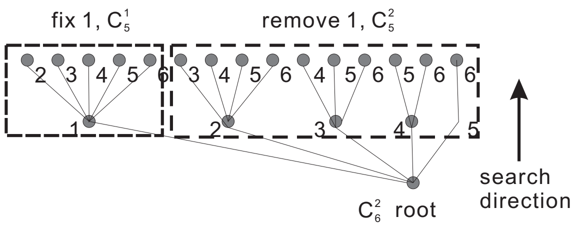

(b) bidirectional branching in upward tree

Figure 3: Bidirectional branching in solution trees

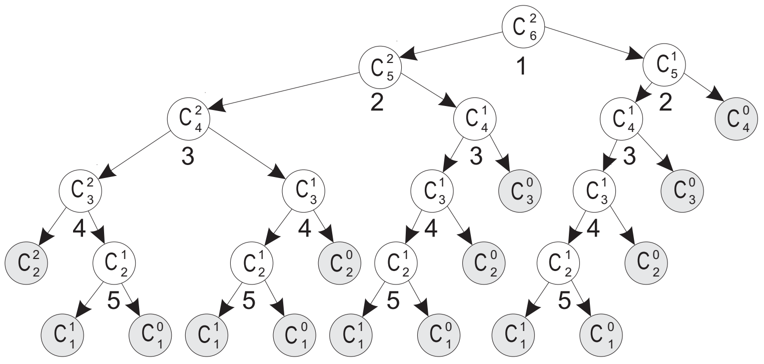


Figure 4: A bidirectional branching example

marked with gray circles. Both upward and downward solution trees can be uniformly represented in the binary tree. Evaluation of the right sub-node increases the fixed set size, hence represents an upward search, whilst evaluation of the left sub-node reduces the candidate set resulting in a downward search. Comparing Figure 4 with Figures 3(a) and 3(b), it can been seen that the right-first search strategy on the downward tree is equivalent to the right-first search strategy on the binary tree. Similarly the right-first search strategy on the upward tree is the same as left-first search strategy on the binary tree. To take advantage of this fact, search on the binary tree can be freely chosen to be either right-first or left-first at each node based on a certain criterion to achieve maximum efficiency.

The maximum efficiency is achieved by pruning as many terminal nodes as possible with as little evaluations as possible. Therefore, an intuitive rule is to perform the simplest-first search. In Figure 4, a node with $\mathcal{C}_m^n$ terminal elements results in two sub-nodes with $\mathcal{C}_{m-1}^n$ (downward) and $\mathcal{C}_{m-1}^{n-1}$ (upward) terminal nodes, respectively. If $\mathcal{C}_{m-1}^n > \mathcal{C}_{m-1}^{n-1}$, then the upward sub-node can be evaluated first and *vice versa*. For example, in the binary tree shown in Figure 4, most of the times search should be performed right-first except at the node with $\mathcal{C}_3^2$ terminal elements, where the left sub-node should be evaluated first. Based on this discussion, the bidirectional search rule is represented as follows.

$$\text{upward search, if } 2(n-f) \leq c \tag{26}$$

$$\text{downward search, if } 2(n-f) > c \tag{27}$$

Efficiency can be improved further by selecting an appropriate element as the decision element at each sub-node. For upward or downward search, the decision element should be selected as the one with largest upwards upper bound given by (10) or the largest downwards upper bound given by (2) (best-first search), respectively. In traditional downwards BAB algorithms, sub-nodes are usually sorted according to their criterion values so that from left to right, criterion values of sub-nodes are in an ascending order. This way, nodes which are more likely to be pruned have more sub-nodes and overall efficiency is improved [19]. However, sorting may incur significant computational overheads especially when the candidate set is relatively large. The decision element selection strategy proposed here requires comparison of only the maximum or minimum values, which is more efficient than sorting.

## 4 Bidirectional controlled variable selection

In this section, the combinatorial problem of selecting CVs using the MSV rule is addressed using the BAB method. Using the MSV rule, selecting CVs is equivalent to selecting $n$ rows of an $m \times n$ matrix $\mathbf{G}$ such that the selected $n \times n$ sub-matrix $\mathbf{G}_{X_n^*}$ has the largest MSV among all possible $n \times n$ sub-matrices, *i.e.*

$$\underline{\sigma}(\mathbf{G}_{X_n^*}) \geq \underline{\sigma}(\mathbf{G}_{X_n}) \qquad \forall X_n \subset \{1, 2, \ldots, m\} \tag{28}$$

In this case, the subset selection problem is equivalent to the selection of $n$ natural numbers from the first $m$ natural numbers, where each selected number represents a row index. Based on this observation, the elements of different sets, *e.g.* $F$ and $C$, represent row indices in the following discussion.

### 4.1 Monotonicity

The BAB method used in this paper requires that the objective function or its upper bound has monotonicity (see Section 2). To show that an upper bound on MSV satisfies the monotonicity requirement, we need the following lemma.

**Lemma 3** Let the matrix $\hat{\mathbf{A}}$ be defined as

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^T & a \end{bmatrix}$$

where $\mathbf{A} \in \mathbb{R}^{p\times p}$ is a Hermitian matrix, $\mathbf{b} \in \mathbb{R}^{p\times 1}$ and $a \in \mathbb{R}$. Let the eigenvalues of $\mathbf{A}$ and $\hat{\mathbf{A}}$ be arranged in descending order. Then [13, Th. 4.3.8]

$$\lambda_{p+1}(\hat{\mathbf{A}}) \leq \lambda_p(\mathbf{A}) \leq \lambda_p(\hat{\mathbf{A}}) \leq \lambda_{p-1}(\mathbf{A}) \leq \cdots \leq \lambda_1(\mathbf{A}) \leq \lambda_1(\hat{\mathbf{A}}) \tag{29}$$

**Proposition 4 (Bidirectional monotonicity of MSV)** *For $\mathbf{A} \in \mathbb{R}^{p\times n}$ and $\mathbf{b} \in \mathbb{R}^{n\times 1}$, let the singular values of $\mathbf{A}$ and $\begin{bmatrix} \mathbf{A}^T & \mathbf{b} \end{bmatrix}^T$ be arranged in descending order. Then*

$$\sigma_{p+1}\left(\begin{bmatrix} \mathbf{A} \\ \mathbf{b}^T \end{bmatrix}\right) \leq \sigma_p(\mathbf{A}) \quad \text{if } p < n \tag{30}$$

$$\sigma_n\left(\begin{bmatrix} \mathbf{A} \\ \mathbf{b}^T \end{bmatrix}\right) \geq \sigma_n(\mathbf{A}) \quad \text{if } p \geq n \tag{31}$$

*Proof*: To prove (30), we note that when $p < n$

$$\sigma_{p+1}\left(\begin{bmatrix} \mathbf{A} \\ \mathbf{b}^T \end{bmatrix}\right) = \lambda_{p+1}^{1/2}\left(\begin{bmatrix} \mathbf{A}\mathbf{A}^T & \mathbf{A}\mathbf{b} \\ \mathbf{b}^T\mathbf{A}^T & \mathbf{b}^T\mathbf{b} \end{bmatrix}\right)$$

Then, using (29), it follows that (30) holds. The inequality in (31) can be proven by noting that when $p \geq n$

$$\sigma_n\left(\begin{bmatrix} \mathbf{A} \\ \mathbf{b}^T \end{bmatrix}\right) = \lambda_n^{1/2}(\mathbf{A}^T\mathbf{A} + \mathbf{b}\mathbf{b}^T) = \lambda_n^{1/2}\left(\begin{bmatrix} \mathbf{A}\mathbf{A}^T & \mathbf{A}\mathbf{b} \\ \mathbf{b}^T\mathbf{A}^T & \mathbf{b}^T\mathbf{b} \end{bmatrix}\right)$$

and applying (29). ■

**Remark 5** *Proposition 4 shows that for a full rank matrix of size $m \times n$ with $m > n$, to select $p$ rows from $m$, if $p \leq n$ the MSV satisfies upward monotonicity, whilst if $p \geq n$, the MSV satisfies downward monotonicity. Therefore, for $p = n$, the MSV satisfies both upward and downward monotonicity and bidirectional BAB can be efficiently conducted.*

To understand the significance of (30) and (31) in the context of MSV maximization problem using BAB method, consider two index sets $I$ and $J$ with $p$ and $q$ elements, respectively, where $p < n < q$. If

$\sigma_p(\mathbf{G}_I) < B$ , (30) implies that all square sub-matrices of $\mathbf{G}$ that can be generated by adding rows to $\mathbf{G}_I$ cannot contain the optimal solution and hence need not be evaluated. Similarly, if $\sigma_n(\mathbf{G}_J) < B$ , (31) implies that all square sub-matrices of $\mathbf{G}$ that can be generated by eliminating rows from $\mathbf{G}_J$ cannot contain the optimal solution and hence need not be evaluated. Thus, using (30) and (31), the optimal solution can be found without complete enumeration.

## 4.2 Fast pruning algorithms

Bidirectional pruning requires the test of two different pruning conditions: $\overline{J}_n(\mathbf{G}_{F\cup C}) < B$ for downwards pruning and $\overline{J}_n(\mathbf{G}_F) < B$ for upwards pruning, where $B$ is the best available lower bound on $J$. The inequalities in (30) and (31) imply monotonicity and hence simplify upper bound calculations. The use of these inequalities, however, involves calculation of singular values, which still requires $\mathcal{O}(n^3)$ floating point operations (flops) for an $n \times n$ matrix [10]. We recognize that the calculation of MSV is only necessary for terminal nodes to update $B$. For non-terminal nodes, it suffices to verify whether they may contain a terminal node that gives a better lower bound than $B$. In this section, we derive determinant based conditions and fast algorithms based on these conditions. For the node under consideration, these algorithms are used to efficiently find the sub-nodes or super-nodes that can be pruned or need to be considered further.

**Proposition 5 (Upwards pruning)** *Let the fat matrix $\mathbf{G}_{F\cup i}$ defined as $\mathbf{G}_{F\cup i} = \begin{bmatrix}\mathbf{G}_F^T & \mathbf{G}_i^T\end{bmatrix}^T$, where $\mathbf{G}_F \in \mathbb{R}^{(p-1)\times n}$ and $\mathbf{G}_i \in \mathbb{R}^{1\times n}$ with $p \le n$. For a positive scalar $B$, if $\sigma_{p-1}(\mathbf{G}_F) = \underline{\sigma}(\mathbf{G}_F) > B$,*

$$\beta_i^F := \mathbf{G}_i\mathbf{G}_i^T - \mathbf{G}_i\mathbf{G}_F^T(\mathbf{G}_F\mathbf{G}_F^T - B^2\mathbf{I}_{p-1})^{-1}\mathbf{G}_F\mathbf{G}_i^T < B^2 \quad \Leftrightarrow \quad \sigma_p(\mathbf{G}_{F\cup i}) = \underline{\sigma}(\mathbf{G}_{F\cup i}) < B \tag{32}$$

*where $\beta_i^F$ denotes the $i^{th}$ upwards pruning index based on the set $F$.*

*Proof*: Using the Schur determinant formula [13]

$$\begin{aligned}\det(\mathbf{G}_{F\cup i}\mathbf{G}_{F\cup i}^T - B^2\mathbf{I}_p) &= \det\begin{bmatrix}\mathbf{G}_F\mathbf{G}_F^T - B^2\mathbf{I}_{p-1} & \mathbf{G}_F\mathbf{G}_i^T \\ \mathbf{G}_i\mathbf{G}_F^T & \mathbf{G}_i\mathbf{G}_i^T - B^2\end{bmatrix} &(33)\\ &= \det(\mathbf{G}_F\mathbf{G}_F^T - B^2\mathbf{I}_{p-1})(\beta_i^F - B^2) &(34)\end{aligned}$$

Now, denoting the eigenvalue decomposition of $\mathbf{G}_F\mathbf{G}_F^T$ as $\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$, we have

$$\det(\mathbf{G}_F\mathbf{G}_F^T - B^2\mathbf{I}_{p-1}) = \det(\mathbf{\Lambda} - B^2\mathbf{I}_{p-1}) = \prod_{k=1}^{p-1}(\lambda_k(\mathbf{G}_F\mathbf{G}_F^T) - B^2) = \prod_{k=1}^{p-1}(\sigma_k^2(\mathbf{G}_F) - B^2) \tag{35}$$

If $\underline{\sigma}(\mathbf{G}_F) > B$, $(\sigma_k^2(\mathbf{G}_F) - B^2) > 0$ for all $k = 1, 2, \cdots, (p-1)$ and thus $\det(\mathbf{G}_F\mathbf{G}_F^T - B^2\mathbf{I}_{p-1}) > 0$. Similar to (35), $\det(\mathbf{G}_{F\cup i}\mathbf{G}_{F\cup i}^T - B^2\mathbf{I}_p) = \prod_{k=1}^{p}(\sigma_k^2(\mathbf{G}_{F\cup i}) - B^2) = (\underline{\sigma}^2(\mathbf{G}_{F\cup i}) - B^2)\prod_{k=1}^{p-1}(\sigma_k^2(\mathbf{G}_{F\cup i}) - B^2)$. Since $\underline{\sigma}(\mathbf{G}_F) > B$, due to the interlacing property in (29), $\sigma_k(\mathbf{G}_{F\cup i}) > B$ for $k = 1, 2, \cdots, (p-1)$ and thus $\prod_{k=1}^{p-1}(\sigma_k^2(\mathbf{G}_{F\cup i}) - B^2) > 0$. Based on (34), the signs of $(\underline{\sigma}^2(\mathbf{G}_{F\cup i}) - B^2)$ and $(\beta_i^F - B^2)$ are the same and (32) follows. ■

In condition (32), the matrix inverse needs to be calculated only once to evaluate all the candidates generated by appending a row to $\mathbf{G}$. Therefore, it is more efficient than direct calculation of MSV. An implementation of the upwards pruning algorithm using the Cholesky factorization is described as follows.

**Algorithm 1 (Upwards pruning)** *Pre-calculate and store $\mathbf{Q} = \mathbf{G}\mathbf{G}^T$ before search, where $\mathbf{G}$ is the original matrix for row selection.*

1. *At a node $\mathcal{S} = (F, C)$, perform the Cholesky factorization, $\mathbf{R}^T\mathbf{R} = \mathbf{Q}_{F,F} - B^2\mathbf{I}_f$, where $\mathbf{Q}_{F,F}$ represents the sub-matrix of $\mathbf{Q}$ constructed by selecting rows and columns with indices in $F$;*
2. *If the factorization fails, $(\mathbf{Q}_{F,F} - B^2\mathbf{I}_f)$ is not positive definite or $\underline{\sigma}(\mathbf{G}_F) < B$. Therefore, the whole node should be pruned, i.e. return without any further calculation;*
3. *For $i \in C$, calculate $\mathbf{x} = \mathbf{R}^{-T}\mathbf{Q}_{F,i}$ using forward substitution through the lower-triangular matrix $\mathbf{R}^T$;*
4. *Calculate $\beta_i^F = \mathbf{Q}_{i,i} - \mathbf{x}^T\mathbf{x}$;*
5. *If $\beta_i^F \le B^2$, perform pruning,* i.e. *remove index $i$ from candidate set $C$;*
6. *Go to step 3 through all $i \in C$.*

For a node with $f$ fixed elements and $c$ candidate elements, the above algorithm requires about $(f + f^3/3 + cf^2 + 2cf + c) = f^3/3 + c(f+1)^2 + f$ flops, where $f$ and $f^3/3$ are related to the computation of $\mathbf{Q}_{F,F} - B^2\mathbf{I}_f$ and Cholesky factorization [10, p.145], respectively, $cf^2$ is associated with the forward substitution (Step 3) and $c(2f+1)$ for Step 4. A direct MSV approach require evaluation of the eigenvalues of $c$ square matrices of size $(f+1) \times (f+1)$, hence requires about $4c(f+1)^3/3$ flops [10, Ch. 8] and hence is very inefficient in comparison with Algorithm 1.

**Proposition 6 (Downwards pruning)** *Let the tall matrix $\mathbf{G}_S$ be defined as $\mathbf{G}_S = \begin{bmatrix}\mathbf{G}_{S\backslash i}^T & \mathbf{G}_i^T\end{bmatrix}^T$, where $\mathbf{G}_{S\backslash i} \in \mathbb{R}^{p\times n}$ and $\mathbf{G}_i \in \mathbb{R}^{1\times n}$ with $p \ge n$. For a positive scalar $B$, if $\sigma_n(\mathbf{G}_S) = \underline{\sigma}(\mathbf{G}_S) > B$,*

$$\alpha_i^S := 1 - \mathbf{G}_i(\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n)^{-1}\mathbf{G}_i^T < 0 \quad \Leftrightarrow \quad \sigma_n(\mathbf{G}_{S\backslash i}) = \underline{\sigma}(\mathbf{G}_{S\backslash i}) < B \tag{36}$$

*where $\alpha_i^S$ denotes the $i^{th}$ downwards pruning index based on the set $S$.*

*Proof*: We first note that $\mathbf{G}_S^T\mathbf{G}_S = \mathbf{G}_{S\backslash i}^T\mathbf{G}_{S\backslash i} + \mathbf{G}_i^T\mathbf{G}_i$ and thus

$$\begin{aligned}\det(\mathbf{G}_{S\backslash i}^T\mathbf{G}_{S\backslash i} - B^2\mathbf{I}_n) &= \det(\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n - \mathbf{G}_i^T\mathbf{G}_i)\\ &= \det(\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n)\det(\mathbf{I}_n - (\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n)^{-1}\mathbf{G}_i^T\mathbf{G}_i)\\ &= \det(\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n)(1 - \mathbf{G}_i(\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n)^{-1}\mathbf{G}_i^T)\\ &= \det(\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n)\alpha_i^S \end{aligned} \tag{37}$$

Similar to the proof of Proposition 5, $\underline{\sigma}(\mathbf{G}_S) > B \Rightarrow \det(\mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n) > 0$ and $\det(\mathbf{G}_{S\backslash i}^T\mathbf{G}_{S\backslash i} - B^2\mathbf{I}_n) = (\underline{\sigma}^2(\mathbf{G}_{S\backslash i}) - B^2)\prod_{k=1}^{n-1}(\sigma_k^2(\mathbf{G}_{S\backslash i}) - B^2)$. We note that

$$\sigma_i(\mathbf{G}_S) = \lambda_i^{1/2}\left(\begin{bmatrix}\mathbf{G}_{S\backslash i}\mathbf{G}_{S\backslash i}^T & \mathbf{G}_{S\backslash i}\mathbf{G}_i^T\\ \mathbf{G}_i\mathbf{G}_{S\backslash i}^T & \mathbf{G}_i\mathbf{G}_i^T\end{bmatrix}\right); \quad i = 1, 2, \cdots, n$$

Since $\underline{\sigma}(\mathbf{G}) = \sigma_n(\mathbf{G}) > B$, due to the interlacing property in (29), $\sigma_k(\mathbf{G}_{S\backslash i}) > B$ for $k = 1, 2, \cdots, (n-1)$ and thus $\prod_{k=1}^{n-1}(\sigma_k^2(\mathbf{G}_{S\backslash i}) - B^2) > 0$. Based on (37), the signs of $(\underline{\sigma}^2(\mathbf{G}_{Si}) - B^2)$ and $\alpha_i^S$ are the same and (36) follows. ∎

Similar to upwards pruning, the matrix inverse needs to be calculated only once to evaluate all the candidates generated by removing a row from $\mathbf{G}_S$ in condition (36). Hence, it is also a very efficient pruning test. An implementation of the downwards pruning algorithm using the Cholesky factorization is described as follows.

**Algorithm 2 (Downwards pruning)** *At a node $\mathcal{S} = (F, C)$, let $S = F \cup C$, $\mathbf{G}_S$ be the corresponding matrix with all $s$ rows indexed by $S$ and $n$ be the number of variables to be selected (the number of rows of $\mathbf{G}_S$).*

1. *Calculate the Cholesky factorization, $\mathbf{R}^T\mathbf{R} = \mathbf{G}_S^T\mathbf{G}_S - B^2\mathbf{I}_n$;*
2. *If the factorization fails, $\underline{\sigma}(\mathbf{G}_S) < B$; Therefore, the whole node should be pruned, i.e. return without any further calculation;*
3. *For $i \in C$, calculate $\mathbf{x} = \mathbf{R}^{-T}\mathbf{G}_i^T$ using forward substitution through the lower-triangular matrix $\mathbf{R}^T$;*
4. *Calculate $\alpha_i^S = 1 - \mathbf{x}^T\mathbf{x}$;*

5. *If $\alpha_i^S \le 0$, perform pruning by fixing $i$,* i.e. *move index $i$ from $C$ to $F$;*

6. *Go to step 3 through all $i \in C$.*

For a node with $f$ fixed elements and $c$ candidate elements, $s = f + c$, and the above algorithm requires about $(n^3/3 + 2sn^2 + n + cn^2 + 2cn + c) = n^3/3 + 2sn^2 + c(n+1)^2 + n$ flops. Here, $(n^3/3 + 2sn^2 + n)$, $cn^2$ and $(2cn + c)$ flops are related to Steps 1, 3 and 4, respectively. Since $\mathbf{G}_S$ is a $s \times n$ matrix, it results in $c$ sub-matrices of size $(s-1) \times n$ upon elimination of one row from $C$. Hence, directly calculating MSV requires $2cn^2(s - 1 + n)$ flops [10, p. 254], which is much more expensive than Algorithm 2.

It is worth noting that both the conditions (32) and (36) require that the MSV of the current node is larger than the best available bound $B$. In both of Algorithms 1 and 2, such a requirement is guaranteed to be satisfied if the Cholesky factorization exists. Therefore, any evaluation of MSV on the non-terminal nodes becomes unnecessary.

**Remark 6 (Multiple pruning)** *The value of $\beta_i^F$ depends on both $F$ and $B$ whilst that of $\alpha_i^S$ depends on $S = F \cup C$ and $B$. For the same $B$, multiple upwards pruning can be conducted simultaneously because $F$ is unchanged in any upwards pruning. Similarly, for a given $B$, multiple downwards pruning is carried out independently since $S$ remains the same during any downwards pruning. These features are useful in parallel computing.*

**Remark 7 (Updating pruning indices)** *If either $F$ is expanded (due to downwards pruning or upwards branching) or $B$ is updated after an upwards pruning test has been conducted, then a new upwards pruning test has to be performed to update $\alpha_i^S$. Similarly, if either $S$ shrinks (due to upwards pruning or downwards branching) or $B$ is updated after a downwards pruning test has been performed, then a new downwards pruning check should be carried out to update $\beta_i^F$.*

The upwards and downwards pruning algorithms can be combined to form a bidirectional pruning algorithm, where the three flags are introduced to check whether a pruning test is necessary; see Remark 7.

**Algorithm 3 (Bidirectional pruning)** *Initially set down-flag, up-flag and bound-flag to true. Assume the target is to select $n$ elements. At a node $\mathcal{S} = (F, C)$ with $f$ fixed indices and $c$ candidate indices, let $r = n - f$.*

1. *while (down-flag or up-flag or bound-flag are true) and $r > 0$ and $c - r > 0$, do*

```
2. if down-flag or bound-flag are true, execute Algorithm 2;
3. if Algorithm 2 fails, break the loop else set down-flag to false;
4. if any element has been moved from $C$ to $F$, set up-flag to true;
5. if (up-flag or bound-flag are true) and $r > 0$, execute Algorithm 1;
6. if Algorithm 1 fails, break the loop else set up-flag to false;
7. if any element has been removed from $C$, set down-flag to true;
8. set bound-flag to false;
9. end of while loop
```

Algorithm 3 results in the largest candidate set $C$ whose elements may be removed or fixed to produce sub-nodes or super-nodes with upper bounds larger than the best available bound $B$. Therefore, the bidirectional algorithm ensures that only the necessary nodes are expanded. This feature is particularly important for the cases where $r = 1$ or $c - r = 1$, *i.e.* where only one element needs to be fixed or removed. In such cases, selecting any element from $C$ produced by Algorithm 3 always leads to an improved bound. Particularly, selecting the element corresponding to the largest $\alpha_i^S$ for $c - r = 1$ cases and fixing the element corresponding to the largest $\beta_i^F$ for $r = 1$ cases often leads to the best subset due to Corollary 1 discussed in Section 4.3.

## 4.3 Fast branching algorithms

For a given node $S = (F, C)$ and lower bound $B$, Algorithms 1 and 2 can be used to efficiently find the subsets and supersets, which can be discarded without computation of singular values. To branch efficiently, however, it is still required that the MSV values for these subsets and supersets be computed and compared; see Section 3.2. The selection of the decision element only requires the relative rankings of the sub- and super-nodes. It is possible to obtain approximate rankings using bounds on the MSV values of the sub- and super-nodes. Note that the use of approximate rankings reduces the computational load significantly, and only affects efficiency but not optimality. With this motivation, we next establish relationships between MSV of supersets and subsets of a node and the upward and downwards pruning indices obtained from Algorithms 1 and 2, respectively.

**Corollary 1 (Upwards upper bound and downwards lower bound)** *For $\beta_i^F$ and $\alpha_i^S$ defined in* (32) *and* (36), *respectively,*

$$\beta_i^F \geq \underline{\sigma}^2(\mathbf{G}_{F\cup i}) = \sigma_{f+1}^2(\mathbf{G}_{F\cup i}) \tag{38}$$

$$\alpha_i^S \leq \frac{\underline{\sigma}^2(\mathbf{G}_{S\backslash i}) - B^2}{\underline{\sigma}^2(\mathbf{G}_S) - B^2} = \frac{\sigma_n^2(\mathbf{G}_{S\backslash i}) - B^2}{\sigma_n^2(\mathbf{G}_S) - B^2} \tag{39}$$

*Proof*: Based on the proof of Proposition 5, we note that

$$\prod_{k=1}^{n}(\lambda_k(\mathbf{G}_{F\cup i}\mathbf{G}_{F\cup i}^T) - B^2) = (\beta_i^F - B^2)\prod_{k=1}^{n-1}(\lambda_k(\mathbf{G}_F\mathbf{G}_F^T) - B^2)$$

According to the interlacing property of eigenvalues in (29),

$$\begin{aligned}
&\beta_i^F - B^2 = (\underline{\lambda}(\mathbf{G}_{F\cup i}\mathbf{G}_{F\cup i}^T) - B^2)\prod_{k=1}^{n-1}\frac{\lambda_k(\mathbf{G}_{F\cup i}\mathbf{G}_{F\cup i}^T) - B^2}{\lambda_k(\mathbf{G}_F\mathbf{G}_F^T) - B^2} \geq \underline{\lambda}(\mathbf{G}_{F\cup i}\mathbf{G}_{F\cup i}^T) - B^2 \\
\Rightarrow &\beta_i^F \geq \underline{\lambda}(\mathbf{G}_{F\cup i}\mathbf{G}_{F\cup i}^T) = \underline{\sigma}^2(\mathbf{G}_{F\cup i})
\end{aligned}$$

Similarly, based on the proof of Proposition 6 and the interlacing property of eigenvalues in (29)

$$\begin{aligned}
&\prod_{k=1}^{n}(\lambda_k(\mathbf{G}_{S\backslash i}^T\mathbf{G}_{S\backslash i}) - B^2) = \alpha_i^S\prod_{k=1}^{n}(\lambda_k(\mathbf{G}_S^T\mathbf{G}_S) - B^2\mathbf{I}_n) \\
\Rightarrow &\alpha_i^S = \frac{\underline{\lambda}(\mathbf{G}_{S\backslash i}^T\mathbf{G}_{S\backslash i}) - B^2}{\underline{\lambda}(\mathbf{G}_S^T\mathbf{G}_S) - B^2}\prod_{k=1}^{n-1}\frac{\lambda_k(\mathbf{G}_{S\backslash i}^T\mathbf{G}_{S\backslash i}) - B^2}{\lambda_k(\mathbf{G}_S^T\mathbf{G}_S) - B^2} \\
\Rightarrow &\alpha_i^S \leq \frac{\underline{\lambda}(\mathbf{G}_{S\backslash i}^T\mathbf{G}_{S\backslash i}) - B^2}{\underline{\lambda}(\mathbf{G}_S^T\mathbf{G}_S) - B^2} \\
&\alpha_i^S \leq \frac{\underline{\sigma}^2(\mathbf{G}_{S\backslash i}) - B^2}{\underline{\sigma}^2(\mathbf{G}_S) - B^2}
\end{aligned}$$

■

According to Corollary 1, $\alpha_i^S$ is related to the lower bound on the MSV values of the sub-nodes and the decision element can be selected by finding the sub-node with the largest $\alpha_i^S$ (best to fix) for upward-first branching, or the smallest $\alpha_i^S$ (best to discard) for downward-first branching based on the best-first rule. Similarly, $\beta_i^F$ provides an upper bound on the MSV values of the super-nodes and thus the decision element can be selected by finding the super-node with the smallest $\beta_i^F$ (best to discard) for downward-first branching, or the largest $\beta_i^F$ (best to fix) for upward-first branching. Hence, the evaluation of the MSV can be completely eliminated for non-terminal nodes. Moreover, according to Remark 7 these indices need to be updated (recalculated) only when either the lower bound $B$ is updated or for $\beta_i^F, i \in C$ when some rows are moved from $C$ to $F$, or for $\alpha_i^S, i \in C$ when some elements are eliminated from $C$.

Combining all the aforesaid techniques, we next present a recursive implementation of bidirectional BAB algorithm, B$^3$. Let $\mathbf{G}$ be a $m \times n$ matrix with $m > n$. The problem is to select $n$ rows such that the MSV of the resulting square matrix is largest among all the square matrices that can be obtained from $\mathbf{G}$. In preparation stage, one needs to calculate $\mathbf{Q} = \mathbf{G}\mathbf{G}^T$, initialize $F = \emptyset$, $C = \{1, \cdots, m\}$, $B^2 = 0$, and set down-flag, up-flag and bound-flag to true.

**Algorithm 4 (Bidirectional branch and bound (B$^3$)[4])** *Assume the given node $\mathcal{S} = (F, C)$.*

```
1. evaluate $f$ and $c$ as the number of elements of $F$ and $C$, respectively. Set $r = n - f$ and $S = F \cup C$.
2. while $r > 0$ and $c > r$ do
3. Algorithm 3;
4. if fails or $r < 0$ or $c < r$ return;
5. if $r = 0$ or $c = r$ break the while loop;
6. if $r > 1$ goto step 9, else select $i = \arg\max \beta_i^F$, $i \in C$;
7. update bound $B^2 = \max(B^2, \underline{\lambda}(\mathbf{Q}_{F \cup i, F \cup i}))$ and set bound-flag to true if the bound is updated;
8. remove $i$ from $C$, set down-flag to true, continue the while loop;
9. if $r - c > 1$ goto step 12, else select $i = \arg\max \alpha_i^S$, $i \in C$;
10. update bound $B^2 = \max(B^2, \underline{\lambda}(\mathbf{Q}_{S \setminus i, S \setminus i}))$ and set bound-flag to true if the bound is updated;
11. move $i$ from $C$ to $F$, set up-flag to true, continue the while loop;
12. if $2r > c$ goto step 15, else select $i = \arg\min \alpha_i^S, i \in C$;
13. call Algorithm 4 with the fixed set $F \cup i$, the candidate set $C \setminus i$ and up-flag being true;
14. remove $i$ from $C$, set down-flag to true, continue the while loop;
15. select $i = \arg\min \beta_i^F$, $i \in C$;
16. call Algorithm 4 with the fixed set $F$, the candidate set $C \setminus i$ and down-flag being true.
17. move $i$ from $C$ to $F$, set up-flag to true;
18. end of the while loop;
```

```
19. if $r = c$ update bound $B^2 = \max(B^2, \underline{\lambda}(\mathbf{Q}_{F\cup C,F\cup C}))$ and set bound-flag to true if the bound is updated;
20. if $r = 0$ update bound $B^2 = \max(B^2, \underline{\lambda}(\mathbf{Q}_{F,F}))$ and set bound-flag to true if the bound is updated;
21. return
```

**Remark 8** *As indicated in [7], by changing $B$ to a vector with $p$ criterion values arranged in ascending order and using the first element as the lower bound for comparison, $B^3$ algorithm can be used to find the best $p$ subsets. This extension is particularly useful for CV selection, where local analysis is used to identify promising candidates and the final set of CVs is selected by evaluating the loss for these candidates based on the non-linear model.*

## 5 Numerical examples

To examine the efficiency of the bidirectional BAB algorithm, numerical tests are conducted using random data and a real self-optimizing control case study. Programs used for loss minimization or MSV maximization are listed in Table 2. As all programs provide the same optimal solution, we only compare the computational efficiency of different algorithms in the following discussion. All tests were conducted on a Windows XP SP2 notebook with an Intel®Core$^{\text{TM}}$ Duo Processor T2500 (2.0 GHz, 2MB L2 Cache, 667 MHz FSB) using MATLAB® R2006b.

Table 2: Branch and bound programs for MSV maximization

| Algorithm | Description |
|---:|---|
| BB1 | original program developed in [7] |
| BB2 | original program developed in [14] |
| UP | upwards pruning using determinant condition (32) |
| DOWN | downwards pruning using determinant condition (36) |
| B$^3$ | the bidirectional branch and bound algorithm (Algorithm 4) |

### 5.1 Random tests

Four sets of random tests are conducted to evaluate the efficiency of the proposed B$^3$ algorithm. The first test consists of selecting $n$ out of $2n$ variables, where $n$ varies from 5 to 20. For each $n$, 100 $2n \times n$ random matrices are generated. For this test, the average computation time and the average number of nodes evaluated are plotted against $n$ in Figures 5 (a) and (b), respectively. The second test is to select $n$ out of 40 variables with $n$ ranging from 1 to 39. The average computation time and average number of node evaluations calculated over 100 random cases are plotted against $n$ in Figures 5 (c) and (d), respectively. In order to compare the efficiency of proposed BAB algorithms, the number of node evaluations required by the brute force search (marked as BRUTE) is also shown in Figures 5 (b) and (d). For clarity, the average computation time required by brute force search is not shown in Figures 5 (c) and (d), but note that the computation time of brute force search is (approximately) directly proportional to the number of node evaluations, *i.e.* number of node evaluations multiplied by the computation time required for a single MSV evaluation.

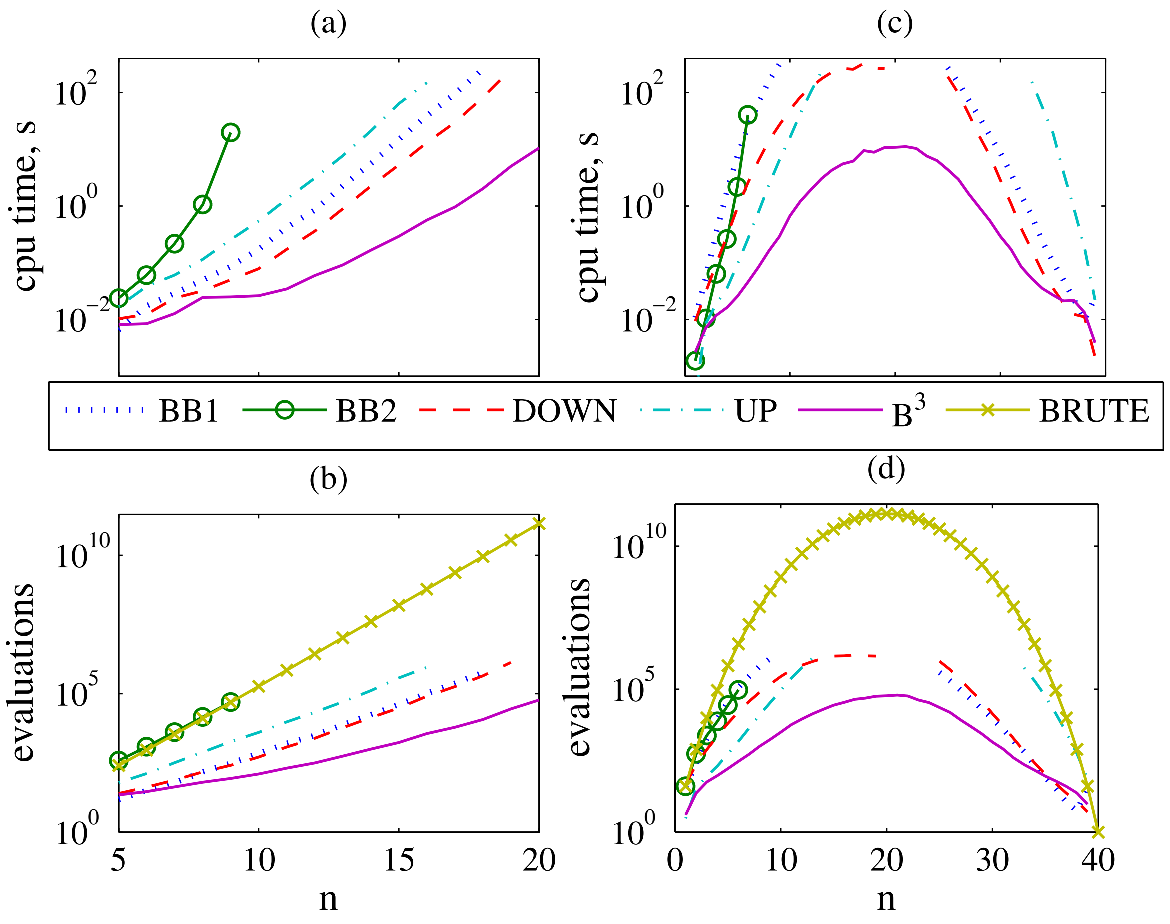


Figure 5: Random test 1: selection of $n$ out of $2n$ variables, (a) computation time against $n$ and (b) number of nodes evaluated against $n$; Random test 2: selection of $n$ out of 40 variables, (c) computation time against $n$ and (b) number of nodes evaluated against $n$.

From Figure 5, it can been seen that the evaluation count of DOWN algorithm is similar to BB1 algorithm [7], another downward algorithm, where power iteration method [10] is used to avoid direct

MSV calculations. However, the determinant based UP algorithm is much more efficient than BB2 algorithm [14], another upward algorithm based on direct MSV calculations. In terms of computation time, both DOWN and UP algorithms demonstrate the efficiency of the determinant algorithms. Furthermore, by combining upward and downwards pruning, the bidirectional $B^3$ algorithm exhibits superior efficiency, extending the manageable selection size beyond $C_{40}^{20}$.

The third and fourth random tests are designed to select 5 out of $n$ and $(n-5)$ out of $n$ variables, respectively. Each selection problem is tested for 100 randomly generated matrices. The average computation time and average evaluation count are summarized in Figure 6 (a)-(d), respectively. All figures clearly show that upward and downward pruning based algorithms are more suitable to select a few variables and discard a few variables from a large candidate set, respectively. However, for both kinds of selection problems, the bidirectional algorithm improves performance dramatically and its efficiency is relatively insensitive to the nature of the selection problem.

## 5.2 HDA case study

The developed algorithms are also applied to select CVs for the HDA process, which was used as a case study in [14]. Details of self-optimizing control of this process can be found in [2]. The process has 8 degrees of freedom, *i.e.* 8 CVs are to be selected from 129 available measurements. It is interesting to note that the problem has $\mathcal{C}_{129}^{8} \approx 1.52 \times 10^{12}$ alternatives, whereas MATLAB® requires about 0.05 ms to calculate the MSV of an $8 \times 8$ matrix. Therefore, if the problem were to be solved using the brute force approach, it would require more than 2 years to find the globally optimal solution.

The computation times required to solve this selection problem using various BAB algorithms are compared in Table 3. It is clear that for this case study, upward approaches have an advantage over downward ones in terms of the number of nodes evaluated. This is reasonable, as very few variables are to be selected from a large number of alternatives. The novel determinant pruning algorithms also demonstrate their efficiency by reducing evaluation count significantly. The most astonishing improvement of performance is achieved by bidirectional algorithm, $B^3$, which reduces both computation time and evaluation count by 4 orders of magnitude.

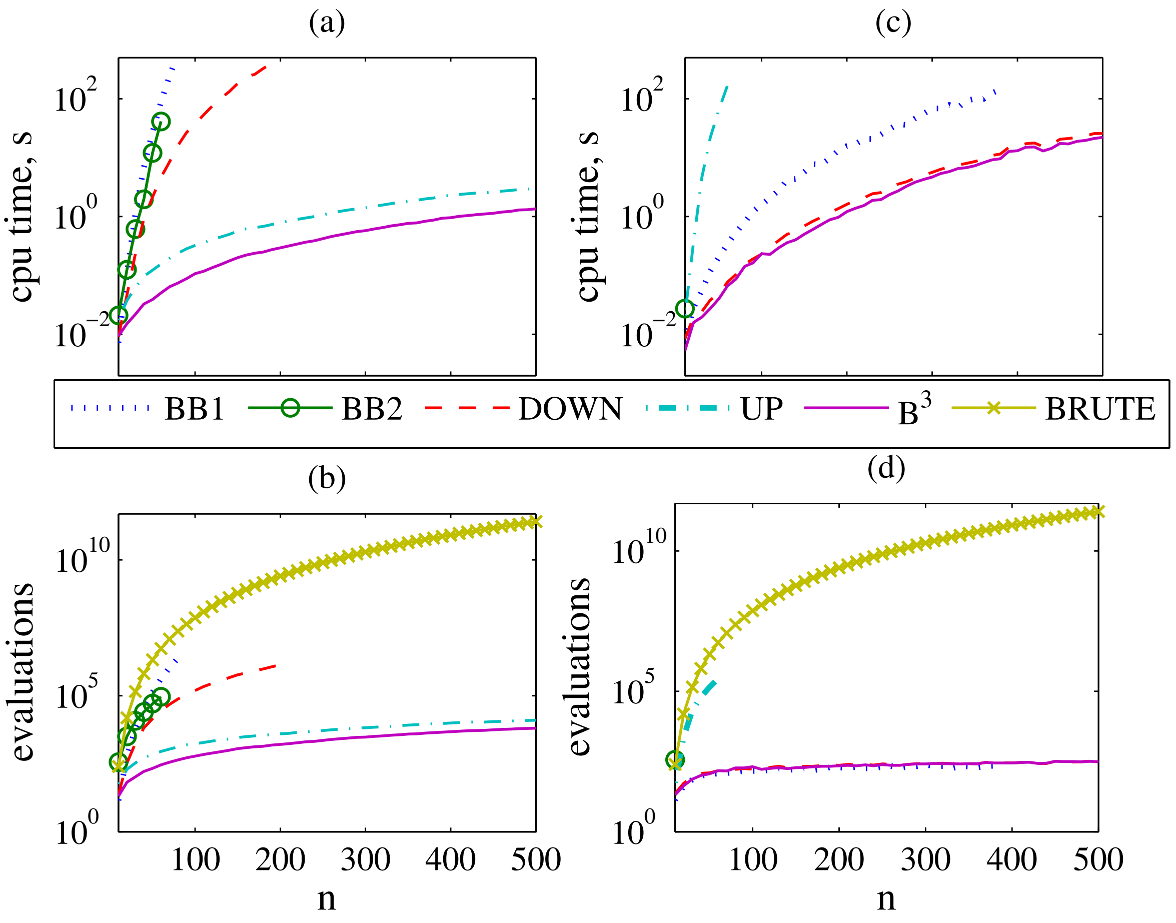


Figure 6: Random test 3: selection of 5 out of $n$ variables, (a) computation time against $n$ and (b) number of nodes evaluated against $n$; Random test 4: selection of $(n-5)$ out of $n$ variables, (c) computation time against $n$ and (d) number of nodes evaluated against $n$.

Table 3: computation time and number of evaluations for the HDA case study

| program | cpu time (s) | evaluation |
|---|---|---|
| BB1 | 2337.6 | 5035735 |
| BB2 | 3809.0 | 809673 |
| UP | 41.98 | 177811 |
| DOWN | 294.51 | 1254891 |
| $B^3$ | 0.046 | 263 |

## 6 Conclusions

In this paper, theoretical frameworks for unidirectional and bidirectional branch and bound (BAB) principles have been developed based on the concepts of fixed and candidate sets. By introducing the upward BAB concept, a hidden upward structure in the traditional downward BAB solution tree and a hidden downward structure in the proposed upward BAB solution tree have been identified. These developments lead to the concepts of bidirectional pruning and branching, and a novel bidirectional BAB scheme. The proposed concepts are independent of the selection criterion, *i.e.* they can be applied to any subset selection problem.

For self-optimizing control problems, the minimum singular value (MSV) criterion has been adopted to select controlled variables (CVs) from the available measurements. The criterion satisfies bidirectional monotonicity and thus upper bound estimation can be simplified. A set of novel determinant based conditions have been developed to accelerate pruning, either in the upwards or downwards directions. By combining these conditions with the bidirectional BAB principles, an efficient bidirectional BAB ($B^3$) algorithm has been developed. Numerical examples clearly show the superior performance of $B^3$ algorithm. Particularly for the HDA case study, $B^3$ algorithm successfully reduces both the computation time and the evaluation count by 4 orders of magnitude, as compared to the previously available methods.

Although MSV maximization is a simple criterion for CV selection, it may fail to identify the CVs providing minimal local loss in some cases. This fact was pointed out by Halvorsen *et al.* [11], who also derived expressions for exact local loss. A bidirectional BAB algorithm based on the exact local loss minimization has been developed, which will be presented in the second part of this work.

## Acknowledgements

The second author gratefully acknowledges the financial support from Office of Finance, Nanyang Technological University, Singapore through grant no. M52120046.

## References

[1] V. Alstad. *Studies on Selection of Controlled Variables.* PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2005. Available at `http://www.nt.ntnu.no/users/skoge/`

`publications/thesis/2005_alstad/`.

[2] A. Araujo and S. Skogestad. Application of plantwide control to the HDA process. I – Steady-state optimization and self-optimizing control. *Control Engineering Practice*, 15(10):1222–1237, 2007.

[3] P. A. Businger and G. H. Golub. Linear least squares solution by householder transformations. *Numerische Mathematik*, 7(3):269–276, 1965.

[4] Y. Cao and V. Kariwala. B3MSV. MATLAB File Exchange, November 2007. Available at `http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=17480&objectType=file`.

[5] Y. Cao and D. Rossiter. An input pre-screening technique for control structure selection. *Computers Chem. Engng.*, 21(6):563–569, 1997.

[6] Y. Cao and P. Saha. Improved branch and bound method for control structure screening. *Chem. Engg. Sci.*, 60(6):1555–1564, 2005.

[7] Y. Cao, D. Rossiter, and D. H. Owens. Globally optimal control structure selection using branch and bound method. In *Proceedings of IFAC-symposium on DYCOPS 5*, pages 183–188, Corfu, Greece, 1998.

[8] X.-W. Chen. An improved branch and bound algorithm for feature selection. *Pattern Recognition Letters*, 24(12):1925–1933, 2003.

[9] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamental and Applications.* Oxford University Press, New York, NY, USA, 1995.

[10] G. H. Golub and C. F. Van Loan. *Matrix computations.* The Johns Hopkins University Press, Baltimore, MD, USA, 3rd edition, 1996.

[11] I. J. Halvorsen, S. Skogestad, J. C. Morud, and V. Alstad. Optimal selection of controlled variables. *Ind. Eng. Chem. Res.*, 42(14):3273–3284, 2003.

[12] F. R. De Hoog and R. M. M. Mattheij. Subset selection for matrices. *Linear Algebra and its Applications*, 422(2-3):349–359, 2007.

[13] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, Cambridge, UK, 1985.

[14] V. Kariwala and S. Skogestad. Branch and bound methods for control structure design. In *Proc. 16th ESCAPE and 9th International Symposium on PSE*, Garmisch-Partenkirchen, Germany, 2006.

[15] I. K. Kookos and J. D. Perkins. Heuristic-based mathematical programming framework for control structure selection. *Ind. Eng. Chem. Res.*, 40:2079–2088, 2001.

[16] M. L. Luyben and C. A. Floudas. Analyzing the interaction of design and control - 1. A multiobjective framework and application to binary distillation column. *Computers Chem. Engng.*, 18(10):933–969, 1994.

[17] C. Mavroidis, S. Dubowsky, and K. Thomas. Optimal sensor location in motion control of flexibly supported long reach manipulators. *J. Dynamic Systems, Measurement and Control*, 119:718, 1997.

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

[19] M. S. Ridout. An improved branch and bound algorithm for feature subset selection. *Applied Statistics*, 37(1):139–147, 1988.

[20] Nikolaos V. Sahinidis. *Branch And Reduce Optimization Navigator (BARON) Users Manual v 4.0.* University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 2000.

[21] S. Skogestad. Plantwide control: The search for the self-optimizing control structure. *J. Proc. Control*, 10(5):487–507, 2000.

[22] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control: Analysis and Design.* John Wiley & Sons, Chichester, UK, 1st edition, 1996.

[23] P. Somol, P. Pudil, and J. Kittler. Fast branch & bound algorithms for optimal feature selection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(7):900–912, 2004.

[24] M. Van de Wal and B. de Jager. A review of methods for input/output selection. *Automatica*, 37(4): 487–510, 2001.

[25] B. Yu and B. Yuan. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition*, 26(6):883–889, 1993.