Branch & Bound Algorithm with Partial Prediction For Use with Recursive and Non-Recursive Criterion Forms

Petr Somol, Pavel Pudil, and Jiří Grim

Dept. of Pattern Recognition, Inst. of Information Theory and Automation,
Academy of Sciences of the Czech Republic, 182 08 Prague 8, Czech Republic
e-mail: {somol, pudil, grim}@utia.cas.cz

Abstract. We introduce a novel algorithm for optimal feature selection. As opposed to our recent *Fast Branch & Bound* (FBB) algorithm [5] the new algorithm is well suitable for use with recursive criterion forms. Even if the new algorithm does not operate as effectively as the FBB algorithm, it is able to find the optimum significantly faster than any other Branch & Bound [1,3] algorithm.
Keywords: subset search, feature selection, search tree, recursive criteria, optimal search, subset selection.

1 Introduction

The problem of optimal feature selection (or more generally of subset selection) is difficult especially because of its time complexity. Any known optimal search algorithm has an exponential nature. The only alternative to the exhaustive search is the *Branch & Bound* (BB) algorithm [1,3] and ancestor algorithms based on a similar principle. Any BB algorithm requires the criterion function fulfilling the *monotonicity condition*. Let $\bar{\chi}_j$ be the set of features obtained by removing j features $y_1, y_2, \cdots, y_j$ from the set Y of all D features, i.e.

$$\bar{\chi}_j = \{\xi_i | \xi_i \in Y, 1 \leq i \leq D; \xi_i \neq y_k, \forall k\} \tag{1}$$

The *monotonicity condition* assumes that for feature subsets $\bar{\chi}_1, \bar{\chi}_2, \cdots, \bar{\chi}_j$, where

$$\bar{\chi}_1 \supset \bar{\chi}_2 \supset \cdots \supset \bar{\chi}_j$$

the criterion function J fulfills

$$J(\bar{\chi}_1) \geq J(\bar{\chi}_2) \geq \cdots \geq J(\bar{\chi}_j). \tag{2}$$

By a straightforward application of this property many feature subset evaluations may be omitted.

Before discussing the new algorithm, let us summarize the BB principle briefly. The algorithm constructs a search tree where the root represents the set of all D features and leaves represent target subsets of d features. While

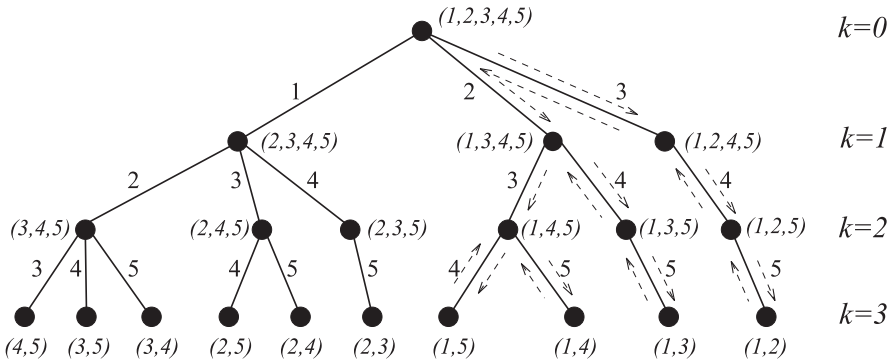

Fig. 1. Example of "branch & bound" problem solution, where $d = 2$ features are to be selected from the set of $D = 5$ features. The dashed arrows illustrate the way of tracking the search tree.

tracking the tree down to leaves the algorithm removes successively single features from the current set of "candidates" ($\bar{\chi}_k$ in the k-th level). The algorithm keeps the information about both the till-now best subset $\mathcal{X}$ and the criterion value X^* it yields (we denote this value the *bound*). Anytime the criterion value in some internal node is found to be lower than the current *bound*, due to the condition (2) the whole sub-tree may be cut-off and many computations may be omitted. The course of the BB algorithm is illustrated on Fig. 1. For details see [1,3,2].

Several improvements of this scheme are known: the "Improved" BB algorithm [3] utilizes a heuristic for ordering tree branches so as to find the optimum faster and therefore to allow more sub-tree cut-offs. The "Fast" BB algorithm [5] introduces a prediction mechanism being able to predict impossibility of cutting-off a sub-tree and therefore to save a significant number of computations.

2 Drawbacks of the Traditional Branch & Bound Algorithm

When compared to the exhaustive search, every BB algorithm requires additional computations. Not only the target subsets of d features $\bar{\chi}_{D-d}$, but also their supersets $\bar{\chi}_{D-d-j}$, $j = 1,\ 2, \cdots, D-d$ have to be evaluated.

The BB principle does not guarantee that enough sub-trees will be cut-off to keep the total number of criterion computations lower than their number in exhaustive search. The worst theoretical case would arise when we defined a criterion function $J(\bar{\chi}_k) = |\bar{\chi}_k| \equiv D - k$; the criterion function would be computed not only in every leaf (the same number of computations as in exhaustive search), but additionally also in every other node inside the tree.

Weak BB performance in certain situations may result from simple facts that nearer to the root: a) criterion value computation is usually slower (evaluated feature subsets are larger), b) sub-tree cut-offs are less frequent nearer the root (higher criterion values may be expected for larger subsets, which reduces the chance of the criterion value to remain under the *bound*, which is updated in leaves). The BB algorithm usually spends most of time by tedious, but less promising evaluation of tree nodes near the root. This effect is to be expected especially for $d \ll D$. In case of the "Improved" BB algorithm a significant number of additional computations is needed for ordering internal search tree node descendants. The advantage following from these computations may become questionable, because a slightly better heuristic organization of the search tree is often outweighted by the additional computational time.

A very effective way of resolving BB disadvantages offers the FBB algorithm, which is able to replace a large number of computations by means of prediction. Although the FBB algorithm requires usually several times less criterion computations than any other BB algorithm, its suitability for many practical problems is limited if the recursive criterion forms are to be used. To resolve this limitation we define a new, more universal algorithm.

3 Improving the "Improved" Algorithm

Let's focus on the "Improved" BB algorithm heuristics for ordering the internal tree node descendants. Let the *criterion value decrease* be the difference between the current criterion value and the value after the removal of a particular feature. Let *bad* features be those features, whose removal from the current candidate set causes only a slight *criterion value decrease*. Let *good* features be those ones, whose removal from the current candidate set causes a significant *criterion value decrease*. (At this stage there is no need to quantify what a slight or significant decrease is).

In this explanation we assume that the BB algorithm constructs a search tree with a given topology (e.g. the "minimum solution tree" described by Yu and Yuan [4]). It is apparent that given the search tree topology, different feature assignments to the tree edges may be defined. The "Improved" algorithm aims to position *bad* features to the right, less dense part of the tree and *good* features to its left, more dense part . Based on such ordering we may expect faster *bound* increase, because preferred removal of *bad* features should keep the candidate criterion value higher. Consequently, removing *good* features from later candidate sets in the left, dense part of the tree gives better chance to decrease the criterion value under the *bound* and therefore to allow more effective sub-tree cut-offs.

The "Improved" BB algorithm operates approximately twice as fast as the "Basic" BB algorithm in most practical problems. However, the ordering heuristic requires a significant number of additional computations. Let's il-

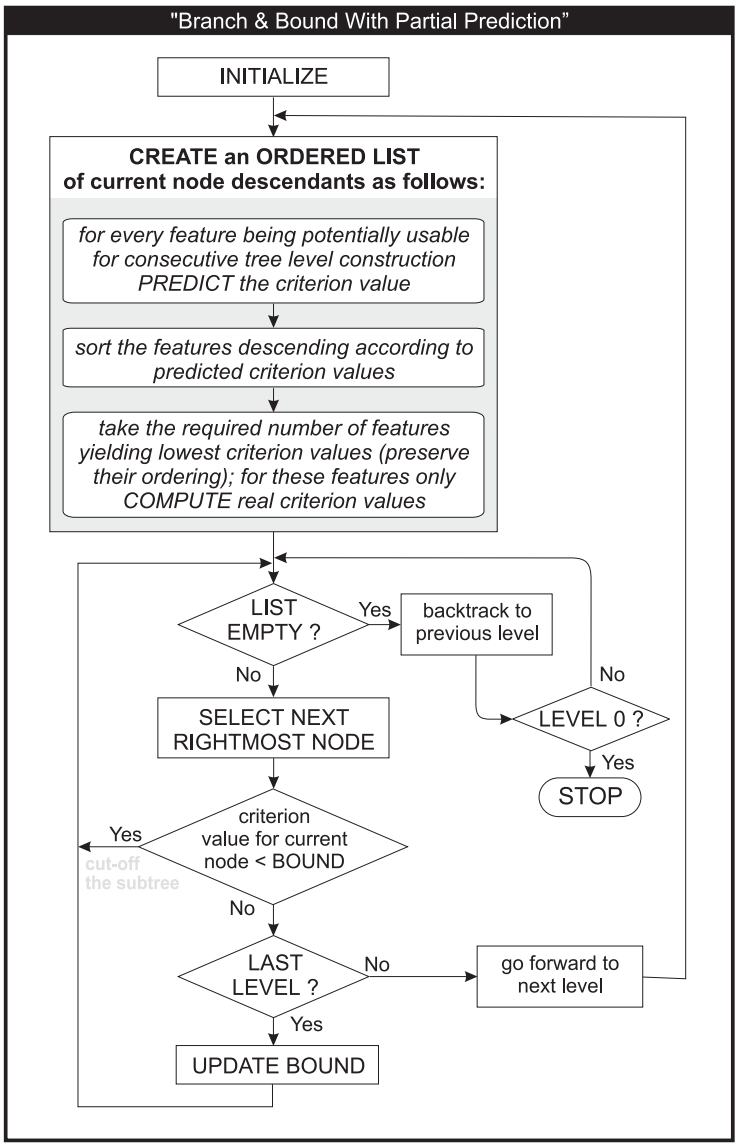

Fig. 2. Simplified diagram of the new algorithm

lustrate this drawback on Fig. 1 – when constructing the first level, i.e. when specifying the ordering of root descendants, the "Improved" algorithm evaluates the *criterion value decrease* for every available feature (all 5 features), although only 3 features are to be assigned to first level edges.

Our intention is to find the same (or very similar) ordering of tree nodes as given by means of the "Improved" BB algorithm with reduced number of

criterion evaluations. To achieve this goal we utilize the prediction mechanism as defined for purposes of the FBB algorithm. The new algorithm will construct the consecutive tree levels in several phases. First the *criterion value decrease* will be *predicted* for every feature being currently available for the tree construction. The features will be sorted descending according to the *predicted criterion value decreases*. Then, the required number of features (beginning from the feature with highest *predicted criterion value decrease*) will be taken to form the consecutive tree level.

Different features appear in different search tree construction stages, therefore we need to collect the prediction information separately for every feature. First we introduce a *vector of feature contributions to criterion value* for storing the individual information about average *criterion value decrease* caused by removing single features from current "candidate" subsets. Next we introduce a *counter vector* recording the number of *criterion value decrease* evaluations for every individual feature.

4 Branch & Bound with Partial Prediction (BBPP)

Our algorithm description is based on the notion from book [2]. We will use following symbols:

constants:

D – number of all features,

d – required number of selected features,

other symbols:

Y – set of all D features,

$J(.)$ – criterion function,

k – tree level ($k = 0$ denotes the root),

$\bar{\chi}_k = \{\xi_j \mid j = 1, 2, \cdots, D - k\}$ – current "candidate" feature subset in k-th tree level,

q_k – number of current node descendants (in consecutive tree level),

$\mathcal{Q}_k = \{Q_{k,1}, Q_{k,2}, \ldots, Q_{k,q_k}\}$ – ordered set of features assigned to edges leading to the current node descendants (note that "candidate" subsets $\bar{\chi}_{k+1}$ corresponding to the current node descendants are fully determined by features $Q_{k,i}$ for $i = 1, \cdots q_k$),

$\mathbf{J}_k = [J_{k,1}, J_{k,2}, \ldots, J_{k,q_k}]^{\mathrm{T}}$ – vector of criterion values corresponding to the current node descendants in consecutive tree level ($J_{k,i} = J(\bar{\chi}_k \setminus \{Q_{k,i}\})$ for $i = 1, \cdots, q_k$),

$\Psi = \{\psi_j \mid j = 1, 2, \cdots, r\}$ – control set of r features being currently available for search-tree construction, i.e. for building consecutive descendant vector $\mathcal{Q}_k$; the Ψ set serves for maintaining the search tree topology,

$\mathcal{X} = \{x_j \mid j = 1, 2, \cdots, d\}$ – current best subset of d features

X^* – current *bound* (criterion value corresponding to $\mathcal{X}$),

$\mathbf{A} = [A_1, A_2, \ldots, A_D]^{\mathrm{T}}$ – *vector of feature contributions to criterion value*,

$\mathbf{S} = [S_1, S_2, \ldots, S_D]^{\mathrm{T}}$ – *counter vector* (together with $\mathbf{A}$ serves for prediction)
Remark: it is necessary to store all values q_j, ordered sets $\mathcal{Q}_j$ and vectors $\mathbf{J}_j$ for $j = 0, \cdots, k$ during the algorithm course to allow backtracking.

The algorithm is to be initialized as follows:
$k = 0$ (starting in the root),
$\bar{\chi}_0 = Y$,
$\Psi = Y$, $r = D$
X^* – lowest possible value (computer dependent)
$S_i = 0$ for all $i = 1, \cdots, D$.

The BBPP Algorithm

Whenever the algorithm removes some feature y_i from the current "candidate" subset and computes the corresponding real criterion value $J(\bar{\chi}_k \setminus \{y_i\})$ in k-th tree level, use the difference $J(\bar{\chi}_k) - J(\bar{\chi}_k \setminus \{y_i\})$ for updating the *prediction information*. Let

$$A_{y_i} = \frac{A_{y_i} \cdot S_{y_i} + J(\bar{\chi}_k) - J(\bar{\chi}_k \setminus \{y_i\})}{S_{y_i} + 1} \tag{3}$$

and let

$$S_{y_i} = S_{y_i} + 1 \tag{4}$$

STEP 1: *Select descendants of the current node to form the consecutive tree level:* first set their number to $q_k = r-(D-d-k-1)$. Construct an ordered set $\mathcal{Q}_k$ and vector $\mathbf{J}_k$ specifying the current node descendants as follows: sort all features $\psi_j \in \Psi, j = 1, \cdots, r$ descending according to their $A_{\psi_j}, j = 1, \cdots, r$ values, i.e.

$$A_{\psi_{j_1}} \geq A_{\psi_{j_1}} \geq \cdots \geq A_{\psi_{j_r}}$$

and choose successively first q_k features among them, i.e. let
$Q_{k,i} = \psi_{j_i}$ for $i = 1, \cdots, q_k$
$J_{k,i} = J(\bar{\chi}_k \setminus \{\psi_{j_i}\})$ for $i = 1, \cdots, q_k$
To avoid future duplicate testing, features ψ_{j_i} cannot be used for construction of consecutive tree levels, so let $\Psi = \Psi \setminus Q_k$ and $r = r - q_k$

STEP 2: *Test the right-most descendant node (connected by the Q_{k,q_k}-edge):* if $q_k = 0$, all descendants were tested, go to **Step 4** (backtracking). If $J_{k,q_k} < X^*$, then go to **Step 3**. Else let $\bar{\chi}_{k+1} = \bar{\chi}_k \setminus \{Q_{k,q_k}\}$. If $k+1 = D-d$, then you have reached a leaf, go to **Step 5**. Otherwise go to the consecutive level: let $k = k + 1$ and go to **Step 1**.

STEP 3: *Descendant node connected by the Q_{k,q_k}-edge (and its possible subtree) may be cut-off:* return feature Q_{k,q_k} to the set of features available for

tree construction, i.e. let $\Psi = \Psi \cup \{Q_{k,q_k}\}$ and $r = r + 1$, $\mathcal{Q}_k = \mathcal{Q}_k \setminus \{Q_{k,q_k}\}$ and $q_k = q_k - 1$ and continue with its left neighbor; go to **Step 2**.

STEP 4: *Backtracking:* Let $k = k - 1$. If $k = -1$, then the complete tree had been searched through; stop the algorithm. Otherwise return feature Q_{k,q_k} to the set of "candidates": let $\bar{\chi}_k = \bar{\chi}_{k+1} \cup \{Q_{k,q_k}\}$ and go to **Step 3**.

STEP 5: *Actualize the bound value:* Let $X^* = J_{k,q_k}$. Store the currently best feature subset $\mathcal{X} = \bar{\chi}_{k+1}$ and go to **Step 2**.

Remark: In Step 1 for $k = 0$ the term $J_{-1,q_{-1}}$ denotes the criterion value on a set of all features, $J(Y)$.

5 New Algorithm Properties

The algorithm may be expected to be most effective, if the individual feature contribution to the criterion value does not change strongly in relation to different subsets. Practical tests on real data fulfilled this property in most of cases. Moreover, the BBPP algorithm proved to be effective even in cases, when due to difficult statistical dependencies individual feature contributions failed to remain stable.

When compared to the FBB algorithm, the BBPP may be expected to be more robust. A potential failure of the prediction mechanism would have only indirect influence on the overall algorithm performance. A potentially wrong ordering of internal tree nodes (i.e. assigning of features to edges) would eventually decrease the efficiency of sub-tree cut-offs, but on the other hand the basic advantage over the "Improved BB" algorithm – reducing the number of additional computations – remains preserved.

When compared to both "Basic" and "Improved" algorithms the BBPP always spends some additional time for maintaining the prediction mechanism. However, this time proved not to be important in case of non-recursive criterion forms, while in case of faster recursive criterion forms it still proved to be short enough to ensure overall algorithm speedup. Moreover, especially for use with recursive criterion forms attempts to define even simpler prediction mechanisms to save computational time (e.g. to utilize the last known feature contribution to criterion value only) have been made with promising results.

Remark: To ensure good results we recommend to evaluate the *individual feature contributions to criterion value* once for all features in the initial algorithm phase. This will ensure a correct start of the prediction mechanism. Moreover, the first search tree level may then be constructed in the same way as in the "Improved" BB, what may prove to be advantageous for later algorithm phases.

6 Experiments

The algorithms were tested on a number of different data sets. Here we present representative results computed on 30-dimensional mammogram data (2 classes – 357 benign and 212 malignant samples) obtained from Wisconsin Diagnostic Breast Center via the UCI repository - ftp.ics.uci.edu. We used both the recursive and non-recursive Bhattacharyya distance as the criterion function. Performance of different methods is illustrated on Fig. 3 and Fig. 4 by a graph of total computational time and a graph of criterion evaluation numbers. We did not include the graph of criterion values, because all the methods yield the same optimum values.

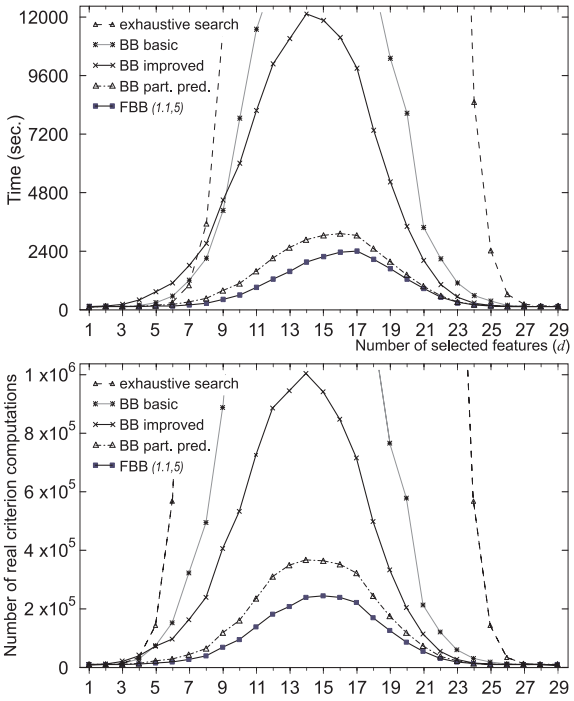

Fig. 3. Example: Optimal subset search methods performance when maximizing the **non-recursive** Bhattacharyya distance on 30-dimensional data (Wisconsin Diagnostic Breast Center). Results computed on a Pentium II-350 MHz computer.

We compare all the results especially with the results of the "Improved" BB algorithm [3,2], because this algorithm is generally accepted to be the most effective optimal subset search strategy. In case of non-recursive criterion functions we compare the new algorithm also with our recent FBB algorithm. Note that in case of non-recursive criterion computations we implemented all BB algorithms so as they construct the "minimum solution tree" [4].

Although the FBB algorithm usually finds the optimum after the smallest number of computations, its principle prevents it to be used with recursive

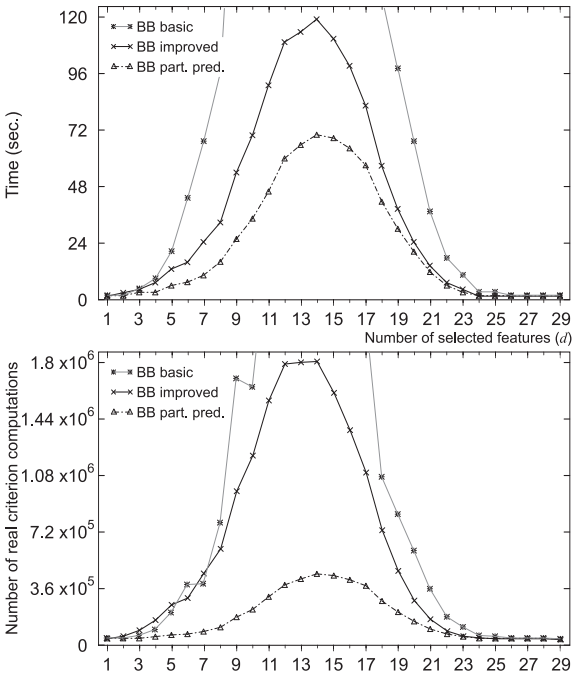

Fig. 4. Example: Optimal subset search methods performance when maximizing the **recursive** Bhattacharyya distance on 30-dimensional data (Wisconsin Diagnostic Breast Center). Results computed on a Pentium II-350 MHz computer.

criterion functions. The graphs on Fig. 3 and Fig. 4 illustrate that the BBPP operates faster than the "Improved" BB when used both with non-recursive or recursive criterion forms. According to expectations, being used with recursive criterion forms the new algorithm brings a less significant speedup; the computational complexity of recursive criterion forms is usually significantly lower than in non-recursive case, although the computational time spend by the prediction mechanism remains the same.

Remark: When used with recursive criterion form, no BB algorithm may utilize the "minimum solution tree" [4] due to the necessity to preserve criterion value computation sequence. The minimum solution tree assumes shortening of straight paths to leaves, what breaks the criterion computation sequence. Because of this reason numbers of computations differ in recursive and non-recursive case.

Both for the FBB and BBPP a slight shift of their graphs to the right may be observed when compared to the "Improved" BB algorithm. The prediction mechanism based algorithm acceleration relates to the number of criterion evaluation savings in internal search tree nodes, therefore with decreasing d the search tree depth increases and allows more effective operation of the prediction mechanism.

The majority of experiments produced results similar to those on Fig. 3 and Fig. 4. In one isolated worst case the speed of the BBPP used with recursive criterion form remained comparable to the speed of "Improved" BB.

Theoretically we can not exclude the prediction mechanism failure – if the individual feature contributions to criterion value were unstable, i.e. changed too often and too strongly, the BBPP operation could become comparable with the "Basic BB" algorithm. However, we have not met such situation in our experiments.

7 Conclusion

We defined a new algorithm for optimal subset search. Its prediction mechanism allows significant time savings when compared to "Basic" or "Improved" Branch & Bound algorithms [3]. The algorithm was experimentally proved to be robust and well suitable for use with different criterion functions, both in recursive and non-recursive form.

Acknowledgement: The work has been supported by the grants of Czech Ministry of Education MŠMT No.VS96063, ME187, CEZ:J18/98:311600001 and Academy of Sciences K1075601.

References

1. Narendra P. M., Fukunaga K. (1977) A branch and bound algorithm for feature subset selection. IEEE Transactions on Computers, C-**26**, 917–922
2. Devijver P. A., Kittler J. (1982) Pattern Recognition: A Statistical Approach. Prentice-Hall
3. Fukunaga K. (1990) Introduction to Statistical Pattern Recognition: 2nd edition. Academic Press, Inc.
4. Yu B., Yuan B. (1993) A more efficient branch and bound algorithm for feature selection. Pattern Recognition, **26**, 883–889
5. Somol P., Pudil P., Ferri F. J., Kittler J. (2000) Fast Branch & Bound Algorithm in Feature Selection. Proc 4th World Multiconference on Systemics, Cybernetics and Informatics SCI 2000, Orlando, Florida, Vol VII, Part 1, 646–651