Pattern Classifier Design by Linear Programming

FRED W. SMITH, MEMBER, IEEE

Abstract—A common nonparametric method for designing linear
discriminant functions for pattern classification is the iterative, or
"adaptive," weight adjustment procedure, which designs the dis-
criminant function to do well on a set of typical patterns. This paper
presents a linear programming formulation of discriminant function
design which minimizes the same objective function as the "fixed-
increment" adaptive method. With this formulation, as with the
adaptive methods, weights which tend to minimize the number of
classification errors are computed for both separable and nonsepa-
rable pattern sets, and not just for separable pattern sets as has been
the emphasis in previous linear programming formulations.

The computation times required by the linear program and con-
ventional iterative techniques have been compared theoretically and
experimentally. The linear program is shown to be definitely faster
when the ratio of the number of patterns to the number of pattern
variables is small. When the ratio is large the linear program may
take longer. Methods for eliminating nonessential patterns from the
design set so as to keep this ratio small are discussed.

Index Terms—Classifier design, comparison, computation time,
"fixed-increment" adaptive, linear programming, nonparametric,
pattern.

INTRODUCTION

LINEAR programming has been used to design linear
discriminant functions for pattern classification; for example, see Minnick and Mangasarian.\(^1\) However, these previous linear programming formulations
have been concerned with testing the patterns for separability, and if they were separable, then optimizing the weights in some manner. The emphasis has not been
on finding the most suitable weights for nonseparable pattern sets. In this paper a linear programming prob-
lem is formulated to minimize the same objective func-
tion as is minimized by the "fixed-increment"\(^2\) adap-
tive method. With this formulation meaningful weights, i.e., weights which tend to minimize the number of class-
fication errors, are obtained regardless of the separa-
bility of the patterns.

The problem of designing discriminant functions for
nonseparable pattern sets is a practical one because in
many practical problems the overlapping probability
densities of the different pattern categories will make
nonseparable pattern sets inevitable. Current practice
for pattern sets which are not completely separable is to
use the adaptive methods anyway, because they tend
to yield adequate classifiers even though they cannot
eliminate all classification errors.

Using linear programming in the design is easier than
using adaptive methods. Scaling of analog pattern vari-
ables is not necessary because it has little effect on the
computation time. The problem of choosing the proper
step size for the adaptive methods does not arise, nor
does the problem of choosing a termination criterion
for nonseparable pattern sets. In addition, the linear
program is often faster because its computation time is
essentially independent of problem difficulty.

FORMULATION OF CLASSIFIER DESIGN PROBLEM

Adaptive classifier design methods attempt to find a
linear weighting of the pattern variables which is posi-
tive for patterns in one category and negative for pat-
terns in the other. Stated mathematically, if \(X_k\) is the
\(k\)th pattern of \(N\) variables

\[
X_k = (x_{1k}, x_{2k}, \ldots, x_{Nk})^T, \quad k = 1, 2, \ldots, K,
\]

and \(W\) is the weight vector of \(N\) weights

\[
W = (w_1, w_2, \ldots, w_N)^T,
\]

then it is desired to find \(\hat{W}\) such that

\[
X_k^TW \geq d, \quad d > 0, \quad k = 1, \ldots, K
\]  \(1\)

for all patterns in the positive category and for the
negative of all patterns in the negative category.\(^1\) The \(d\)
represents a "dead zone" which, for reliability, forces
\(X_k^TW\) away from the inconclusive condition \(X_k^TW = 0\).

The scalar \(d\) establishes a scale factor for the
weights\(^1\) and was taken as unity throughout this paper.
In most meaningful problems \(K \gg N\), often by a factor
of 10 or more, in order to assure that the weights will be
statistically significant and thus will properly classify
other patterns which have the same statistics as the
\(X_k\).

There are a variety of closely related methods for
the adaptive design of linear discriminant functions. All of
the methods can be interpreted using the concept of
pattern error functions given in Koford and Groner.\(^1\)\(^6\)
These methods find the desired \(\hat{W}\) by minimizing the
mean error function

\[
\hat{h} = \sum_{k=1}^{K} \pi_k h_k
\]  \(2\)

where \(h_k\) is the pattern error function associated with
the pattern \(X_k\), \(\pi_k\) is a weighting coefficient, and \(K\) is the
total number of patterns. For the fixed-increment adap-

\(^1\) One of the \(x_{ik}\) can be a constant "threshold input." The cor-
responding "threshold weight" \(w_0\) thus sets the threshold for the
classifier. Because no distinction is made for the threshold input in
this paper the total number of weights is \(N\), not \(N+1\) as in refer-
ences [3),(6),(10).
tive method the pattern error function is defined as
\[ h_k = - (X_k^T W - d) \quad \text{when} \quad X_k^T W < d, \]
\[ h_k = 0 \quad \text{when} \quad X_k^T W \geq d, \]
where all patterns in the negative category have been multiplied by \(-1.\) With this pattern error function the \( W \) which makes \( h = 0 \) will satisfy the inequalities of (1), and thus will correctly classify all of the \( X_k \). Two variations of the fixed-increment adaptive method will be described.

The Steepest-descent design method\(^{[1]}\) starts from some arbitrary initial \( W \), such as \( W = 0 \), and changes \( W \) in increments,
\[ \Delta W = - \frac{\beta}{N} \frac{\partial h(W)}{\partial W} = - \frac{\beta}{N} \sum_k \pi_k X_k, \tag{3} \]
where the summation is made over \( k \) for which \( h_k \neq 0 \), and \( \beta/N \) is a constant scalar which controls the size of \( \Delta W \). Each change in \( W \) requires evaluating \( X_k^T W \) for all \( K \) patterns, and will hereafter be referred to as one iteration of the steepest-descent method.

The one-at-a-time design method\(^{[3]}\) is an approximation to the steepest-descent method and is similar except that the change in \( W \),
\[ \Delta W = \frac{\beta}{N} \pi_k X_k, \quad \text{if} \quad h_k \neq 0 \]
\[ \Delta W = 0, \quad \text{if} \quad h_k = 0, \tag{4} \]
is made after each \( X_k^T W \) is evaluated. To avoid confusion, one iteration in the one-at-a-time method will be taken to mean evaluating \( X_k^T W \), and making the required changes in \( W \), once for each of the \( K \) patterns.

The steepest-descent and the one-at-a-time iterative design methods are often started at \( W = 0 \). However, as will be demonstrated in this paper it is advantageous to start them with \( W \) at a multiple of the difference of the means of the two categories, i.e., at
\[ W = \alpha \sum_{k=1}^K \pi_k X_k, \tag{5} \]
where all \( X_k \) in the negative category have been multiplied by \(-1.\), and where \( \alpha \) is a scaling constant. It is desirable to choose \( \alpha \) in a manner which tends to minimize \( h \) before the adaption is started. In this study \( \alpha \) was taken as the largest scalar \( \alpha_M \) which gave a minimum \( h \).

Formulation as a Linear Programming Problem

The objective function used in the fixed-increment adaptive design method, \( h \) of (2), is a linear segment function of \( W \) and thus can also be minimized by linear programming. To formulate the minimization of (2) as a linear programming problem, the inequalities of (1) must be rewritten as the equalities
\[ XW + H = D + S, \tag{6} \]
where
\[ X = \begin{bmatrix} X_1^T \\ \vdots \\ X_K^T \end{bmatrix}, \]
\[ H = (h_1, h_2, \ldots, h_K)^T, \]
\[ D = (d, d, \ldots, d)^T, \]
\[ S = (s_1, s_2, \ldots, s_K)^T. \]

The vector \( H \) contains the pattern error functions \( h_k \) of (2); the vector \( S \) consists of slack variables which allow the inequalities to be written as equalities.

The simplex method requires that the components of \( H, D, \) and \( S \) all be non-negative. In addition, only \( S - H \) will be computed; therefore, to remove ambiguities, at least one of \( s_k \) or \( h_k \) will always be defined to be zero for each \( k \). Because the components of \( W \) can be of either sign, and because the simplex method requires all variables to be non-negative, it is necessary to express \( W \) as
\[ W = W^+ - W^- \]
where \( W^+ \) and \( W^- \) have non-negative components. Again to remove ambiguities, at least one of each of the corresponding components of \( W^+ \) and \( W^- \) will always be defined to be zero.

The objective function (2) can be expressed in matrix notation as
\[ \tilde{h} = \Pi^T H \tag{7} \]
where \( \Pi = (\pi_1, \pi_2, \ldots, \pi_K)^T \). To minimize \( \tilde{h} \), using linear programming, it is convenient to transform the minimization of (7) into the equivalent maximization of
\[ z = \Pi^T D - \Pi^T H. \tag{8} \]
Substitution of \( D - H \) from (6) into (8) gives
\[ z = \Pi^T X W - \Pi^T S. \]

The initial matrix equation that must be considered in linear programming thus is
\[ \begin{bmatrix} -\Pi^T X & \Pi^T X & 0 & \Pi^T & 1 \end{bmatrix} \begin{bmatrix} W^+ \\ W^- \\ H \\ D \\ S \end{bmatrix} = \begin{bmatrix} 0 \\ \Pi^T \end{bmatrix} \tag{9} \]
The simplex method can be used to manipulate the rows of (9) in a manner which finds the \( W, H, \) and \( S \) which maximize \( z \).\(^{[1]}\) Basically, this method finds a succession of solutions to (9), each of which has a \( z \) which is greater

\(^{[1]}\) In the usual linear programming terminology the \( h_k \) are called "artificial" variables; however, in this formulation they are not artificial but of primary concern.
than or equal to the \( z \) of the previous solution. The initial solution is \( W^* = W^+ = \mathbf{0}, H = D, S = 0, \) and \( z = 0. \)

If the pattern set is separable, i.e., a \( \mathcal{W} \) satisfying the inequalities of (1) exists, \( \mathbf{h} = 0 \) is possible and the simplex method terminates with \( z = \Pi^T D; \) if the set is not separable, \( \mathbf{h} = 0 \) is not possible and the simplex method terminates with \( z \) at some lesser value. In both cases a weight vector \( W \) is always computed. By proper programming it is necessary to store in computer memory only a matrix of the size of \( \chi \) (a \( K \times N \) matrix), two vectors of length \( K \), and two vectors of length \( N \).

Applying the simplex method to the linear programming problem of (9) results in longer computation times than required by the previous formulations which only tested for separability, because the previous formulations could be programmed to solve the dual problem, i.e., to work with \( \chi^T \) instead of \( \chi \). The time per pivot (iteration of the linear program) is the same for both formulations while the number of pivots for the dual is on the order of the number of rows of \( \chi^T, N \), which is generally much less than the order of \( K \) (the number of rows of \( \chi \)) required for the formulation of (9). However, with the dual formulation the \( h_k \) (artificial variables) do not enter the problem and no weights are calculated for nonseparable problems. Thus, the formulation including the \( h_k \) gains for its additional pivots the capability to compute weights for nonseparable as well as separable patterns. Neither formulation has an advantage in data storage requirements because both require the storage of a \( K \times N \) matrix.

**Linear Program Evaluation**

The computational accuracy and the computation time were evaluated for the linear program by using it to design the classifiers for several carefully chosen classification problems. The patterns for these problems and the required weights are generated by recursive relations given in Section IV of Muroga.\(^{[14]}\) These problems were used by Muroga to supply a lower bound on the maximum magnitude weight of the "minimum weights" required for any problem in \( N \) binary variables. The minimum weights \( W' \) for a problem are defined as those weights which for a dead zone of \( d = 1 \) satisfy the inequalities of (1) and have a minimum sum of weight magnitudes, i.e.,

\[
\min \sum_{i=1}^{N} |w_i|.
\]

The sum of weight magnitudes of the \( W' \) for a problem is of interest because it gives an indication of the difficulty of the problem. Both the number of iterations required by the adaptive algorithms and the minimum sensitivity of the classifier to errors in the weights vary with the sum of weight magnitudes of the minimum weights.\(^{[20]}\) Because these problems are used as a lower bound on the maximum weight magnitude, which tends to increase with the sum of weight magnitudes, these test problems are probably close to the most difficult problems in \( N \) binary variables.

**Computation Time of the Linear Program**

Two problems are used to illustrate the computation time required by the linear program. The first consisted of the eight patterns of Muroga's bound for eight-variable patterns. The minimum weights for this problem, as given by Table II of Muroga,\(^{[14]}\) are

\[
W' = (2, 3, 4, 5, 8, 9, 15, 17)^T
\]

which have a sum of 63.

The second problem consisted of the 128 eight-variable patterns representing all 256 eight-variable patterns. (Only 128 of the 256 patterns are necessary since with 256 patterns multiplying a pattern in the negative category by \(-1\) makes it equal to a pattern in the positive category.) The desired category for each of these patterns was obtained by classifying it with the weight vector for the first pattern set. Thus, the minimum weights for this problem are the same as for the first pattern set.

The one-at-a-time design method and possibly the linear program are a slight function of the pattern order. The eight-variable patterns were arbitrarily used in the same order as they were generated by Muroga's algorithm. The 128 eight-variable patterns were arbitrarily ordered in what would be considered the decreasing order of binary numbers from 11111111 to 10000000 if the \(-1\)'s were interpreted as zeros. The \( \pi_k \) for each pattern was \( 1/K \).

Table I gives the computation times and numbers of iterations (called pivots) for these problems when the linear program is written in **FORTAN IV** for a CDC 3200 with floating-point hardware. The time per iteration per data element of \( \chi \) is also given.

**Table I**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Computation Time (seconds)</th>
<th>Number of Pivots</th>
<th>Time per Iteration per Data Element (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Patterns</td>
<td>0.3</td>
<td>13</td>
<td>0.18</td>
</tr>
<tr>
<td>128 Patterns</td>
<td>32.5</td>
<td>158</td>
<td>0.20</td>
</tr>
</tbody>
</table>

The results in Table I indicate that the time per iteration per data element is essentially independent of the classification problem. Therefore, it is possible to extrapolate the computation time required by the linear program to other problems. The number of pivots required by the linear program for these problems (and the many other problems, both separable and nonseparable, solved with the linear program\(^{[10]}\)) is from one to
two times \( K \). (It can be shown theoretically that at least \( K \) iterations are required for a separable problem.) Because the computation time varies with the number of data elements, \( N \times K \), as well as with the number of iterations, it follows that the computation time of the linear program will vary with \( N \times K^2 \).

**Computational Accuracy of the Linear Program**

The computation time of the linear program depends primarily on the value of \( N \times K^2 \) and is almost completely independent of the difficulty of the problem. This conclusion assumes that the linear program is capable of computing a solution without accumulating unacceptable computational errors. Accumulated computational error is a problem for the linear program, but not for the adaptive methods because the simplex method is an “open-loop” computation while the other methods have “feedback” computations in which the effects of any computational errors are exponentially damped.

To test the computational accuracy of the linear program it was tried on the patterns of Muroga’s bound for 16-, 20-, and 24-variable patterns. The sums of weight magnitudes for these problems are, respectively, 14,765; 235,317; and 3,761,041.

The weights computed for the 16-variable problem were the minimum weights with negligible error. The weights for the 20-variable problem accumulated an error of approximately one part in \( 10^8 \) which amounted to an error of +0.66 over the maximum weight, 58,869. However, the classifier still gave outputs which satisfied the inequalities. The program was unable to compute any meaningful weights for the 24-variable problem. The linear program is thus capable of solving problems in which the sum of weight magnitudes is 235,317. In addition, in the study reported in Smith\[9\] it was successfully used on problems of unknown difficulty which required more than 100 iterations.

**Computation Time of the Adaptive Methods**

To evaluate the computation time of the linear program relative to the adaptive methods, the steepest-descent and one-at-a-time methods were programmed for the CDC 3200 and used on the two eight-variable problems. For both of these methods the \( \beta \sigma_\tau \) as defined by (3) and (4) was set at 2.0 (with \( d = 1 \)). This value is larger than the compromise 0.5 suggested by Mays.\[3\]

It was chosen because for these problems the larger \( \beta \) gave shorter computation times and did not drastically increase the sum of weight magnitudes. According to Mays\[11\] the computation time required for the fixed-increment adaptive method is also typical of that required by other adaptive methods, such as the “modified relaxation” method.

The computation time for Muroga’s eight patterns of eight variables is indicated in Fig. 1 which gives \( h \) versus computation time for the three different methods. This computation time includes only the time for computation; the time taken for computer input and output is not included. For the steepest-descent and one-at-a-time methods \( h \) varies erratically with time, and therefore only the approximate envelope is given. The envelope for the one-at-a-time method is smaller because \( h \) was computed only at every ten iterations and many of its extreme excursions were missed. Each of the iterative methods was started at \( W = 0 \), which gives \( h = 1 \).

The time functions of Fig. 1 show that for this problem the linear program is considerably faster, taking approximately 0.3 second to reach \( h = 0 \); the one-at-a-time and steepest-descent methods took 7.8 and 9.6 seconds (763 and 961 iterations), respectively, to reach \( h = 0 \), i.e., to terminate, and 7.6 and 4.4 seconds, respectively, to reach zero classification errors for the first time.\[4\] (The algorithms do not necessarily terminate the first time the classifier gives zero classification errors because \( h \) may not be zero, and thus may be minimized further.) Starting the one-at-a-time method with \( W \) equal to the difference of the means given by (5) with \( \alpha = \alpha_M \) made essentially no difference in the computation time. No advantage is gained for the steepest-descent method by starting with \( W \) at the difference of means. (It always sets \( W \) equal to some multiple of the difference of means.)

\[4\] The times given in these examples for the one-at-a-time method to reach zero classification errors for the first time are approximate because the number of classification errors was checked only at every ten iterations (approximately every 0.10 second for the eight patterns and every 1.14 seconds for the 128 patterns).
Fig. 2. \( \hat{h} \) versus computation time for 128-pattern problem.

Fig. 2 gives \( \hat{h} \) versus computation time for the 128 patterns of eight variables. For this problem the linear program and the one-at-a-time method starting at \( W = 0 \) took considerably longer than the other methods. The former took 32.5 seconds to reach \( h = 0 \) while the latter took 35.4 seconds (543 iterations) to reach \( h = 0 \) and zero errors. The steepest-descent method took 7.9 seconds (72 iterations) to reach \( h = 0 \) and 1.5 seconds to reach zero errors for the first time. The fastest computational method for this problem was the one-at-a-time method started with \( W \) equal to the difference of the means as given by (5) with \( \alpha = \alpha_M \). It took 4.9 seconds to reach \( h = 0 \) and zero classification errors. The steepest-descent method in this example also computed on the first iteration the multiple of the difference of means given by (5) with \( \alpha = \alpha_M \), and thus gained no advantage by starting at the difference of means.

Extrapolation of the required computation time to other problems is less exact for the adaptive methods than for the linear program. For the one-at-a-time method an upper bound on the number of changes made in \( W \) is given by Mays\(^\text{[1]}\) as

\[
N \frac{W^*W^*}{d^2} \left( \frac{2d + \beta}{\beta} \right),
\]

where \( W^* \) is the minimum length \( W \) which satisfies the inequalities, and the iteration starts with \( W = 0 \).\(^6\) Mays' experimental work\(^\text{[2]}\) indicates that the required number of changes is approximately one half of the bound. When \( K \) is little larger than \( N \), the number of \( X_k^T W \) tested is approximately the same as the number of changes. However, when \( K \gg N \) the number of tests will be considerably greater than the number of changes and Mays' bound does not give a good indication of the number of tests. Nevertheless, to extrapolate the computation time required by the adaptive methods to other problems it will be optimistically assumed that (10) gives a good approximation to the number of tests required by the most suitable of the adaptive methods. This is equivalent to assuming that the number of tests is independent of \( K \). Because the time for each test varies with \( N \) and the number of tests varies with \( N(W^*W^*) \) (assuming \( d = 1 \)), it follows that the computation time will vary with \( N^3(W^*W^*) \).

The validity of this assumption is supported by the number of iterations required in the eight-variable problems. For both 8 and 128 patterns with \( d = 1 \), using \( W' = (2, 3, 4, 5, 8, 9, 15, 17) \) as an approximation to \( W^* \), the bound can be computed to be 11 408 changes in \( W \). In the eight-pattern problem the number of patterns tested was \( 8 \times 763 = 6104 \). Although the one-at-a-time method starting with \( W = 0 \) required many more inspections than the bound (128 \times 543 = 69 504 tests), the other adaptive methods were much faster. The number of tests for the one-at-a-time method starting with the \( W \) given by (5) with \( \alpha = \alpha_M \) and for the steepest-descent method are roughly one half to one times the bound, i.e., \( 128 \times 72 = 9216 \) for the steepest-descent, and \( 128 \times 43 = 5504 \) for the one-at-a-time method starting at the stated difference of the means.

**Comparison of Computation Times**

The computation times of the adaptive methods and the linear program can be compared by comparing \( N^2(W^*W^*) \) and \( NK^2 \) for a particular problem. If the problem is simple or has a large number of patterns, then \( N(W^*W^*)/K^2 \) is small and the computation time of the linear program is longer. If the problem is difficult or has a small number of patterns, then \( N(W^*W^*)/K^2 \) is large, and the linear program is faster.

One additional consideration, however, can modify the above conclusion. In problems where \( K \gg N \) it may not be necessary to use all \( K \) patterns in the design. Generally, the inequalities of a small subset of the patterns will imply the inequalities of all the patterns.\(^6\) (This is the inherent assumption in nonparametric pattern classifier design.) Therefore, if some simple means could be found to determine a subset of the patterns containing the boundary set, then only the subset would have to be used in the design, and the computation time of the linear program would vary with \( N^3 \) instead of \( NK^2 \).

The possible method for determining this subset would be to design some simple explicitly computed classifier with all the patterns. The set of patterns which were incorrectly classified by this classifier or had small values of \( X_k^T W \) could then be used in the linear program under the assumption that this subset contained the

---

\(^6\) Such a subset of the patterns is called the boundary set.\(^\text{[3]}\) It generally consists of \( N \) patterns.
boundary patterns for the total set of patterns. A possible explicitly computed classifier would be the one which uses as weights the difference of the means given by (5) with \( \alpha = \alpha_M \). Another such classifier uses the weights which are optimal for Gaussian patterns with equal covariance matrices for both categories. These weights are essentially [4]–[6]

\[
W = [x^T x]^{-1} x^T M,
\]

where \( M = W \) in (5) with \( \alpha = 1 \).

Another method, which probably would be more satisfactory, would be to use the linear program initially to compute a set of weights using only a small random subset of the total pattern set. The resulting weight vector could then be used to compute \( H \) and \( S \), and a set of \( K \) equations analogous to (9) could be set up. Because the weights initially came from the linear program the simplex method could be started from this set of equations. Assuming that the inequalities of the randomly selected subset correctly imply the inequalities of most of the patterns of the total set, the final solution would be reached in a small number of iterations. This two-step application of linear programming would have the advantage over the explicit classifiers that the final solution reached would be essentially the same as that which would have been computed if the linear program had been started with all the patterns.

**Extrapolation to Nonseparable Patterns and Continuous Variables**

The previous comparison of classifier design methods has been made for separable pattern sets of \(+1, -1\), binary variables. Comments are in order on how the results of this comparison might relate to nonseparable problems and patterns with continuous variables.

In nonseparable problems the main objective is to minimize \( h \). The properties of \( h \) as a function of time are similar for either separable or nonseparable problems. For the adaptive methods \( h \) has the same rapid variation with slowly decreasing mean value; the linear program still generally requires fewer than \( 2K \) iterations to reach a solution. Properties of the relative computation times of the different methods should thus be roughly the same for separable and nonseparable pattern sets.

With continuous variables proper scaling is necessary for adaptive methods if the rate of convergence of \( h \) is to be maintained. The linear program is little affected by the scaling of the variables, either in computational accuracy or number of iterations. Therefore, when the continuous variables are properly scaled for the adaptive methods, the relative computation times should be roughly the same for continuous variables as for binary variables. If they are not properly scaled, the computation times of the adaptive methods will be increased.

**Conclusion**

The problem of designing linear discriminant functions for pattern classification has been formulated as a linear programming problem. In this formulation the linear program minimizes the same objective function as does the fixed-increment adaptive design method. The performance of the linear program in computing classifier weights has been compared experimentally with the steepest-descent iterative method and the one-at-a-time adaptive method in two problems, and the comparison has then been extrapolated theoretically to other problems.

The relative computation times of the linear program and the adaptive methods were found to depend on the relative values of \( K^2 \) and \( N(W^TW^*) \) (where \( W^* \) is the minimum length \( W \)). If \( K^2 \) is significantly larger the linear program will take longer. If \( N(W^TW^*) \) is significantly larger the adaptive methods will take longer. It is conjectured that the \( K^2 \) factor in problems with large \( K \) can always be reduced to less than \( N(W^TW^*) \) by selecting a subset of the patterns that contains the essential patterns of the problem.

The experimental tests were made for separable pattern sets and patterns with binary variables. However, experience with other problems suggests that the results on relative computation time are also valid for problems with nonseparable pattern sets and patterns with continuous variables.

**Acknowledgment**

Acknowledgment and thanks are given to Marilyn Paddleford and Margaret Wright who performed the computer programming for the research reported in this paper.

**References**


