2009 International Conference on Computer Engineering and Applications
IPCSIT vol.2 (2011) © (2011) IACSIT Press, Singapore

A Review of Suboptimal Branch and Bound Algorithms

Songyot Nakariyakul

Department of Electrical and Computer Engineering, Thammasat University
Khlong Luang, Pathumthani 12120 Thailand

Abstract. The branch and bound algorithm is an optimal feature selection method that is well-known for its computational efficiency. However, when the dimensionality of the original feature space is large, the execution time required by the branch and bound algorithm becomes very excessive. If the optimality of the algorithm is allowed to be compromised, the search time can be greatly reduced by employing the look-ahead search strategy to eliminate many solutions deemed to be suboptimal early in the search. In this paper, a comparative study of the effect of the look-ahead scheme on four major branch and bound algorithms is carried out on two real data sets. We give a guideline for setting the look-ahead parameter to reduce the computational time while obtaining a good solution.

Keywords: branch and bound algorithm, dimension reduction, feature selection, search algorithms, suboptimal solutions.

1. Introduction

Feature selection techniques refer to algorithms that select a subset of d features from an original set of D features. The problem of feature selection is very important in machine learning and pattern recognition applications [1, 2] because use of feature selection not only helps avoid overfitting and improves classification performance but also reduces the cost of collecting redundant features. Many high-dimensional databases (D is large) such as medical images and hyperspectral remote sensing data often include irrelevant features for classification, and use of feature selection is mandatory. Thus, developing effective feature selection algorithms has been the focus of much work in statistical pattern recognition.

There are many feature selection techniques in the literature, but only exhaustive search and the branch and bound (BB) algorithm [3] (and its variants) are known to provide the optimal solutions. An exhaustive search finds the best subset of d features out of D by evaluating a given criterion function J for all possible feature subset combinations and selecting the subset with the best criterion function J value. An exhaustive search is only feasible for low-dimensional problems. For a large-dimensional database, the number of possible feature subsets that need to be exhaustively searched is exponentially large, making the exhaustive search impractical. The BB algorithm explores the search space more efficiently than an exhaustive search by organizing the search to reject many subsets that are guaranteed to be suboptimal without computing their J values. Although many modified versions [4–7] of the BB algorithm have been proposed to improve the search speed of the original BB algorithm, the computational time required by these BB algorithms becomes excessive when the dimensionality D of the original feature space reaches a few dozens or more. However, if the optimality of the algorithm is allowed to be compromised, the search time can be greatly reduced.

In this paper, we investigate the performance of the basic BB algorithm and its improvements when the optimality of the BB algorithm is not retained. The main aim is to explore a trade-off between the optimality of the selected feature subsets and the reduced execution time. Although many comparative studies have been carried out for the BB algorithms [3–8], only [3] studied the suboptimal solutions of the basic BB algorithm, and [8] compared the suboptimal solutions of the basic BB algorithm with those of one of its recent improvements. We provide an overview of four major BB algorithms and their suboptimal solutions

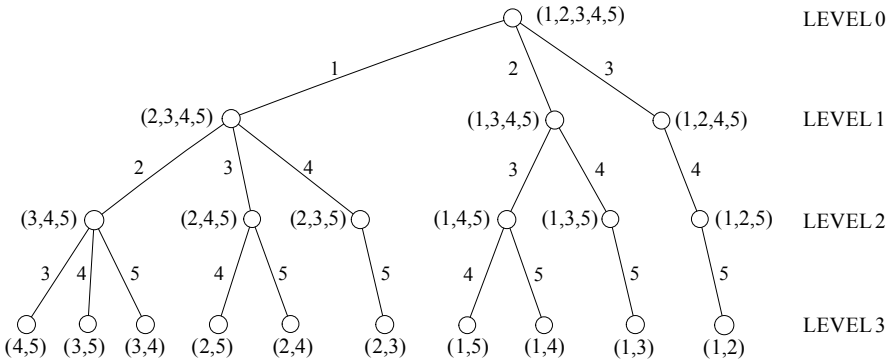


Fig. 1: The solution tree for the basic BB algorithm when $d = 2$ and $D = 5$.

in Sections 2 and 3, respectively. The advantages and disadvantages of each version of the BB algorithm are noted using experimental results for two real data sets in Section 4. We advance conclusions in Section 5.

2. The Branch and Bound Algorithm

In this section, we describe the basic BB algorithm [3] and prior improvements to it [4–7]. We use the notation in [9] that $Y \setminus \{y_1, y_2, ..., y_j\}$ denotes removing the subset of j features $y_1, y_2, ..., y_j$ from the feature set Y. The BB algorithm requires that the criterion function J satisfies monotonicity. That is, assume that $\{y_1, y_2, ..., y_i\}$ is a subset of $\{y_1, y_2, \ldots, y_j\}$ for $i < j$, the monotonicity property of the criterion function requires that the J values of the two sets, $Y \setminus \{y_1, y_2, ..., y_i\}$ and $Y \setminus \{y_1, y_2, ..., y_j\}$, fulfill $J(Y \setminus \{y_1, y_2, ..., y_i\}) \geq J(Y \setminus \{y_1, y_2, ..., y_j\})$. Thus, a subset with a larger J value is better than one with a smaller J value.

The basic BB algorithm selects $D - d$ features to be *discarded*. Fig. 1 is a solution tree corresponding to selecting the best $d = 2$ features out of $D = 5$ total features. It has three levels, with one feature being omitted at each level of the tree. The root of the tree (the top) corresponds to the original set of all five features, and the leaves at the bottom of the tree correspond to all ten possible subsets of two features. The problem is to find the best node (leaf) at the bottom of the tree with the largest J by exploring the tree with the fewest number of J calculations. The BB algorithm starts the search at the top of the tree (level 0), and all nodes at level-1 are analyzed. The successor nodes (all nodes below) of the node with the largest J are further explored, and the search continues to the leaves of the tree (level 3 in our example). The current best subset (leaf) is found with an initial bound B for the criteria function J. The algorithm then *backtracks* to any unexplored nodes at level 2 and then those unexplored nodes at level 1. If $J > B$ for a node (larger J values are better), its successor nodes are explored further (as long as their J values remain larger than B). If $J < B$ for a node, its successor nodes (leaves) at the bottom of the tree are cut off because they cannot be the optimal subset due to the monotonicity property of J. If a new leaf with a $J > B$ is found, the bound B is updated with the new larger J value. The search and backtracking continues until all leaves in the tree are either explored or cut off from the tree; thus, the BB algorithm gives the optimal solution.

Several improvements have been made to the basic BB algorithm. Fukunaga [4] ordered the nodes in the tree when designing the solution tree such that the leftmost successor node of a given node has a smaller J value than the rightmost successor node of that node. The aim is to increase the chance of cutting off the leftmost successor nodes, which have more branches than any other nodes. We refer to this method as the *ordered BB algorithm*. The ordered BB algorithm efficiently searches the solution tree in the process of designing it, but it requires more J calculations than the basic BB algorithm when $d << D$ [6].

Yu and Yuan showed [5] that in searching the rightmost path of the tree in Fig. 1 that has only single-branching intermediate nodes, J needs to be calculated only once at node (1, 2) at the bottom leaf of the tree, but not at intermediate nodes (1, 2, 4, 5) and (1, 2, 5). By removing these single-branching intermediate nodes in the tree, one obtains the *minimum solution tree* [5] and saves many unnecessary J calculations. All BB algorithm experiments in this paper implement the *minimum solution tree*.

The fast branch and bound (FBB) algorithm [6] proposed by Somol *et al.* uses a statistical prediction mechanism to estimate $\hat{J}$ rather than calculating the true J for each node, since calculation of the true J represents the major computation needed in the search. Calculation of the estimated $\hat{J}$ is much faster than

calculation of the true J criterion function. When the estimated $\hat{J}$ for a node is less than the current bound B, the true criterion function J for that node is computed, and whether that node is cut-off is based on the true J criterion function value and the present B value. Hence, the FBB algorithm selects the optimal solution, since nodes are never cut off based on the $\hat{J}$ estimation.

Recently, the adaptive branch and bound (ABB) algorithm [7] improves on prior BB algorithms in four major aspects. First, the ABB algorithm uses the suboptimal sequential floating forward selection (SFFS) technique [9] to select a large initial bound B. A large initial bound allows the search to cut off many branches early at the lower levels near the root of the tree. Next, a simple feature ordering is employed to order nodes in the tree. Third, the node ordering technique and the initial bound B are used to find the starting search level (not one) in the tree. Fourth, ABB models the criterion function J as a deterministic function and uses it along with the J value of the current node and the current bound B to *jump search* to successor nodes at the predicted tree level. The ABB algorithm was shown [7] to perform faster than other BB algorithms when $d << D$.

3. The Look-ahead Search Strategy for Suboptimal Solutions

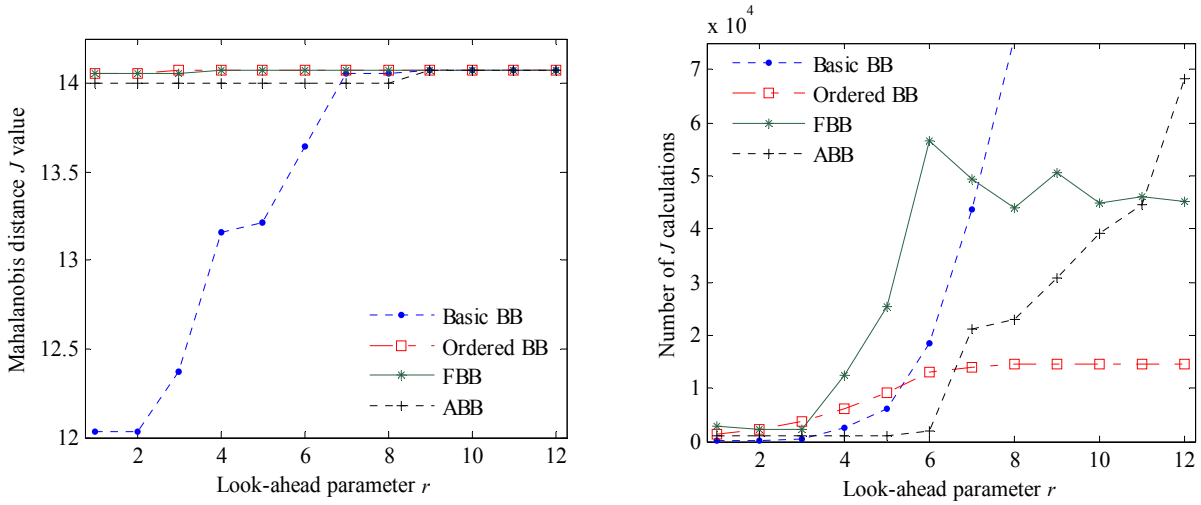
We now discuss the look-ahead search strategy [3] used to increase the search speed of the BB algorithm. To cut off branches of the current node, the BB algorithm checks whether its J value is less than the current best bound B obtained at some leaf node (at level $D - d$). Using the bound B obtained at the leaf node to cut off nodes ensures that the successor nodes of the node with $J < B$ cannot be optimal. If we allow the optimality of the BB algorithm to be compromised, we can eliminate many more successor nodes. This is done by using the current largest bound b_{k+r} found so far at level $k + r$, rather than the bound B at level $D - d$, to cut off nodes, where k is the level of the tree of the node currently being searched and r is a *look-ahead* parameter. That is, the J value of the current node at level k is compared with the best bound b_{k+r} at r levels further down the tree. If $k + r$ is greater than $D - d$ (the leaf level), we set b_{k+r} to B. Since b_{k+r} is always larger than or equal to the bound B, many more branches can be cut off earlier. This look-ahead scheme was incorporated into the basic BB algorithm and shown to reduce many more J calculations [3]. However, there is no guarantee that the obtained solution is optimal because the bound B at the leaf node is no longer used to cut off the tree.

To incorporate the look-ahead scheme into the basic BB and ordered BB algorithms is quite straightforward; we simply record the best bound b_k found so far at each level k when we traverse from the root to the leaves of the tree. In FBB, when the true J values for the bound b_{k+r} at level $k + r$ are not available (i.e., only predicted J values are found so far), the bound B at level $D - d$ is used instead at that level for node cutoffs. In the ABB algorithm, we backtrack from the leaf node with the initial bound B chosen by the SFFS to the root of the tree ($k = 0$) and record the J value at each tree level along the path. These J values are used as the initial bound b_{k+r} at level $k + r$. For all versions of the BB algorithm, the bound b_{k+r} is updated every time a larger value of J is found at level $k + r$.

4. Experimental Results

We discuss the effect of the look-ahead search strategy on the performance of the basic BB, ordered BB, FBB, and ABB algorithms. To compare the performance of different feature selection algorithms, we compute the criterion function J value for the feature subsets chosen and the number of J calculations required by each BB algorithm. The Mahalanobis distance is used as the criterion function J, since it is computationally efficient, monotonic as required by the BB algorithm, and used in prior feature selection comparisons [2, 7]. As noted earlier, a subset with a larger J value is better than one with a smaller J value. We record the number of J calculations to compare the computational complexities of different BB algorithms, since calculation of J represents the major computation needed because it requires a matrix inversion. All of our experiments were carried out using MATLAB7 on a Pentium IV-2.8 GHz computer with 4 GB of RAM.

We test the algorithms on two data sets from the UCI repository [10]: mammogram data from Wisconsin Diagnostic Breast Center and an SPECTF heart database. The data sets contain samples for two-class classification problems. The mammogram data have 30 features (D = 30) from 357 benign and 212



(a) (b)

Fig. 2: (a) The Mahalanobis distance criterion function J values and (b) the number of J calculations required by different BB algorithms to select $d = 11$ features for different look-ahead parameters r for the mammogram database.

malignant samples. The SPECTF heart data set describes diagnosis results of cardiac Single Proton Emission Computed Tomography images. It contains 44 features ($D = 44$) from 55 normal and 212 abnormal samples.

We ran the basic BB, ordered BB, FBB, and ABB algorithms with the look-ahead search strategy on the mammogram database ($D = 30$). In general, the J value increases as the look-ahead parameter r increases, especially for the basic BB algorithm. When $r = 0$, the basic BB algorithm approximates the sequential backward selection (SBS) algorithm, since it does not obtain a good initial bound B. Advanced BB algorithms, such as the ordered BB, FBB, and ABB algorithms, are capable of providing excellent J values even when the look-ahead parameter r is small. When the number of selected features d is small, the J values of the ordered BB, FBB, and ABB algorithms essentially overlap for almost all r values. When r is large, all BB algorithms give the same optimal solution as expected. However, ABB performs this with the smallest number of J calculations when the number of selected features d is small. This is expected, since [7] demonstrated that ABB is best when $d << D$. Next, we investigate the performance of each BB algorithm when d is large. Figs. 2a and 2b show the Mahalanobis distance criterion function J values obtained and the number of J calculations required by the different BB algorithms to select $d = 11$ features versus the look-ahead parameter r. From Fig. 2a, the ordered BB and FBB algorithms provide the optimal solution (J value = 14.07) when $r \geq 4$, whereas the basic BB and ABB algorithms do not until $r \geq 9$. From Fig. 2b, the number of J calculations required by the FBB algorithm is not monotonically increasing as r increases. This is due to the fact that different r values cause the search to cut off the nodes at different tree levels, which in turn effects the statistical prediction mechanism in the FBB algorithm. The ordered BB algorithm is the fastest algorithm when $r \geq 7$. The ABB algorithm is known to be slow when the condition $d << D$ is not satisfied (this is the case in Fig. 2). Thus, ABB is slower than the FBB algorithm when $r = 12$. When r is large, all BB algorithms approach their optimal versions. Even though all BB algorithms give the same optimal solution when $r \geq 9$, the basic BB algorithm requires a large number of J calculations. For instance, when $r = 12$, the basic BB algorithm requires approximately 400,000 J calculations. Thus, use of the basic BB algorithm is not advisable when d is not far smaller than D. Experiments for different d cases were also carried out, and we obtained similar trends.

Since the performance of each branch and bound algorithm depends on the database used, we tested the four BB algorithms on the SPECTF heart data set ($D = 44$). This is a more difficult data for which the dimensionality of the original feature space is large and for which there are a large number of possible subset combinations. Fig. 3 shows the results for the $d = 7$ case. The basic BB algorithm is the worst in terms of both the criterion function J values and the number of J calculations required (when $r > 9$). Use of the basic BB algorithm is not recommended for a large-dimensional data set. Advanced BB algorithms provide the optimal solution even when r is small, but the ordered BB algorithm now requires significantly more J calculations than the FBB and ABB algorithms. Again, the ABB algorithm outperforms other BB algorithms when $d << D$.

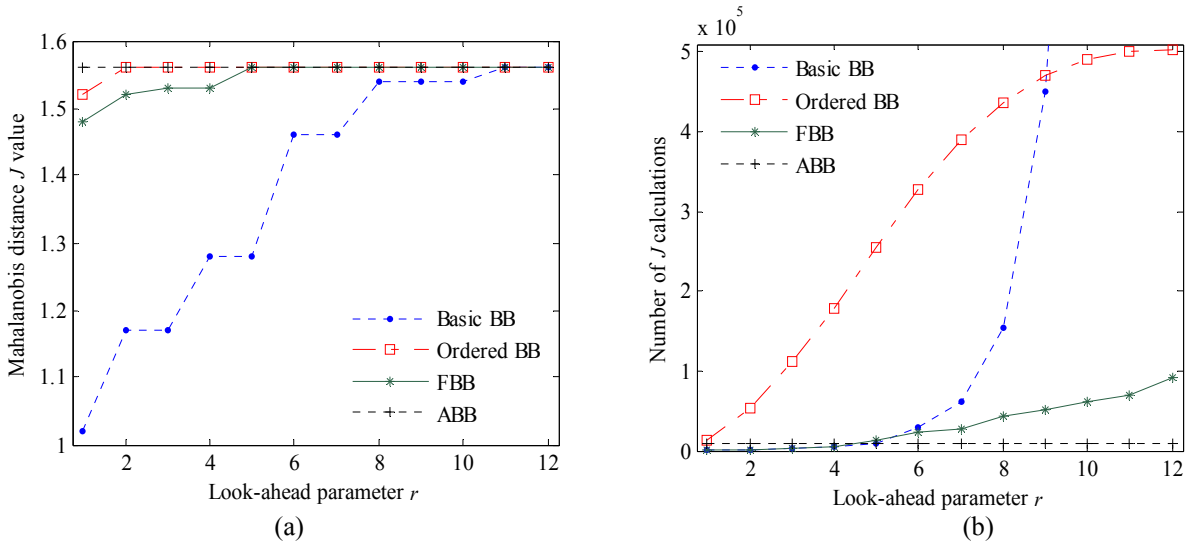


Fig. 3: (a) The Mahalanobis distance criterion function J values and (b) the number of J calculations required by different BB algorithms to select $d = 7$ features for different look-ahead parameters r for the SPECTF database.

5. Conclusions

We investigate the effect of the look-ahead search strategy on four different branch and bound algorithms. We test it on two real data sets from the UCI collection. Although the optimal solutions of the algorithm are no longer guaranteed, we find that the solutions provided by advanced BB algorithms such as the ordered BB, FBB, and ABB algorithms are optimal even when the look-ahead parameter r is small. On the other hand, the computational cost required by each BB algorithm is reduced significantly with the look-ahead search scheme. For the problem of selecting d features out of large D (≥ 50), we suggest that the look-ahead strategy should be incorporated into the FBB or ABB algorithm to speed up the search. Use of the basic BB algorithm is not recommended for a large-dimensional database.

6. References

[1] H. Liu, and M. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Norwell, MA: Kluwer Academic Publishers, 1998.

[2] M. Kudo, and J. Sklansky. Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition* 2000, **33**: 25-41.

[3] P.M. Narendra, and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. Comput.* 1997, **C-26** (9): 917-922.

[4] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. 2nd ed., San Diego, CA: Academic Press, Inc., 1990.

[5] B. Yu, and B. Yuan. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition* 1993, **26**: 883-889.

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

[7] S. Nakariyakul, and D. Casasent. Adaptive branch and bound algorithm for selecting optimal features. *Pattern Recognition Lett.* 2007, **28**: 1415-1427.

[8] S. Nakariyakul. On the suboptimal solutions using the adaptive branch and bound algorithm for feature selection. In: *Proceedings of the 2008 International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR)*. 2008, pp. 384-389.

[9] P. Pudil, J. Novovicova, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Lett.* 1994, **15**: 1119-1125.

[10] A. Asuncion, and D.J. Newman. UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. School of Information and Computer Science, Univ. of California, Irvine, CA, 2007.