Fast Branch & Bound Algorithm in Feature Selection

Petr Somol and Pavel Pudil
Department of Pattern Recognition, Institute of Information Theory and Automation,
Academy of Sciences of the Czech Republic, 182 08 Prague 8, Czech Republic

and

Francesc J. Ferri
Departament d'Informàtica , Universitat de València, Dr Moliner, 50 46100 Burjassot (València), Spain

and

Josef Kittler
Centre for Vision, Speech and Signal Processing, University of Surrey, Guildford GU2 7XH. UK

ABSTRACT

We introduce a novel algorithm for optimal subset selection. Due to its simple mechanism for predicting criterion values the algorithm finds optimum usually several times faster than any other known Branch & Bound [5], [7] algorithm. This behavior is expected when the algorithm is used in conjunction with non-recursive and/or computationally expensive criterion functions.

Keywords: Subset Search, Feature Selection, Search Tree, Optimum Search, Subset Selection.

1. INTRODUCTION TO THE BRANCH & BOUND PRINCIPLE

The problem of optimal feature selection is difficult especially because of its computational complexity. The exhaustive search procedure is applicable to lower-dimensional problems only. A well known alternative to exhaustive search is the Branch & Bound (BB) algorithm presented first in 1977 by Narendra and Fukunaga [1]. The BB is a "top-down" algorithm with backtracking. It is based on the assumption that the adopted criterion function fulfills 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.

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

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

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

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

For better understanding let us recall the BB principle first. Consider the problem of selecting $d = 2$ out of $D = 5$ features. Figure 1 illustrates the way the BB constructs its search tree. Leaves represent target subsets of d features, while the root represents the set of all features, Y. The tree construction is illustrated by the dashed arrows. The digits associated with edges in Figure 1 denote features being removed from the current "candidate" set while the algorithm tracks the edge down (and being returned back while the algorithm backtracks up). Nodes in the k-th level represent current subsets of $D - k$ features. For example, the *- node represents a set containing features y_1, y_3, y_4, y_5 obtained from the previous set by removing feature y_2. Every time the algorithm reaches a leaf node, the corresponding criterion value is used to update the *bound* (the current maximum). On termination of the algorithm the *bound* will contain the optimum criterion value.

The BB algorithm's advantage over an exhaustive search derives from the ability to omit the construction of certain search tree branches. Consider a situation, where the algorithm reaches the *-node in Figure 1. The *bound* has been updated recently according to the target subset containing features y_1, y_2. There is a chance that the criterion value computed for the current subset (y_1, y_3, y_4, y_5) would be lower than the current *bound*. Because of the *monotonicity condition* (2) nowhere in *-node sub-tree the criterion value may exceed the *bound*. Therefore the sub-tree construction is unnecessary (sub-tree would be *cut-off*), thus saving time.

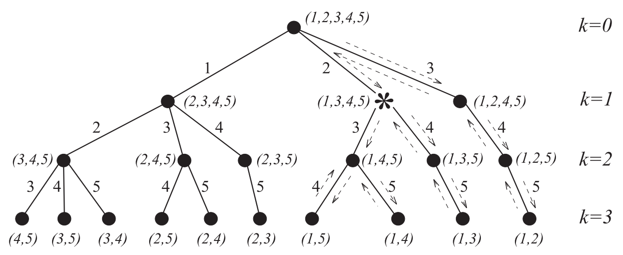

Fig. 1. *Branch & Bound search tree.*

It is apparent that for a search tree of given topology there exist many different feature orderings (assignments to edges). The assignment is to be specified whenever the algorithm constructs a consecutive tree level, i.e. selects immediate descendants of current node (the *-node in Figure 1 has two descendants representing subsets y_1, y_4, y_5 and y_1, y_3, y_5 obtained by removing feature y_3 or y_4, respectively). Ordering the descendants according to their criterion value may result in faster increase in the *bound* value, more sub-tree cut-offs and eventually in faster algorithm operation (see [2], [5], [6], [3]).

The algorithm could also be modified to construct the *minimum search tree* described by Yu *et al.* [7] and other improvements as using approximate monotonicity [4].

2. DRAWBACKS OF THE BRANCH& BOUND ALGORITHM

Let us consider the use of BB algorithm with criterion functions, whose computational complexity is high and at least polynomially dependent on the current subset size (e.g. non-recursive forms of standard probabilistic distance measures like Bhattacharyya, etc.).

When compared to exhaustive search, every BB algorithm requires additional computations in terms of criterion evaluations. Not only 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 if 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, less promising evaluation of tree nodes near the root. This effect is to be expected especially when $d \ll D$. Based on this observation we introduce a novel algorithm, which effectively resolves the above mentioned drawback.

3. FAST BRANCH & BOUND

The Fast Branch & Bound (FBB) algorithm aims to reduce the number of criterion function computations in internal search tree nodes. The algorithm requires the criterion function to fulfill the *monotonicity condition* (2) just like the BB algorithm.

The simplified algorithm description would be as follows: the algorithm attempts to utilize the knowledge of criterion value changes after removal of single features for future prediction of criterion values without the need for their computation. Prediction is allowed under certain circumstances only, e.g. not in leaves. Both the really computed and predicted criterion values are treated in the same way, i.e. for ordering of node descendants during the tree construction phase.

If the predicted criterion value remains significantly higher than the current *bound*, we may expect that even the actual value would not be lower and therefore the corresponding sub-tree could not be cut-off. In this situation the algorithm continues to construct the consecutive tree level. However, if the predicted value comes close to the *bound* (and therefore there arises a chance that the real value is lower than the *bound*), the real criterion value must be computed. Only if real criterion values are lower than the current *bound*, sub-trees may be cut-off. Note that this prediction scheme does not affect the optimality of the obtained results. The course of FBB algorithm remains similar to BB, possible sub-tree cut-offs are allowed according to real criterion values only. Possible inaccurate predictions may result in nothing worse than constructing sub-trees, which would have been pruned out by means of classical BB algorithm. However, this situation is usually strongly outweighed by criterion computation savings in other internal nodes, especially near the root, where the criterion computation tends to be slower.

The prediction mechanism utilizes the information about the averaged criterion value change caused by removing a particular feature. Obviously, the following trade-off arises: we want to use prediction as frequently as possible to save time, on the other hand we have to compute the actual criterion values to precise (or at least to estimate) the information needed for the prediction mechanism. Different features appear in different search tree construction stages, therefore we need to collect the prediction information separately for every feature.

The FBB Search Tree Construction

First we introduce a *vector of feature contributions to the criterion value* for storing the information about the average criterion value change caused by removing single features from the current subsets. Next we introduce a *counter vector* recording the number of criterion change evaluations for every individual feature. Before any criterion evaluation the record in *counter vector* is checked for the given feature. If the prediction mechanism has accumulated enough information (the counter value is bigger than a pre-specified *minimum number of feature contribution evaluations* constant, see section 4), the required criterion value will be predicted, otherwise it will be computed.

Since the information about the significance of the criterion value has to be maintained at each internal node, we introduce a vector of value types for nodes in the current tree level. If the criterion value was predicted, the pertinent type is denoted by “P”. If the criterion value was actually computed, the type is denoted by “C”. The difference between criterion values computed in current internal node and its parent node is used to update the information in the *vector of feature contributions to the criterion value* only if both the criterion values have been really computed (both for the current node and its parent node the “C” is recorded in the *vector of value types*). Should we attempt to update

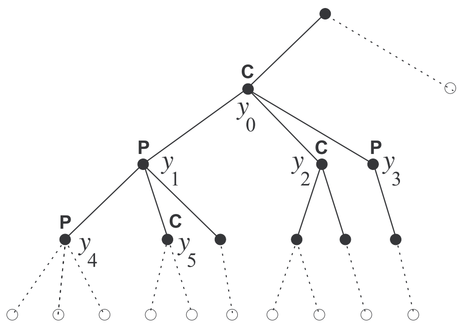

Fig. 2. A search tree situation: Criterion value has been computed for nodes y_0, y_2, y_5 and predicted for nodes y_1, y_3 and y_4. The prediction information in the *vector of feature contribution* may be updated according to the difference of criterion values in y_0 and y_2 only. Criterion values in other nodes are unusable for updating.

the vector of feature contributions by a difference of values of mixed or predicted types, we would deteriorate the accuracy of prediction mechanism (by bringing in and multiplying the estimation error). The prediction mechanism is limited by a lack of information in early phases of the FBB algorithm. We may therefore expect frequent updates of the vector of feature contributions. Later, with the increasing number of “reliable” features (whose contribution to the criterion value has been evaluated for more than required minimum number of times, as indicated in the *counter vector*) criterion values are obtained by the prediction mechanism instead of actual computation more often and the updating process becomes less frequent. An example of a search tree situation is illustrated in Figure 2.

The aim of the described prediction mechanism is not only to reduce the number of criterion value computations at the internal nodes when the algorithm explores the tree down to the leaves, but also to estimate the proper moment (node), when the criterion value decreases below the current *bound* and the possibility to cut-off a sub-tree arises. Let us denote the case when the algorithm stops the prediction too soon (and the computed values still remain higher than the current *bound*) as a *pessimistic prediction error*. Further, we denote the case when the algorithm utilizes a prediction for too long (misses the real possibility of cutting-off current sub-tree and continues to construct consecutive tree level) as an *optimistic prediction error*. The prediction mechanism behavior may be adjusted by introducing an *optimism constant* in the following way: let every value from the *vector of feature contribution* be multiplied by the optimism constant before being used for prediction. Higher values of the *optimism constant* would protect the algorithm from missing real possibilities to cut-off sub-trees, but on the other hand the predicted criterion values would decrease faster, reach the current *bound* faster and the prediction mechanism would stop sooner than necessary.

A simplified flowchart of the FBB algorithm may be found on Figure 3.

4. FAST BRANCH & BOUND ALGORITHM DESCRIPTION

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,

$\delta \geq 1$ – *minimum required number of feature contribution evaluations*, this integer constant affects the start of prediction mechanism, (see section 4),

$\gamma \geq 0$ – optimism constant, (see Section 4),

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 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 $i = 1, \cdots, q_k$),

$\mathbf{T}_k = [T_{k,1}, T_{k,2}, \ldots, T_{k,q_k}]^{\mathrm{T}},\ T_{k,i} \in \{“C'', “P''\}$ for $i = 1, \cdots, q_k$ – criterion value type vector (records the type of corresponding $J_{k,i}$ values),

$\Psi = \{\psi_j \mid j = 1, 2, \cdots, r\}$ – 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,

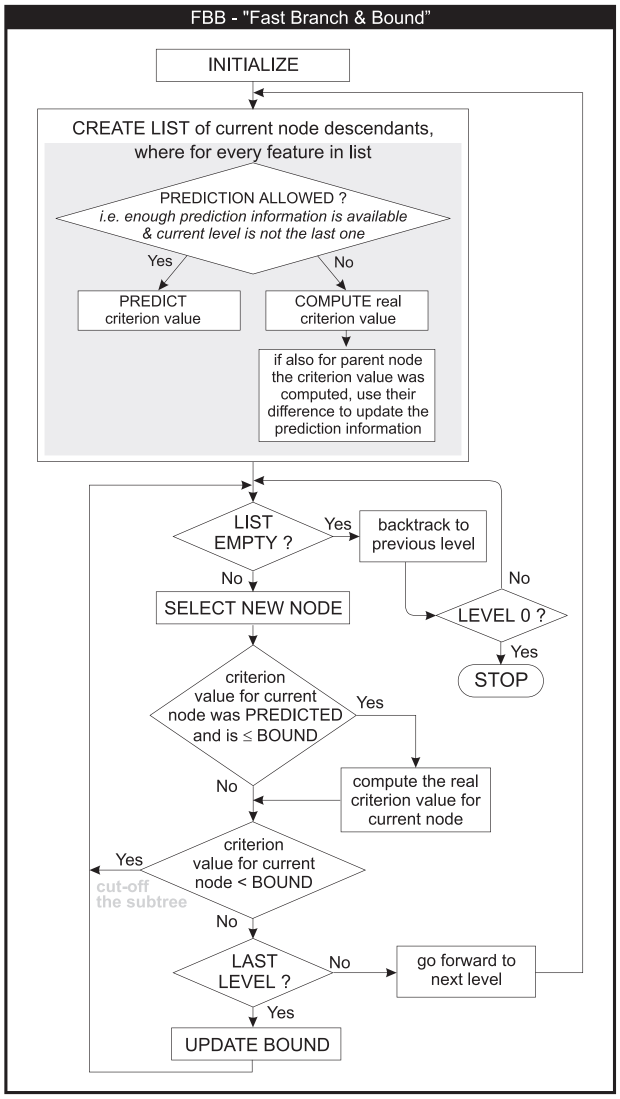
FBB - "Fast Branch & Bound"
INITIALIZE
CREATE LIST of current node descendants, where for every feature in list
PREDICTION ALLOWED ? i.e. enough prediction information is available & current level is not the last one
Yes
No
PREDICT criterion value
COMPUTE real criterion value
if also for parent node the criterion value was computed, use their difference to update the prediction information
LIST EMPTY ?
Yes
backtrack to previous level
No
LEVEL 0 ?
Yes
STOP
No
SELECT NEW NODE
criterion value for current node was PREDICTED and is ≤ BOUND
Yes
compute the real criterion value for current node
No
criterion value for current node < BOUND
Yes
cut-off the subtree
No
LAST LEVEL ?
No
go forward to next level
Yes
UPDATE BOUND

Fig. 3. *A simplified diagram of the FBB algorithm*

$\mathcal{X} = \{x_j \mid j = 1, 2, \cdots, d\}$ – 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 the criterion value,*
$\mathbf{S} = [S_1, S_2, \ldots, S_D]^{\mathrm{T}}$ – *counter vector* (together with $\mathbf{A}$ serves for prediction),
$\mathbf{V} = [v_1, v_2, \ldots, v_{q_k}]^{\mathrm{T}}$ – temporary vector for sorting purposes only,

Remark: it is necessary to store all values q_j, ordered sets $\mathcal{Q}_j$ and vectors $\mathbf{J}_j$ and $\mathbf{T}_j$ for $j = 0, \cdots, k$ during the algorithm run 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 $i = 1, \cdots, D$
$\delta = 5$ (see section 4)
$\gamma = 1.1$ (see section 4).

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, and if also the predecessor value $J(\bar{\chi}_k) \equiv J(\bar{\chi}_{k-1} \setminus \{y_j\})$ (after previous removal of some feature y_j) had been computed (as indicated by T_{k-1,y_j} ="C"), then update the information in the *vector of feature contributions to the criterion value*, $\mathbf{A}$, as follows:

$$A_{y_i} = \frac{A_{y_i} \cdot S_{y_i} + J_{k-1,y_j} - J(\bar{\chi}_k \setminus \{y_i\})}{S_{y_i} + 1}$$

and let

$$S_{y_i} = S_{y_i} + 1$$

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 vectors $\mathbf{J}_k$ and $\mathbf{T}_k$ specifying the current node descendants in the following way: for every feature $\psi_j \in \Psi, j = 1, \cdots, r$ if $k+1 < D-d$ (nodes are not leaves) and $S_{\psi_j} > \delta$ (prediction allowed), let

$$v_j = J_{k-1,q_{k-1}} - A_{\psi_j}$$

i.e. predict by subtracting the appropriate prediction value based on ψ_j feature from the criterion value obtained in the parent node, otherwise the value must be computed, i.e. let

$$v_j = J(\bar{\chi}_k \setminus \{\psi_j\}).$$

After obtaining all v_j values sort them in the ascending order

$$v_{j_1} \leq v_{j_2} \leq \cdots \leq v_{j_r}$$

and for $i = 1, \cdots, q_k$ let

$Q_{k,i} = \psi_{j_i}$

$J_{k,i} = v_{j_i}$ if v_{j_i} records a computed value

$J_{k,i} = J_{k-1,q_{k-1}} - \gamma \cdot A_{\psi_{j_i}}$

if v_{j_i} records a predicted value

$T_{k,i}$ ="C" if v_{j_i} records a computed value

$T_{k,i}$ ="P" if v_{j_i} records a predicted value

To avoid future duplicate testing, features ψ_{j_i} in $\mathcal{Q}_k$ cannot be used for the construction of consecutive tree levels, so let $\Psi = \Psi \setminus \mathcal{Q}_k$ 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 T_{k,q_k} ="P" and $J_{k,q_k} < X^*$, compute the real value $J_{k,q_k} = J(\bar{\chi}_k \setminus \{Q_{k,q_k}\})$ and mark T_{k,q_k} ="C". If T_{k,q_k} ="C" and $J_{k,q_k} < X^*$, 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 next 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 sub-tree) 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: *Update 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 for the set of all features, $J(Y)$.

5. FAST BRANCH & BOUND ALGORITHM PROPERTIES

The algorithm may be expected to be most effective when the individual feature contributions to the criterion value do not change strongly in relation to different subsets. Practical tests on real data exhibit this property in the majority of cases. The FBB showed to be effective even in cases, when due to difficult statistical dependencies individual feature contributions failed to remain stable.

When compared to classical BB algorithms the FBB algorithm always spends additional time for maintaining the prediction mechanism. However, this additional time showed not to be important, especially when compared to time savings arising from the pruned criterion computations.

Obviously, the prediction mechanism of the FBB algorithm may fail to save computational time under certain circumstances. The FBB is not suitable for use with recursive criteria, where calculating $J(\bar{\chi}_k)$ value requires the knowledge of previous value $J(\bar{\chi}_{k-1})$. The use of prediction in FBB leads to performing the criterion calculations not level by level but in different stages. This makes it impossible to calculate successive criterion values recursively. In general, the FBB algorithm shows to be most advantageous when used with computationally demanding non-recursive criterion functions with higher than linear computational complexity (in terms of the feature set size).

The prediction mechanism may be influenced by altering the δ (*minimum required number of feature contribution evaluations*) and γ (*optimism*) constants.

Setting the δ value greater than 1 may be advantageous for protection against cases, where the first evaluated feature contribution to the criterion value is not typical and therefore the early start of prediction mechanism based on this information would cause too inaccurate predictions. On the other hand the δ value should remain close to 1, because its increase would slow-down the start of the prediction mechanism. A better estimate of feature contributions resulting from setting high δ usually cannot outweigh the time lost by prediction start delay. A practical setting for general use verified to be $\delta = 5$.

The γ value affects the "optimism" of the prediction mechanism. Values $\gamma > 1$ reduce the occurrence of optimistic prediction errors. Values $0 < \gamma < 1$ reduce the occurrence of pessimistic prediction errors. For $\gamma = 0$ the algorithm collects the required prediction information and then completes the search in a way equivalent to exhaustive search. Increasing the γ value shifts the algorithm functionality nearer to classical Branch & Bound. A practical setting for general use proved to be $\gamma = 1.1$. This slightly "pessimistic" value serves as a protection against too inaccurate prediction which could shift the algorithm functionality to exhaustive search.

6. EXPERIMENTS

The reference BB and FBB 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 the Wisconsin Diagnostic Breast Center via the UCI repository - ftp.ics.uci.edu. We used the non-recursive Bhattacharyya distance as the criterion function. Different performance of different methods is illustrated on Figure 4 by (a) graph of total computational time and (b) graph of criterion evaluation number. We did not include the graph of criterion values, because all methods yield the same optimum values.

We compare all the results especially to the results of *Improved Branch & Bound* [5], [2], because this algorithm is supposed to be the most effective known optimal subset search strategy. Note that we implemented all Branch & Bound algorithms so that they construct the *minimum solution tree* described by Yu and Yuan [7].

Computational complexity of the exhaustive search is well known, therefore we do not pay much attention to it. Let us note only that in our example the exhaustive search required for $d = 15$ approximately $140\times$ more criterion evaluations than the *Improved Branch & Bound*. Figure 4 illustrates also the *Basic Branch & Bound* algorithm [5]. Confirming our expectations its effectiveness is inferior to any other BB algorithm.

Graphs show a significantly higher effectivity of the *Fast Branch & Bound* when compared to any Branch & Bound. Note the slight FBB graph shift to the right, which follows from the FBB principle; the FBB algorithm is based on saving the criterion computations in internal search tree nodes, therefore a more significant saving of computations may be expected for deeper search trees (lower d).

The difference between FBB and Improved BB in (b) is a bit lower. It may be expected that due to prediction errors the FBB constructs its search tree less effectively (in the sense of feature order specification). Its main advantage is overall reduction in the criterion computations (especially more tedious computations nearer to search tree root), which caused approximately its $3 - 6\times$ faster operation when compared to the Improved BB in our example (specifically for $d \approx 7$ the FBB run $20\times$ faster).

From our experience, the FBB algorithm operates significantly faster than any other Branch & Bound algorithm (provided we do not use recursive criteria). This is apparent even in situations, when classical Branch & Bound fails to operate faster than exhaustive search. The FBB operates still faster (with exception of d values near to 0 or D preventing the prediction mechanism to obtain enough information to start). The algorithm seems to be robust and often operates in a way close to an "ideal" algorithm, where any criterion computation results in cutting-off some sub-tree, or needs to be computed in leaf nodes only.

It should be noted that we may consider also different prediction mechanisms. The simple overall averaging mechanism does not respect local context, in which particular feature contributions are evaluated. We experimented with an "exponential forgetting" mechanism to ensure bigger influence of the most recently computed criterion values. The results of such modified algorithm remained comparable to other results presented in this paper. The use of a stronger prediction mechanism resulted in minor (at most 5%) occasional time savings only. Defining other potentially more sophisticated prediction mechanisms is generally possible. However, the practitioner should keep in mind that maintaining the more sophisticated prediction mechanism would require additional time that may easily outweigh its positive influence.

7. CONCLUSION

Based on a detailed study of the Branch & Bound algorithm principle we developed a novel algorithm for optimal subset selection which is suitable especially for use with computationally expensive criterion functions. The Fast Branch & Bound algorithm shows to be significantly more efficient in most practical situations than any other traditional Branch & Bound algorithm (unless used with recursive criterion functions) due to its prediction mechanism for excluding unnecessary criterion computations.

We describe the algorithm in the context of feature selection, although its usability may be much broader without doubt, similarly to the known Branch & Bound algorithm.

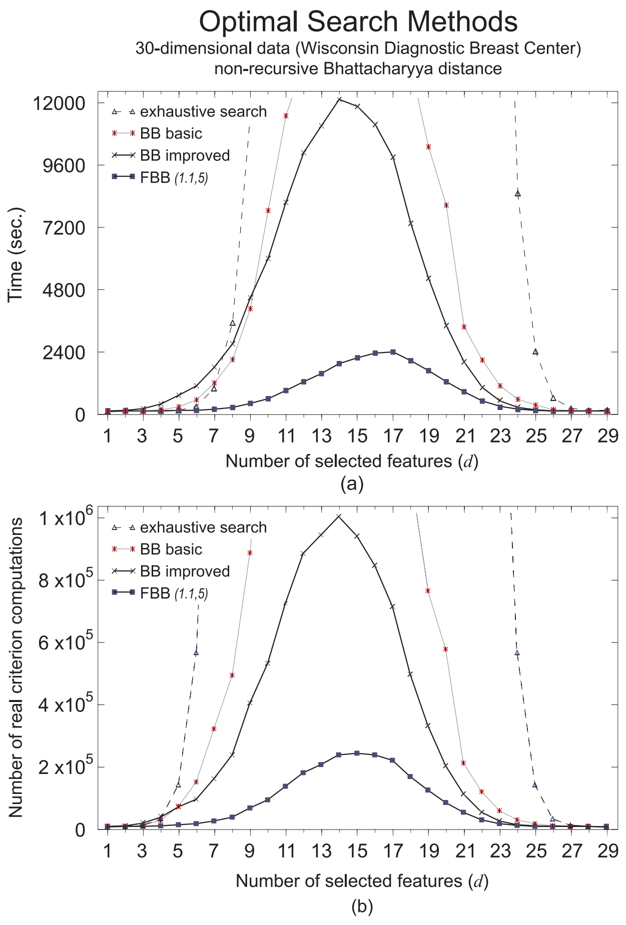

Fig. 4. *Optimal subset search methods results*

8. REFERENCES

[1] P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, C-26:917–922, September 1977.

[2] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, 1982.

[3] Y. Hamamoto, S. Uchimura, Y. Matsuura, T. Kanaoka, and S. Tomita. Evaluation of the branch and bound algorithm for feature selection. *Pattern Recognition Letters*, 11(7):453–456, July 1990.

[4] I. Foroutan, J. Sklansky. Feature selection for automatic classification of non-gaussian data. *IEEE Trans. on SMC, 17:187–198, 1987*

[5] K. Fukunaga. *Introduction to Statistical Pattern Recognition: 2nd edition*. Academic Press, Inc., 1990.

[6] J. Kittler. Feature set search algorithms. In *Pattern Recognition and Signal Processing, C. H. Chen, Ed.*, pages 41–60, The Netherlands: Sijthoff and Noordhoff, 1978.

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