## Correspondence

# A Branch and Bound Algorithm for Feature Subset Selection

PATRENAHALLI M. NARENDRA AND KEINOSUKE FUKUNAGA

*Abstract*—A feature subset selection algorithm based on branch and bound techniques is developed to select the best subset of $m$ features from an $n$-feature set. Existing procedures for feature subset selection, such as sequential selection and dynamic programming, do not guarantee optimality of the selected feature subset. Exhaustive search, on the other hand, is generally computationally unfeasible. The present algorithm is very efficient and it selects the best subset without exhaustive search. Computational aspects of the algorithm are discussed. Results of several experiments demonstrate the very substantial computational savings realized. For example, the best 12-feature set from a 24-feature set was selected with the computational effort of evaluating only 6000 subsets. Exhaustive search would require the evaluation of 2 704 156 subsets.

*Index Terms*—Branch and bound, combinatorial optimization, feature selection, recursive computation.

## I. Introduction

The problem of feature subset selection is to select a subset of ($m$) features from a larger set of ($n$) features or measurements to optimize the value of a criterion over all subsets of the size $m$. There are $\binom{n}{m} = n!/m!(n-m)!$ such subsets. Exhaustive evaluation of all the subsets is computationally prohibitive, as the number of subsets to be considered grows very rapidly with the number of features, for example, $\binom{12}{6}$ is 924, while $\binom{24}{12}$ is 2 704 156. Stepwise techniques [1] and dynamic programming solutions [2] are more efficient because they avoid exhaustive enumeration, but they offer no guarantee that the selected subset yields the best value of the criterion among all subsets of size $m$.

Here, we present a branch and bound formulation of the feature subset selection problem. The algorithm is very efficient because it avoids exhaustive enumeration by rejecting suboptimal subsets without direct evaluation *and* guarantees that the selected subset yields the globally best value of any criterion that satisfies monotonicity. Branch and bound methods are powerful combinatorial optimization tools and similar formulations have been applied to other problems in pattern recognition, such as clustering [5] and nearest neighbor computation [6]. Here an efficient subset enumeration scheme is developed to realize maximum advantage of the branch and bound principle. Recursive equations which facilitate rapid computation are derived for the class of quadratic criteria such as the discriminant function, divergence and Bhattacharyya distance for the normal case, etc. A suboptimal variant of the globally optimal branch and bound algorithm is also presented. Results of computer experiments demonstrating the efficiency of the algorithms are also included.

Manuscript received August 21, 1975; revised June 16, 1976. This work was supported in part by NSF Grant GJ-35722.

P. M. Narendra was with the Canada Centre for Remote Sensing, Ottawa, Ont., Canada. He is now with Honeywell Systems and Research Center, Minneapolis, MN 55413.

K. Fukunaga is with the School of Electrical Engineering, Purdue University, Lafayette, IN 47907.

## II. The Branch and Bound Algorithm

Let the number of features in the original set be $n$. We have to select a subset of $m$ features so that the value of a criterion is optimized over all subsets of size $m$.

Let $(Z_1, \cdots Z_{\bar{m}})$ be the $\bar{m} = n - m$ features to be *discarded* to obtain an $m$ feature subset. Each variable $Z_i$ can take on values in $\{1, 2, \cdots n\}$. But the order of the $Z_i$'s is immaterial, hence, we will consider only sequences of $Z_i$'s such that

$$Z_1 < Z_2 < \cdots < Z_{\bar{m}}. \tag{1}$$

A more general enumeration of the subsets will be given later.

The feature selection criterion is $J_{\bar{m}}(Z_1, \cdots, Z_{\bar{m}})$, a function of the $m$ features obtained by discarding $Z_1, \cdots, Z_{\bar{m}}$ from the $n$ feature set. The feature subset selection problem is to find the optimum subset $Z_1^*, \cdots, Z_{\bar{m}}^*$ such that

$$J_{\bar{m}}(Z_1^*, \cdots, Z_{\bar{m}}^*) = \max J_{\bar{m}}(Z_1, \cdots, Z_{\bar{m}}), \qquad Z_1, \cdots, Z_{\bar{m}}.$$

Fig. 1 is a solution tree enumerating all the possible subsets satisfying relation (1) for $n = 6$ and $m = 2$ ($\bar{m} = 4$). Each node is uniquely identified by the discarded feature, for example (1,4) for node $A$.

Let us assume that the criterion $J$ satisfies monotonicity, which is defined by

$$J_1(Z_1) \geq J_2(Z_1, Z_2) \geq \cdots \geq J_{\bar{m}}(Z_1, \cdots, Z_{\bar{m}}). \tag{2}$$

The monotonicity is not particularly restrictive, as it merely means that a subset of features should be not better than any larger set that contains the subset. Indeed, a large variety of feature selection criteria does satisfy the monotonicity relation. Discriminant functions and distance measures such as the Bhattacharyya distance and divergence are examples.

Let $B$ be a lower bound[1] on the optimum (maximum) value of the criterion $J_{\bar{m}}(Z_1^*, \cdots, Z_{\bar{m}}^*)$, i.e.,

$$B \leq J_{\bar{m}}(Z_1^*, \cdots, Z_{\bar{m}}^*). \tag{3}$$

If $J_k(Z_1, \cdots Z_k)(k < \bar{m})$ were less than $B$, then by (2),

$$\left.\begin{array}{c} J_{\bar{m}}(Z_1, \cdots, Z_k, Z_{k+1}, \cdots, Z_{\bar{m}}) \leq B \\ \text{for all possible } \{Z_{k+1}, \cdots, Z_{\bar{m}}\}. \end{array}\right\} \tag{4}$$

This means that whenever the criterion evaluated for any node is less than the bound $B$, all nodes that are successors of that node also have criterion values less than $B$, and therefore cannot be the optimum solution. This forms the basis for the branch and bound algorithm.

The branch and bound algorithm successively generates portions of the solution tree and computes the criterion. Whenever a suboptimal partial sequence or node is found to satisfy (4), the subtree under the node is implicitly rejected, and enumeration begins on partial sequences which have not yet been explored.

### The Enumeration Scheme

We note that with reference to Fig. 1, the nodes at a given level do not all have the same number of terminal nodes. Node (1,2) has three successors, while node (1,4) has only one. As a result, if the suboptimality test (4) is satisfied for node (1,2) (i.e., if $J_2(1,2) < B$), six sequences are rejected as being suboptimal, while for node (1,4), only the single sequence

[1] $B$ could be the best value of $J_{\bar{m}}(Z_1, \cdots, Z_{\bar{m}})$ found so far in the search, for example.

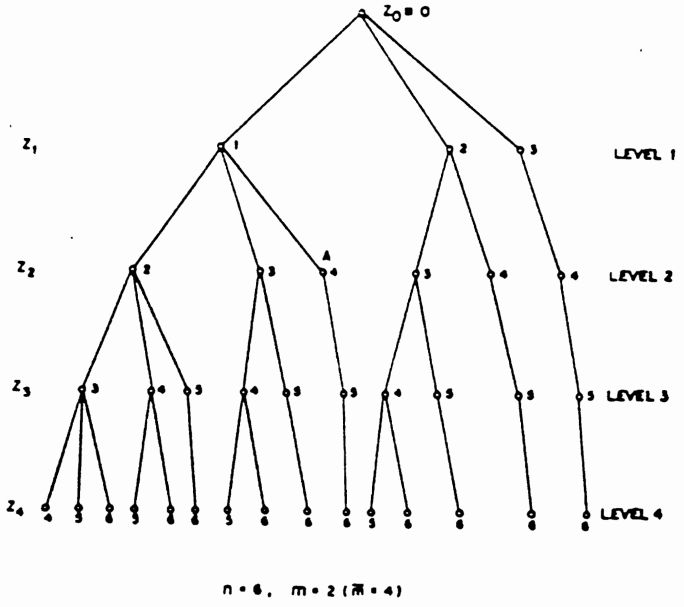


Fig. 1. Illustrating the solution tree generated by relation (1).

(1,4,5,6) would be rejected. The enumeration scheme that follows is based on the same tree structure as in Fig. 1. But, the successors of each node are ordered at each level so that the successors with smaller values of the partial criterion will be nodes which will have a larger number of successors in turn. This ensures that maximum advantage will be realized from the suboptimality test (4).

## *The Branch and Bound Algorithm*

The following notation will be used in the algorithm.

LIST($i$): An *ordered* list of the features enumerated at level $i$.

POINTER($i$): The pointer to the element of LIST($i$) being currently considered. For example, if the current element in LIST($i$) is the $k$th, then POINTER($i$) = $k$.

SUCCESSOR($i,k$): The number of successors that the $k$th element in LIST($i$) can have.

AVAIL: A list of available feature values that LIST($i$) can assume.

*Step 0—(Initialization):* Set $B = B_0$, AVAIL = $\{1,2,\cdots,n\}$, $i = 1$, LIST(0) = $\{0\}$, SUCCESSOR(0,1) = $m + 1$, POINTER(0) = 1.

*Step 1—[Initialize List(i)]:* Set NODE = POINTER($i - 1$). Compute $J_i(Z_1,\cdots,Z_{i-1},k)$ for all $k$ in AVAIL. Rank the features in AVAIL in the increasing order of $J_i(Z_1,\cdots,Z_{i-1},k)$ and store the smallest $p$ of these in LIST($i$) in the increasing order (with the first element in LIST($i$) being the feature in AVAIL yielding the smallest $J_i$), where $p$ = SUCCESSOR($i - 1$, NODE). Set SUCCESSOR($i,j$) = $p - j + 1$, for $j = 1,2,\cdots,p$. Remove LIST($i$) from AVAIL.

*Step 2—(Select new node):* If LIST($i$) is empty, go to Step 4. Otherwise, set $Z_i = k$ where $k$ is the *last* element in LIST($i$). Set POINTER($i$) = $j$ where $j$ is the current number of elements in LIST($i$). Delete $k$ from LIST($i$).

*Step 3—(Check bound):* If $J_i(Z_1,\cdots,Z_i) < B$, return $Z_i$ to AVAIL and go to Step 4. If level $i = m$, go to Step 5. Otherwise, set $i = i + 1$ and go to Step 1.

*Step 4—(Backtrack):* Set $i = i - 1$. If $i = 0$, terminate the algorithm. Otherwise, return $Z_i$ to AVAIL and go to Step 2.

*Step 5—(Final level, update bound):* Set $B = J_{\overline{m}}(Z_1,\cdots,Z_{\overline{m}})$ and save $(Z_1,\cdots,Z_{\overline{m}})$ as $(Z_1^*,\cdots,Z_{\overline{m}}^*)$. Return $Z_{\overline{m}}$ to AVAIL. Go to Step 4.

The flowchart in Fig. 2 illustrates the algorithm. The functioning of the algorithm is as follows: starting from the root of the tree, the successors of the current node are enumerated in the ordered list LIST($i$). The successor, for which the partial criterion $J_i(Z_1,\cdots,Z_i)$ is maximum (the rightmost successor), is picked as the new current node and the algorithm moves on to the next higher level. The lists LIST($i$) at each level $i$ keep track of the nodes that have been explored. The SUCCESSOR variables determine the number of successor nodes the current node will have at the next level. AVAIL keeps track of the feature values that can be enumerated at any level.

Whenever the partial criterion is found to be less than the bound, the algorithm backtracks to the previous level and selects a hitherto unexplored node for expansion. Whenever the algorithm reaches the last level $\overline{m}$, the lower bound $B$ is updated to be the current value of $J_{\overline{m}}(Z_1,\cdots,Z_{\overline{m}})$ and the current sequence $(Z_1,\cdots,Z_{\overline{m}})$ is saved as $(Z_1^*,\cdots,Z_{\overline{m}}^*)$. When all the nodes in LIST($i$) for a given $i$ are exhausted, the algorithm backtracks to the previous level. When the algorithm backtracks to level 0, it terminates.

At the conclusion of the algorithm $(Z_1^*,\cdots,Z_{\overline{m}}^*)$ will give the complement of the best feature set. Fig. 3 illustrates, for a random example, the tree enumerated including the nodes which were rejected by the suboptimality test (4). At level 1, features 4, 3, and 6 were enumerated because $J_1(4) < J_1(3) < J_1(6) < J_1(1) < J_1(2), J_1(5)$. The present algorithm is totally independent of the ordering of the features. No sequence is enumerated more than once (even as a permutation) and all possible sequences are considered either explicitly or implicitly, guaranteeing optimality of the subset sought. Moreover, the suboptimality test (4) is always used to the best advantage, rendering the algorithm very efficient.

## *Suboptimal Solutions*

We have presented an efficient branch and bound formulation which guarantees global optimality of the subset under the assumption of monotonicity. If we allow global optimality to be compromised, it is possible to improve the efficiency even further. We present a scheme that employs lookahead in the basic branch and bound algorithm, so that nonoptimal solutions were detected higher up in the tree, eliminating more suboptimal solutions.

We define upper bounds $b_i$ of the criterion for each stage $i$ as follows:

$b_i$ is the best (largest) value of $J_i(Z_1,\cdots,Z_i)$ found so far in the search. $b_i$ is updated every time a $J_i(Z_1,\cdots,Z_i)$ is computed to be greater than the current value of $b_i$. In Step 3 of the algorithm, we replace the test $J_i(Z_1,\cdots,Z_i) < B$ by $J_i(Z_1,\cdots,Z_i) < b_{i+l}$, where $l$ is an integer constant representing the lookahead factor. It is understood that $b_{i+l} = b_{\overline{m}}$, for $i + l \geq \overline{m}$.

With this change, instead of comparing the partial criterion $J_i(Z_1,\cdots,Z_i)$ with the bound $B$ (on $J_{\overline{m}}(Z_1,\cdots,Z_{\overline{m}})$), we now compare it with a bound $b_{i+l}$(on $J_{i+l}(Z_1,\cdots,Z_{i+l})$) for $l$ stages down the tree. Note that if $l = \overline{m} - 1$, then $b_{i+l} = b_{\overline{m}} \equiv B$ for all $i$, and the scheme is equivalent to the original branch and bound algorithm. For $l < \overline{m} - 1$ since $b_i \leq B$, more branches are eliminated in Step 3 of the algorithm and only the more promising nodes are evaluated at the higher levels of the tree. So,

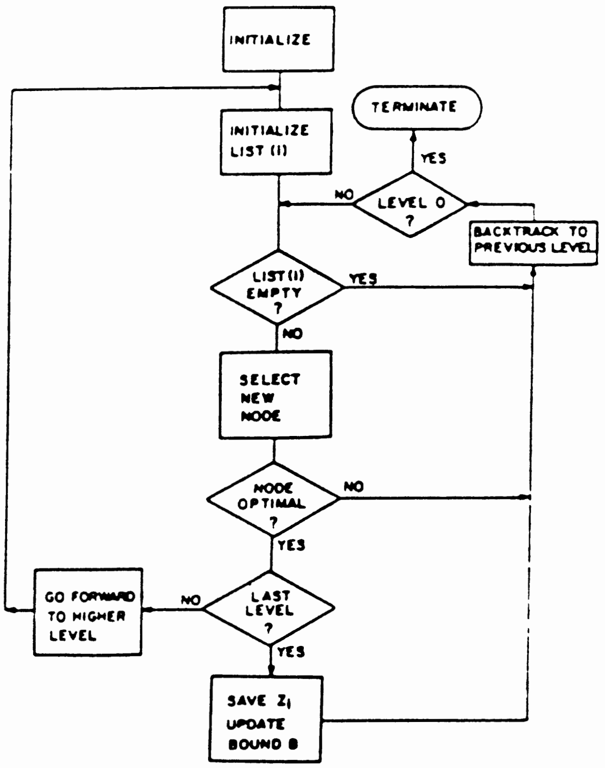


Fig. 2. Flowchart for the branch and bound algorithm.

for $l < \bar{m} - 1$ the algorithm enumerates fewer nodes and is therefore more efficient, although now there is no guarantee that the subset obtained is globally optimal. Incidentally, for $l = 0$ the algorithm approximates the familiar backward selection schemes. Therefore, we can expect that for $0 < l < \bar{m} - 1$ the algorithm would fall between the backward selection schemes and the globally optimal branch and bound algorithm, both in efficiency and optimality of the subset obtained. Indeed, this is borne out by the experimental results reported in Section IV.

## III. Recursively Computable Criteria

We noted in the previous section that the algorithms are implemented with the criterion evaluated for the partial sequences $(Z_1, \cdots, Z_k)$. The nature of the enumeration schemes requires that the value of the criterion be computed successively as features are deleted from the full set. For the class of quadratic criteria, we derive recursive equations to evaluate the criterion as a new feature is deleted from the present partial set. We will first consider the following quadratic form

$$J_k = X_k^T S_k^{-1} X_k \tag{6}$$

where $X_k$ is a $k$-vector and $S_k$ a $k \times k$ positive definite matrix when $k$ features are present.

This quadratic form is the basis of various quadratic criteria such as the discriminant function, the Fisher criterion, and Mahalanobis distance. Bhattacharyya distance and the divergence, for the normal case, also have terms of the same form as (6).

### A. Recursive Computation of $S_k^{-1}$

The inversion of $S_k$ is the major computational effort in evaluating (6) as features are successively deleted from the full set of features. When the $i$th feature is deleted it is necessary to compute the inverse of $S_k$ with the $i$th row and column deleted. Without loss of generality, let the feature being deleted correspond to the $k$th row and column of $S_k$.

$$S_k = \begin{array}{cc|l} \left[\begin{matrix} S_{k-1} & Y \\ Y^T & s_{kk} \end{matrix}\right] & & \begin{matrix} k-1 \\ 1 \end{matrix} \\ \underbrace{\quad}_{k-1} \quad \underbrace{\quad}_{1} & & \end{array} . \tag{7}$$

A fundamental identity in matrix algebra [7] gives $S_k^{-1}$ in terms of $S_{k-1}^{-1}$ as

$$S_k^{-1} = \begin{bmatrix} S_{k-1}^{-1} + \frac{1}{d} S_{k-1}^{-1} Y Y^T S_{k-1}^{-1} & -\frac{1}{d} S_{k-1}^{-1} Y \\ -\frac{1}{d} Y^T S_{k-1}^{-1} & \frac{1}{d} \end{bmatrix} \tag{8}$$

where

$$d = s_{kk} - Y^T S_{k-1}^{-1} Y. \tag{9}$$

If we write

$$S_k^{-1} = \begin{array}{cc|l} \left[\begin{matrix} A & C \\ C^T & b \end{matrix}\right] & & \begin{matrix} k-1 \\ 1 \end{matrix} \\ \underbrace{\quad}_{k-1} \quad \underbrace{\quad}_{1} & & \end{array} \tag{10}$$

then, by (8) and (10) it can be verified that

$$S_{k-1}^{-1} = A - \frac{1}{b} C C^T. \tag{11}$$

Hence, $S_{k-1}^{-1}$ can be computed from $S_k^{-1}$ with little computational effort. With reference to the algorithm, the inverse matrices $S_k^{-1}$ are stored for each level. The inverse at any level is computed from that of the previous level using the recursive equation (11). Whenever the algorithm backtracks and proceeds down another branch, the inverse for the new $S_k$ can be recomputed from the inverse at the level at which the branching occurred. For example, suppose in Fig. 3, after (3,1,6,2) is explored, the algorithm backtracks to the node (3,5). The value of $S_k^{-1}$ for the node (3,5) can be computed from the current value of $S_k^{-1}$ at level 1. The $S_k^{-1}$ for level two is updated to be this value as feature 5 is now chosen to be $Z_2$.

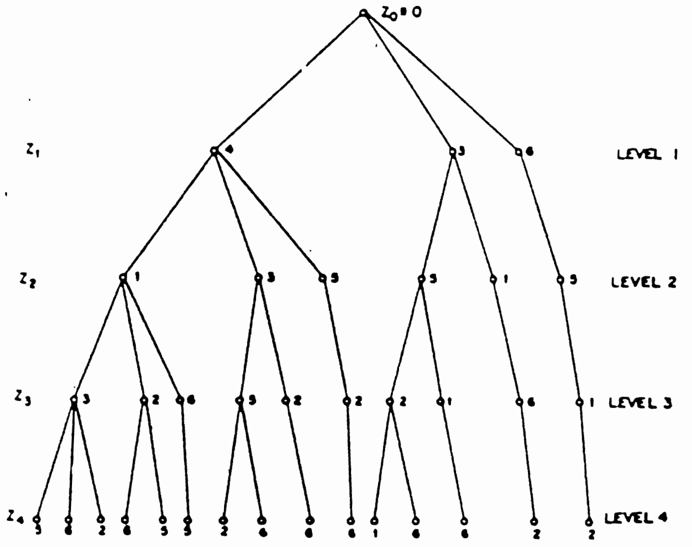


Fig. 3. A solution tree generated by the enumeration scheme in the algorithm.

## B. Recursive Computation of $X_k^T S_k^{-1} X_k$

It is also possible to recursively compute the quadratic $X_k^T S_k^{-1} X_k$ for $k - 1$ features given its value with $k$ features and the $S_k^{-1}$ matrix from the previous level. This is useful in Step 1 of the algorithm where it is necessary to compute the criterion after deleting one feature at a time from a partial set. This avoids computation of $S_{k-1}^{-1}$ during this step. Once a node is selected however (Step 3), the $S_k^{-1}$ for the new level has to be updated using (11).

Let the criterion with $k$ features be denoted by $J_k = X_k^T S_k^{-1} X_k$, and let us assume that the $k$th feature is being deleted as before.

$J_{k-1} = X_{k-1}^T S_{k-1}^{-1} X_{k-1}$ is the criterion with $k - 1$ features where

$$X_k = \begin{bmatrix} X_{k-1} \\ x_k \end{bmatrix} \begin{matrix} \} k-1 \\ \} 1 \end{matrix}, \tag{12}$$

and $S_{k-1}$ is defined as in (7).

As a consequence of (11), it is easy to verify that

$$\begin{aligned} X_{k-1}^T S_{k-1}^{-1} X_{k-1} &= X_{k-1}^T \left[ A - \frac{1}{b} CC^T \right] X_{k-1} \\ &= X_{k-1}^T A X_{k-1} - \frac{1}{b} (C^T X_{k-1})^2 \\ &= X_k^T S_k^{-1} X_k - \left[ b x_k^2 + 2 x_k C^T X_{k-1} + \frac{1}{b} (C^T X_{k-1})^2 \right] \\ &= X_k^T S_k^{-1} X_k - \frac{1}{b} [(C^t : b) X_k]^2. \end{aligned} \tag{13}$$

Note that $(C^T:b)$ is a row of $S_k^{-1}$ corresponding to the feature being deleted. Hence $(C^T:b)\, X_k$ is merely the inner product of that row with $X_k$. Thus, the $J_i(Z_1, \cdots, Z_{i-1}, j)$ in the algorithms can be directly evaluated from $J_{i-1}(Z_1, \cdots, Z_{i-1})$ by (13) without actually having to compute $S_{k-1}^{-1}$ for all the variables $j$. Incidentally, (13) also furnishes proof that $J$ is monotonic.

## V. Computer Experiments

The algorithms were tested on multispectral data acquired from airborne remote sensing scanners at the Laboratory for Applications for Remote Sensing (LARS) at Purdue University.

The data were comprised of 423 sample vectors each, from two classes classified as soybean and corn. There were 12 data channels corresponding to 12 bands of the spectrum in which the sensing was performed. Each channel is, of course, a feature and the problem was to select a subset of the channels which was best, according to a given criterion. The criterion chosen was the discriminant function defined as follows:

$$J_d = (M_1 - M_2)^T \left( \frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (M_1 - M_2)$$

where $M_i$ are the sample means for the two classes and $\Sigma_i$ are the within class scatter matrices.

First the algorithm was applied to choose the best four out of the 12 channels. There are $\binom{12}{4} = 495$, four-channel subsets. The total number of partial sequences of all sizes enumerated (nodes visited) was 37. Since the criterion is to be evaluated once for each node visited, this would suggest a computational saving of the order of ten over exhaustive search. However, we note that the recursive equations were used for evaluating the bounds. Hence, we estimate a further saving of the order of four (because four feature subsets would be considered an exhaustive search without recursive computation).

what distance measure to use for multiclass ca

The experiment was repeated with data from additional classes such as rye, wheat, rape, etc., taken in pairs. The average number of subsets enumerated was 35, with a minimum of 28 and a maximum of 64.

To evaluate the performance of the algorithms for large problems, an additional set of 12 features was generated by taking the square of the first 12 features. The covariance matrix and the means were computed for the 24 feature set. It is to be expected that the resulting 24 feature set is very correlated, and there may be several subsets that yield very close values of the criterion.

The optimal algorithm was applied to the 24 feature problem to choose the best 12 features. There are $\binom{24}{12} = 2\,704\,156$ subsets. The algorithm explored 5646 subsets and took 269 s of the PDP-10 central processing unit (CPU) time to terminate. This demonstrates a very considerable saving over the exhaustive search. Table I gives the number of nodes expanded (subsets enumerated) at each level and the number of nodes for which the inequality (4) was satisfied at each level. Because of the reordering of the features, every node that was rejected at each level of the algorithm results in a large number of suboptimal sequences being discovered. Hence, fewer nodes are enumerated overall. The additional complexity of the feature ordering appears justified in the light of its efficiency. Also, this renders the algorithm independent of the initial ordering of the features.

### Suboptimal Solutions

The lookahead scheme in Section II was incorporated into the basic algorithm. Experiments were repeated with the 12 channel and 24 channel corn-soybean data. Tables II and III give the corresponding results for different values of the lookahead factor $l$. For $l = 0$, the algorithm approximates backward selection as the worst feature is discarded at each stage, and no backtracking is involved. $l = 7$ corresponds to the globally optimal scheme (the basic branch and bound algorithm). With the 12-channel example, we see that for $l \geq 1$, the optimum subset is obtained. But the number of nodes enumerated for $l = 1$ (33) is not substantially smaller than for the optimal case (37). The larger example in Table III is more indicative of the effectiveness of the scheme. The optimal subset is obtained with $l = 3$, and the number of subsets evaluated is only 1008

TABLE I
Summary of Behavior of the Algorithm for the 24-Variable Problem

| | LEVEL | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| No. of nodes enumerated | 13 | 91 | 323 | 631 | 1091 | 1674 | 2024 | 1742 | 1242 | 910 | 347 | 188 |
| No. of nodes rejected | 0 | 13 | 90 | 127 | 199 | 452 | 756 | 800 | 466 | 633 | 180 | 168 |

TABLE II
Results of Suboptimal Selection Scheme for the 12-Channel Example

| | LEVEL OF OPTIMIZATION | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 7 (OPTIMAL) |
| Number of Nodes Explored | 8 | 33 | 36 | 36 | 37 | 37 |
| CPU Time (PDP-10) in Seconds | 0.81 | 0.93 | 0.98 | 0.98 | .98 | .98 |
| Criterion Value with Subset Selected | 7.36 | 7.98 | 7.98 | 7.98 | 7.98 | 7.98 |
| Feature Subset Selected | (1 4 6 10) | (4 6 10 11) | (4 6 10 11) | (4 6 10 11) | (4 6 10 11) | (4 6 10 11) |

TABLE III
Results of the Suboptimal Selection Scheme for the 24-Channel Example

| | LEVEL OF OPTIMIZATION | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 Optimal |
| No. of Nodes Explored | 12 | 102 | 389 | 1008 | 1790 | 3318 | 5017 | 5397 | 5549 | 5646 | 5646 | 5646 |
| CPU Time* (PDP-10) Seconds | 2.23 | 6.37 | 18.57 | 45.32 | 79.71 | 161.22 | 223.37 | 244.8 | 268.27 | 268.49 | 268.5 | 268.5 |
| Criterion Value with Subset Selected | 6.87 | 7.01 | 9.27 | 9.47 | 9.47 | 9.47 | 9.47 | 9.47 | 9.47 | 9.47 | 9.47 | 9.47 |
| No. of Features in the Selected Subset Different from the Optimal Subset | 3 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

versus 5646 for the optimal algorithm. Hence, for the larger variable problems, the suboptimal lookahead scheme with smaller values of $l$ is more efficient than the optimal algorithm and yields more optimal subsets than backward sequential selection.

## VI. Summary

A branch and bound algorithm to select the globally best feature subset, based on an efficient subset enumeration scheme, was presented. A suboptimal variant of the basic branch and bound algorithm was suggested. For a general class of feature selection criteria, recursive equations were developed to facilitate the implementation of the algorithms. Results of computer experiments were presented to substantiate the claim that the algorithms are very efficient.

## References

[1] A. N. Mucciardi and E. E. Gose, "A comparison of seven techniques for choosing subsets of pattern recognition properties," *IEEE Trans. Comput.*, vol. C-20, pp. 1023–1031, Sept. 1971.
[2] C. Y. Chang, "Dynamic programming as applied to feature subset selection in a pattern recognition system," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-3, pp. 166–171, Mar. 1973.
[3] S. W. Golomb and L. D. Baumert, "Backtrack programming," *J. Ass. Comput. Mach.*, vol. 12, pp. 516–523, Jan. 1965.
[4] E. L. Lawler and D. E. Wood, "Branch-and-bound methods—A survey," *Oper. Res.*, vol. 149, no. 4, 1966.
[5] W. L. G. Koontz, P. M. Narendra, and K. Fukunaga, "A branch and bound clustering algorithm," *IEEE Trans. Comput.*, pp. 908–915, Sept. 1975.
[6] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing $k$-nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, pp. 750–753, July 1975.
[7] C. R. Rao, *Linear Statistical Inference and Its Applications.* New York: Wiley, 1965, p. 29.
[8] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* New York: Academic, 1972, ch. 9.

# Correction to "An Application of Relaxation Labeling to Line and Curve Enhancement"

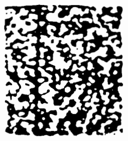
STEVEN W. ZUCKER, ROBERT A. HUMMEL, AND AZRIEL ROSENFELD

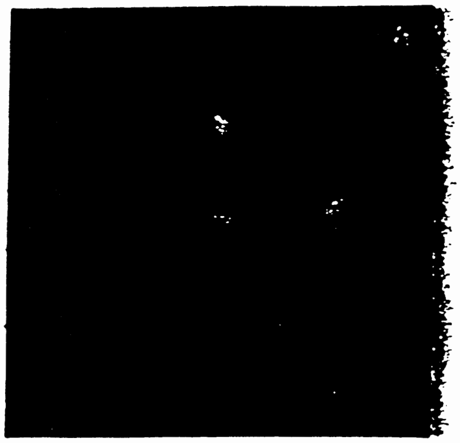
In the above paper[1] Figs. 4–7, and 10–12 were inadvertently misrepresented. They are correctly displayed here.

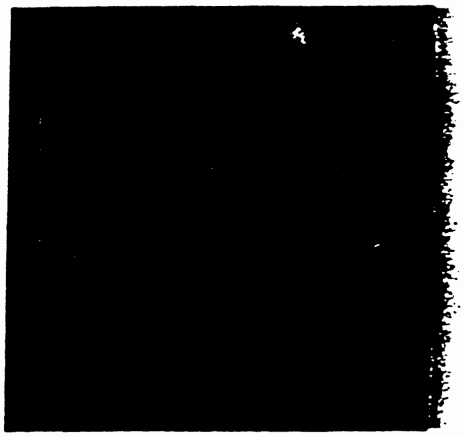
Manuscript received June 8, 1977.

S. W. Zucker was with the Computer Science Center, University of Maryland, College Park, MD 20742. He is now with the Department of Electrical Engineering, McGill University, Montreal, P.Q., Canada.
R. A. Hummel and A. Rosenfeld are with the Computer Science Center, University of Maryland, College Park, MD 20742.

[1] S. W. Zucker, R. A. Hummel, and A. Rosenfeld, *IEEE Trans. Comput.*, vol. C-26, pp. 394–403, Apr. 1977.


(a)


(b)


(c)

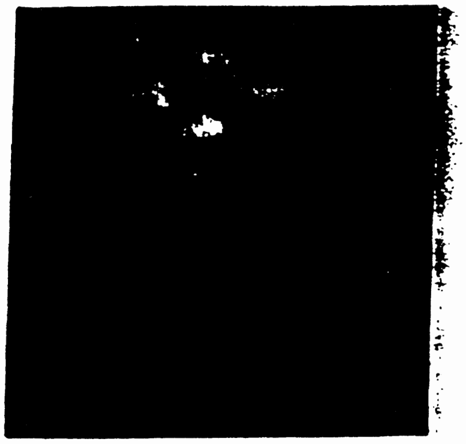

(d)

Fig. 4. (a) Computer-generated line in noise. (b) Initial probability assignments for (a), obtained from nonlinear line detector responses. (c)–(g) Iterations 1–4, and 8 of the relaxation process applied to (b).