Regressions by Leaps and Bounds

George M. Furnival and **Robert W. Wilson, Jr.**

School of Forestry, Yale University
New Haven, Connecticut

USDA Forest Service
Northeastern Forest Experiment Station

This paper describes several algorithms for computing the residual sums of squares for all possible regressions with what appears to be a minimum of arithmetic (less than six floating-point operations per regression) and shows how two of these algorithms can be combined to form a simple leap and bound technique for finding the best subsets without examining all possible subsets. The result is a reduction of several orders of magnitude in the number of operations required to find the best subsets.

KEY WORDS

Linear Regression
Regression Computation
Subset Selection

1. INTRODUCTION

An investigator involved in a multiple regression analysis with k independent variables often suspects, and even hopes, that a subset of these variables may adequately explain his data. It may well be that the main purpose of the investigation is simply to identify the factors of importance in some process or phenomenon. Subset selection is also employed when the goal of the analysis is prediction because the full equation on all k variables is often unstable. The ridge regression approach of Hoerl and Kennard (1970) may be preferable here but the elimination of variables is an attractive strategy when costs of measurement are large.

Many of the criteria which have been suggested for use in identifying the 'best' subset are monotone functions of the residual sum of squares (RSS) for subsets with the same number of independent variables (Hocking, 1972). Hence, the problem of finding the 'best' subset can often be reduced to the problem of finding those subsets of size p, $p = 1, 2 \cdots k - 1$, with minimum RSS. The PRESS statistic described by Allen (1971) is an exception, but the adjusted R-square, the minimum mean square residual, and the C_p statistic of Mallows (1966; also described in Draper and Smith, 1966) are all monotone functions of the RSS.

The search for the subsets with minimum RSS can be approached in a straight forward manner by computing all possible regressions but the amount of computation required can be formidable. The number of possible subsets increases exponentially with k and the number of operations (multiplications and divisions) required to invert the moments matrix associated with each subset is of order k^3. The cost in computer time is large enough to make the procedure impractical for even moderate values of k; hence it is not surprising to find a number of investigators engaged in (1) attempts to reduce the amount of computation involved in examining a subset and in (2) developing procedures for finding the best subsets without examining all possible subsets.

The present paper is concerned with both approaches to computational efficiency. We will describe several algorithms for computing the residual sums of squares for all possible regressions with what we believe to be a minimum of arithmetic, and we will show how two of these algorithms can be combined to form a leap and bound technique for finding the best subsets without examining all possible subsets.

2. ALL POSSIBLE REGRESSIONS

Garside (1965) and Schatzoff, Fienberg and Tsao (1968) have described algorithms for computing all possible regressions which are much superior to the naive approach involving the direct inversion of the moments matrix associated with each subset of independent variables. The average number of operations per regression for the direct approach is of order k^3 whereas both Garside and Schatzoff achieve an order of k^2 by repeated application of an ingenious technique for modifying one inverse to produce another.

Although the two algorithms are quite similar, the one developed by Schatzoff *et al.* requires less than half as much computation as that described by Garside. The reduction is achieved by taking advantage of the symmetry of the moments matrices and by the deletion of unneeded rows and columns

Received June 1972; revised March 1974

as the successive inverses are computed. A further reduction of $\frac{1}{3}$ or more, depending on the size of k, can be obtained (Furnival, 1971) by abandoning the idea of computing each new inverse from its immediate predecessor. More rows and columns can be deleted and a corresponding gain in efficiency achieved by returning to inverses produced in earlier stages of the computations.

Unfortunately, we have now arrived at what appears to be a dead end. The number of operations per regression is still of order k^2 and it does not seem possible to generate the $2^k - 1$ inverses with less computation. However, the limiting word here is inverses. If we are satisfied with less output for each regression, further savings are possible. We can, for example, compute the regression coefficients, their variances and the residual sum of squares with a number of operations per regression which is of order k and, if we are satisfied with only the residual sum of squares, the number of operations per regression can be reduced to slightly less than six (Furnival, 1971).

2.1 *The Matrix Operators.*

Several authors (Beaton, 1964; also quoted in Schatzoff *et al*, 1968) have described matrix operators which can be conveniently used in computing full inverses for all possible regressions. We will describe two additional operators. The first, which is often called Gaussian elimination, produces only residual sums of squares. The second, which we will call a semi-sweep, produces regression coefficients and the diagonal elements of the inverses as well as the RSS.

Both of our operators assume a $(k + 1) \times (k + 1)$ product (or correlation) matrix with row and column $k + 1$ containing the products associated with the dependent variable. We begin with the matrix stored in the first block of a three-dimensional array A(L,I,J) where L,I, and J are the block, row and column indices. Then at each step of our procedure we produce a submatrix containing the statistics for a subset regression by pivoting with one of our matrix operators on either the original matrix in block one or some submatrix stored in another block as the result of a previous pivot.

An explicit definition of our version of Gaussian elimination is given in the following Fortran subroutine:

```
  SUBROUTINE GAUSS (IB, IS, IP, A, KP)    1
  LB = IP + 1                             2
  DO 1 L = LB, KP                         3
  A(IS, IP, L) = A(IB, IP, L)/A(IB, IP, IP)  4
  DO 1 M = L, KP                          5
1 A(IS, L, M) = A(IB, L, M)
    - A(IB, IP, M)*A(IS, IP, L)           6
  RETURN                                  7
```

The variable arguments are IB, the index of the source block, IS, the index of the storage block and IP, the index of the pivot row and column. A is the three dimensional storage array and the value of KP is $k + 1$. The subroutine operates only on the upper half of the symmetric matrix and only on those rows and columns with indices greater than or equal to IP. At the conclusion of a pivot or elimination, the element A(IS,KP,KP) contains the sum of squares of residuals (RSS) for one of the subset regressions.

The semi-sweep operator requires a list, IND, of the previous pivots on a submatrix and, since it will not always be convenient for us to begin with the first element of IND in storing these pivot indices, we also include in the calling sequence IA, the location in IND of the first pivot index and IZ, the location of the last pivot index.

```
  SUBROUTINE
    SEMI (IB, IS, IP, A, KP, IND, IA, IZ)   1
  A(IS, IP, IP) = 1.0/A(IB, IP, IP)         2
  CALL GAUSS (IB, IS, IP, A, KP)            3
  IF (IA.GT.IZ) TO TO 2                     4
  LB = IP + 1                               5
  DO 1 L = IA, IZ                           6
  B = A(IB, IND(L), IP)/A(IB, IP, IP)       7
  A(IS, IND(L), IND(L)) = A(IB, IND(L),
    IND(L)) + B*A(IB, IND(L), IP)           8
  DO 1 M = LB, KP                           9
1 A(IS, IND(L), M) = A(IB, IND(L), M)
    - B*A(IB, IP, M)                        10
2 RETURN                                    11
```

The arguments IB,IS,IP,A and KP are as defined for GAUSS. Statements 5–10 operate above the pivot row; the diagonal elements are processed by statement 8 and the elements to the right of the pivot column by statements 9 and 10. The major difference between our semi-sweep and a full sweep is that we do not operate on off-diagonal elements to the left of the pivot column.

At the conclusion of a pivot, the element A(IS,KP,KP) again contains the sum of squares of residuals for a subset regression. In addition, the elements A(IS,I,I) and A(IS,I,KP) contain, respectively, the diagonal element of the inverse and the regression coefficient associated with the I-th independent variable. The off-diagonal elements of the inverse are not computed.

2.2 *The regression tree.*

The sequences of pivots utilized in our approach to the computation of all possible regressions are derived from the binary tree of Figure 1. At the root of the tree is the original matrix and at each interior node a submatrix derived from the original matrix by a series of pivots (solid lines) and deletions (dotted lines). Finally, each terminal node or leaf

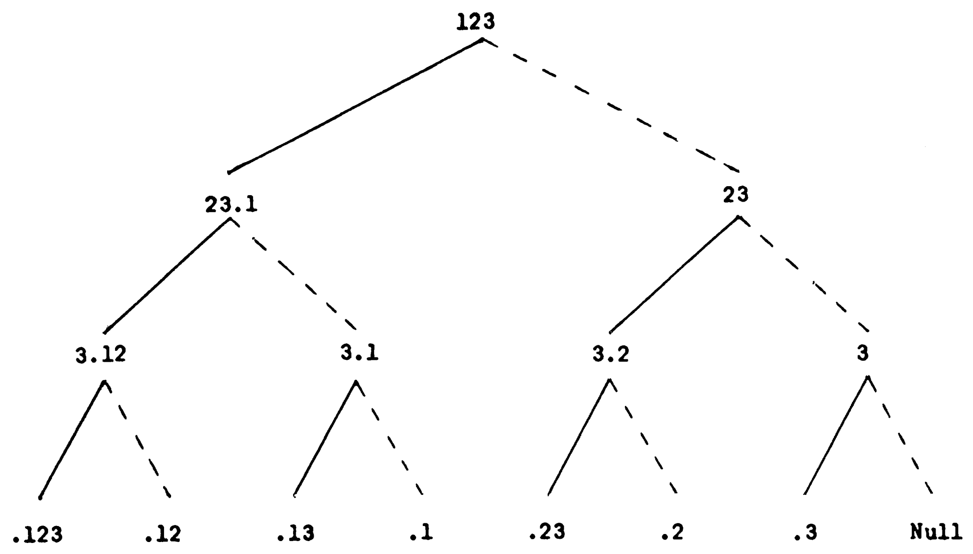

FIGURE 1—The regression tree

represents one of the 2^k possible subset regressions including the null regression.

The labeling of the nodes utilizes a dot notation similar to that employed for partial correlation coefficients. The integers listed before the dot are the subscripts of independent variables present in the submatrix on which pivots have not yet been performed; the subscripts following the dot correspond to variables on which pivots have been performed. Missing subscripts indicate that the rows and columns associated with those variables have been deleted in deriving the submatrix from the original matrix. Thus, the submatrix 3.1 has been obtained from the original matrix by pivoting on X(1) and deleting X(2). A row and column associated with the dependent variable is, of course, always present.

The tree is constructed by beginning at the root and 'splitting' the matrix into two new submatrices—one obtained by pivoting on the first variable of the matrix, the other by deleting the row and column associated with that variable. The process is repeated for the submatrices until all variables have been treated either by pivoting or by deletion.

The binary nature of the tree can be used as the basis for an argument that our approach to all possible regressions is at least as efficient as any other procedure utilizing a sequence of Gaussian eliminations. We argue that there must be at least one pivot on the full matrix and, without loss of generality, assume that this pivot is performed on variable one. The result is a $k - 1$ variable submatrix conditioned on X(1) and it is obvious that

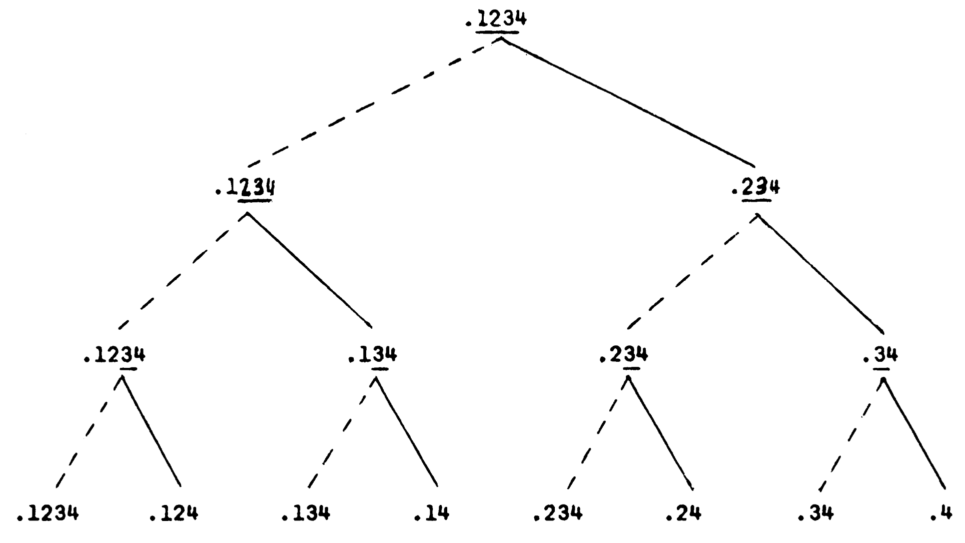

FIGURE 2—The bound tree

all regressions containing X(1) can be derived more easily from this submatrix than from the original matrix. The remaining regressions, those without X(1), can clearly be obtained from the other half of the split—that is, from the $k - 1$ variable submatrix formed by deleting X(1) from the full matrix—just as easily as from the full matrix. We can, therefore, proceed recursively by applying our argument calling for at least one pivot on the full matrix to the submatrices and, finally, after the k-th round of splits, we arrive at the residual sums of squares for the $2^k - 1$ regressions of the problem with the knowledge that the amount of computation involved could not be reduced by using some other pattern of pivots and deletions.

Pivots will have been performed on one k-variable matrix, two $(k - 1)$-variable matrices and so on down to 2^{k-1} one-variable matrices. Thus, one-half of the $2^k - 1$ regressions will have been computed by pivoting on submatrices containing only one independent variable.

It might appear that the tree specifies a rigid order of computation but, in fact, quite a bit of flexibility is permitted. Figure 3 gives a condensed version of a four-variable tree with the dotted lines omitted. Deletions are now implied and interior nodes as well as terminal nodes represent regressions. The tree can be traversed in any 'biologically feasible' order—the only restraint is that a father be 'born' before his son.

The most obvious approach is to search the tree horizontally, level by level, from top to bottom. This procedure produces the regressions in a convenient and natural order—all one-variable regressions followed by all two-variable regressions and so forth as shown in Table 1. The drawback is that all of the submatrices produced in the traverse of a level must be stored until they in turn have been utilized in the pivots required for the traverse of the next level.

Much less storage (no more than $k + 1$ storage blocks) is required if the tree is searched vertically branch by branch and there are at least three useful variations here. The first is obtained by beginning at the root and moving from father to older son at an interior node. At a terminal node, the move is to the next younger brother, or if there is no brother, to the father's next younger brother or, if there is no remaining uncle, to the grandfather's next younger brother and so on as necessary. The process ends with a return to the root and the regressions are produced in a dictionary-like or lexicographic order (Table 1).

The second variation is obtained by applying the vertical search procedure just described to the tree in Figure 4 which is nothing more than the mirror image of Figure 3 with the indexing of the variables reversed. The regressions are produced in what we refer to as 'binary' order (Table 1). That is, if the regressions are numbered with k-digit binary integers in the order in which they are calculated, then the variables present in a regression can be determined from the bit pattern of the integer as illustrated by the following series:

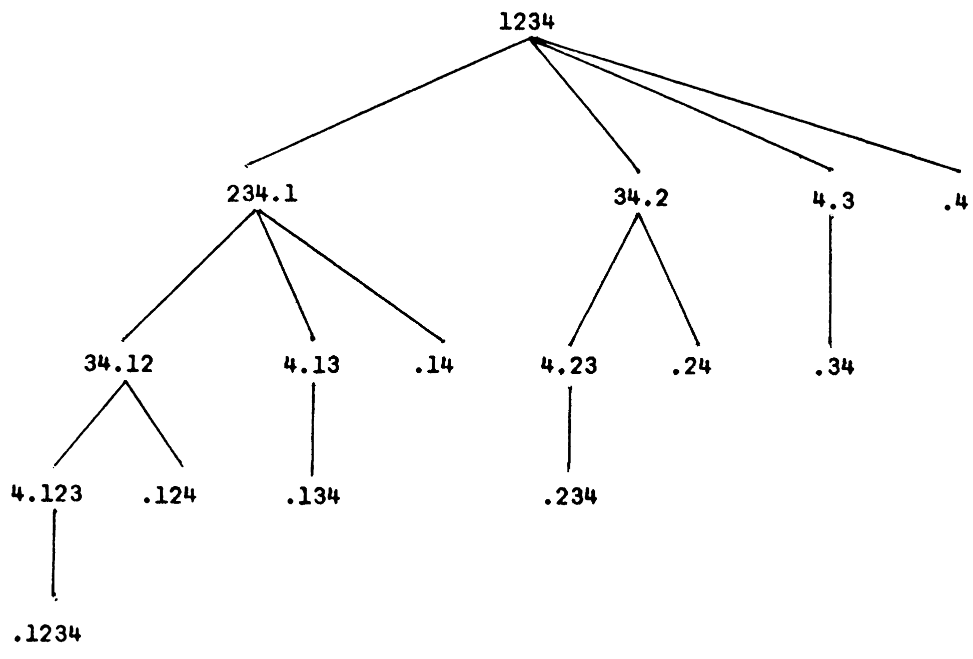

FIGURE 3—The natural and lexicographic tree

TABLE 1—*Sequences of Regressions*

Natural	Lexicographic	Binary	Familial
1	1	1	1
2	12	2	2
3	123	12	3
4	1234	3	4
12	124	13	12
13	13	23	13
14	134	123	23
23	14	4	123
24	2	14	14
34	23	24	24
123	234	124	34
124	24	34	124
134	3	134	134
234	34	234	234
1234	4	1234	1234

Binary Integer	Regression Variables
0001	1
0010	2
0011	12
0100	3
. . .	. . .

Our final method of traversing the regression tree is again applied to Figure 4 and is something of a hybrid with both horizontal and vertical elements. We again move from father to older son as previously described but, when a node is visited, the sons of that node, not the node itself, are listed in order of age from oldest to youngest. The result is sometimes described as familial order since all the siblings of a family are listed together. Again $k + 1$ storage blocks are required; the storage can be this small only because there are never more than $k - r + 1$ sons in a family in the r-th generation.

2.3 *Programming.*

Algorithms for traversing the regression tree in natural, lexicographic, binary and familial order will be given in the form of short Fortran programs. In principle, we could employ either our semi-sweep or Gaussian elimination or, for that matter, the Beaton sweep, in any of the four algorithms. However, in our Fortran implementations of the binary and familial traverses, the list of previous pivots required for the sweep operators is not produced as an integral part of the programs.

It is not necessary in the programming of the traverses to make explicit provision for the deletions of Figure 1. The rows and columns to be deleted before a pivot are always those with subscripts greater than the index of the last previous pivot and less than the index of the up-coming pivot. Hence, these deletions are performed automatically by GAUSS which operates only on those rows and columns with subscripts greater than or equal to the pivot index. SEMI accomplishes the same purpose by utilizing a list of previous pivots to limit its operations.

The sequence of pivots employed in the natural traverse is conceptually quite simple. We begin with the original matrix and pivot on each row in turn beginning with the first. Each pivot produces a submatrix and we pivot in turn on the remaining rows of these submatrices and so on until the submatrices have been exhausted.

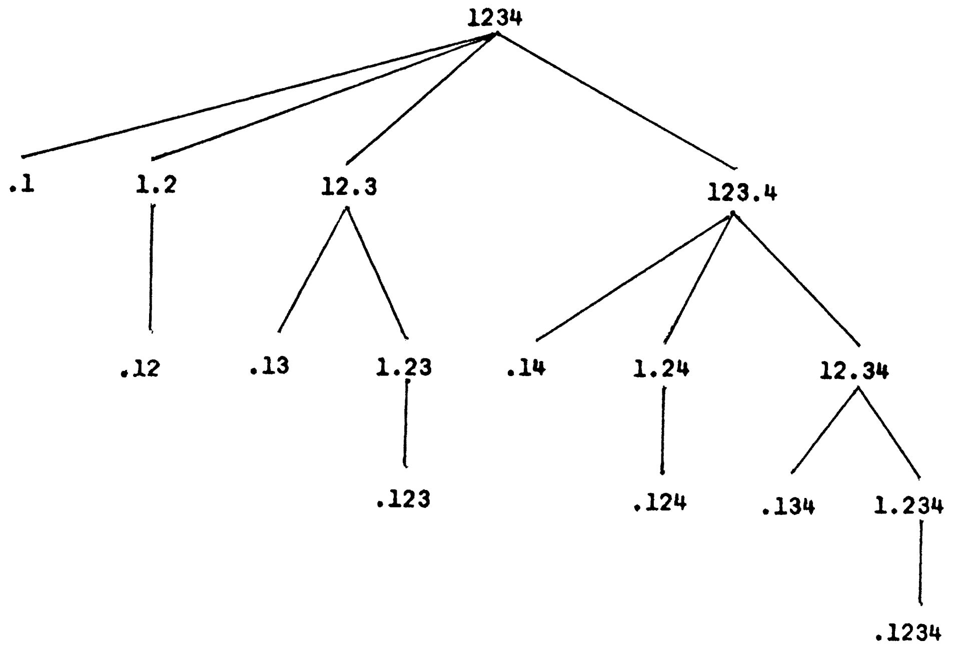

FIGURE 4—The binary and familial tree

```
     DO 1 L = 1, K                                  1
   1 IND(L) = 0                                     2
     M = K                                          3
     IB = 0                                         4
     IS = 1                                         5
   2 IB = MOD(IB, MAX) + 1                          6
     DO 3 L = M, K                                  7
     IF(IND(L).LT.L) GO TO 3                        8
     IND(L - 1) = IND(L - 1) + 1                    9
     IND(L) = IND(L - 1)                           10
   3 CONTINUE                                      11
   4 IND(K) = IND(K) + 1                           12
     IS = MOD(IS, MAX) + 1                         13
     CALL SEMI(IB, IS, IND(K), A, K + 1,
       IND, M, K - 1)                              14
     WRITE(6,  ) (IND(L), A(IS, IND(L),
       K + 1), L = M, K)                           15
     IF(IND(K).LT.K) GO TO 4                       16
     IS = IS - 1                                   17
     IF(IND(M).EQ.M) M = M - 1                     18
     IF(M.GT.0) GO TO 2                            19
     STOP
```

Statement 15 in the program writes the index and the coefficient associated with each independent variable in a subset regression and statements 7–12 and 16–19 identify the regressions. MAX is the dimension corresponding to the first subscript of A; a value of 150 is large enough for a problem with ten independent variables. The diagonal element of the inverse needed for computing the variance of a regression coefficient is available in A(IS,L,L) but we omit this calculation in the interest of brevity.

Our lexicographic algorithm is little more than a method of labeling the subset regressions. We pivot on the last variable in the regression; the index of the storage block is the index of the pivot plus one; and, because of this storage convention, the index of the source block is the index of the previous pivot plus one. Thus, if we are computing the subset regression on variables 1, 2 and 4; then IP is four, IS is five (4 + 1) and IB is three (2 + 1).

```
     IND(1) = 0                                     1
     M = 1                                          2
   1 M = M + 1                                      3
     IND(M) = IND(M - 1) + 1                        4
   2 CALL SEMI(IND(M - 1) + 1, IND(M) + 1,
       IND(M), A, K + 1, IND, 2, M - 1)             5
     WRITE (6,  ) (IND(L), A(IND(M) + 1,
       IND(L), K + 1), L = 2, M)                    6
     IF(IND(M).LT.K) GO TO 1                        7
     M = M - 1                                      8
     IND(M) = IND(M) + 1                            9
     IF(M.GT.1) GO TO 2                            10
     STOP                                          11
```

We recommend the use of this algorithm when coefficients and other statistics such as variances and F-ratios are to be computed and printed for each subset regression. The space requirements are much less than for the previous algorithm and the order in which the regressions are produced is reasonably intelligible.

The binary algorithm also operates from the subscripts of the independent variables in the subset regressions. However, the identification array, NK, is a binary counter with a list of ones and zeros which indicate the presence or absence of the independent variables and the indexing of the independent variables is reversed—that is, the first row (and column) of the matrix contains the products with X(K), the next with X(K − 1) and so forth to X(1)—but the dependent variable is still X(K + 1). The use of the binary array requires that the array be scanned to obtain the indices of the present and immediately preceeding pivot; the reverse indexing requires that IB, IP and IS be complemented.

```
     DO 1 L = 1, K                                  1
   1 NK(L) = 0                                      2
     NK(K + 1) = 1                                  3
     L = 1                                          4
   2 NK(L) = 1                                      5
     DO 3 M = L, K                                  6
     IF(NK(M + 1).EQ.1) GO TO 4                     7
   3 CONTINUE                                       8
   4 CALL GAUSS(K - M + 1, K - L + 2,
       K - L + 1, A, K + 1)                         9
     WRITE(6,  ) A(K - L + 2, K + 1, K + 1),
       (NK(N), N = 1, K)                           10
     DO 5 L = 1, K                                 11
     IF(NK(L).EQ.0) GO TO 2                        12
   5 NK(L) = 0                                     13
     STOP                                          14
```

Statements 5 and 11–13 increment the binary step counter, NK. The incrementation locates the position of the lowest order 1 in NK and this position determines the index of the pivot. Statements 6–8 locate the next lowest order 1; this second position determines the index of the previous pivot and the index of the source block. A variant of this algorithm as described in Frayer *et al* (1971) is recommended when residual sums of squares or R-squares only are desired for all possible regressions with k greater than 12. The output is a very compact table with 16 regressions to a line and 912 to the page.

Our familial algorithm also assumes that the independent variables are in reverse order and it also employs a binary counter. However, the block index is now obtained from the position of the first rather than the second 1 and it is convenient to refer to a stage rather than a step counter. At each stage, we pivot on all of the remaining rows of the submatrix before returning to the counter.

```
     DO 1 L = 1, K                                  1
   1 NK(L) = 0                                      2
     M = K + 1                                      3
```

```
2 NK(M) = 1                                   4
  DO 3 L = 2, M                               5
  CALL GAUSS(K − M + 2, K − L + 3,
    K − L + 2, A, K + 1)                      6
  NK(L − 1) = 1                               7
  WRITE(6,  ) A(K − L + 3, K + 1, K + 1),
    (NK(N), N = 1, K)                         8
3 NK(L − 1) = 0                               9
  DO 4 M = 2, K                              10
  IF(NK(M).EQ.0) GO TO 2                     11
4 NK(M) = 0                                  12
  STOP                                       13
```

The positions of the 1's in the binary counter at any stage correspond to the indices of the previous pivots on the source submatrix. Statement 7 adds a 1 for the current pivot so that NK can be used for labeling the regression and statement 9 sets the bit back to zero.

This algorithm was first programmed in 1958 for the IBM 650 (Furnival, 1958; also described in Ware *et al*, 1962) and later for the 709 and 7094 (Furnival, 1964). It has also been utilized as a screening option in a complete regression package by Grosenbaugh (1967).

2.4 *Discussion.*

The total number of operations (floating point multiplications and divisions) required to compute all possible regressions is the same for the four algorithms but varies with the matrix operator.

Full sweep	$2^{k-3}(k^2 + 11k + 24) - (k^2 + 5k)/2 - 3$
Semi-sweep	$2^{k-1}(3k + 6) - 3(k + 1)$
Gaussian	$6(2^k) - k(k + 7)/2 - 6$

An IBM 370/158 with fast multiply under the H compiler, optimizing level two, performs these operations at a rate of approximately 2,000,000 per minute.

The maximum number of independent variables which can be processed by our algorithms is limited both by the amount of output produced and by the number of arithmetic operations which must be performed. If we adopt an arbitrary output limit of 50,000 individual regression statistics (about 100 compact pages), then the upper limit of k is approximately 11 for a full sweep, 12 for our semi-sweep and 15 for Gaussian elimination.

However, we believe that our algorithms will be most useful in empirical studies of the distribution of regression statistics. In such cases, the output can be limited to summary statistics and the time required to process the subset regressions is the limiting factor. If we adopt an upper limit of five minutes for a medium size computer such as the IBM 370/158 then the maximum value of k is approximately 17 for the full sweep, 18 for our semi-sweep and 20 for Gaussian elimination.

We have been unable to reduce the amount of floating point arithmetic required by our algorithms. However, it is possible to reduce the indexing and fixed point housekeeping by performing several (r) preliminary splits and obtaining a number (2^r) of submatrices which are then processed in 'parallel' by one of the algorithms. Each step of the computations then produces 2^r regressions and the amount of housekeeping is greatly reduced. Storage requirements are, of course, increased but can be reduced somewhat by the use of linear arrays. It is probable that algorithms which permit parallel computation will become more useful as array processors such as Illiac IV come into general use.

3. A Branch and Bound Procedure

A number of authors have described procedures for finding the best subset regressions without computing all possible regressions (Hocking and Leslie, 1967; Beale, Kendall and Mann, 1967; LaMotte and Hocking, 1970). All of these methods are based on the fundamental inequality

$$\mathrm{RSS(A)} \leq \mathrm{RSS(B)}$$

where A is any set of independent variables and B is a subset of A. In other words, it is impossible to reduce the residual sum of squares for a regression by deleting variables from that regression.

3.1 *Inverse trees.*

The use of the inequality to restrict the number of subsets evaluated in a search for the 'best' subset regressions is illustrated by the tree diagram in Figure 5. At the root is a five-variable inverse and the subset regressions are computed by pivoting variables out of the regression. The RSS for a node is obviously a lower bound for the RSS of its offspring. Hence, if we arrive at the node $.2\underline{345}$, say, and had already computed one, two, and three-variable regressions with RSS smaller than that for $.2\underline{345}$, then we could ignore the 14 descendents of $.2\underline{345}$.

The values in parentheses are the RSS for the subset regressions and the underlining in the labels at the nodes of Figure 5 indicates what variables will be removed by future pivots on the submatrix. Thus, from the node $.12\underline{45}$ will be formed those regressions which can be obtained by deleting all possible combinations of X(4) and X(5).

Simple branch and bound algorithms for subset selection can be developed by applying our all-possible traverses to inverse trees, but the results are not fully satisfactory. The difficulty is best shown by an example. Suppose we are at $.1\underline{345}$ with two and three-variable regressions which have R-squares smaller than that for $.1\underline{345}$ but our

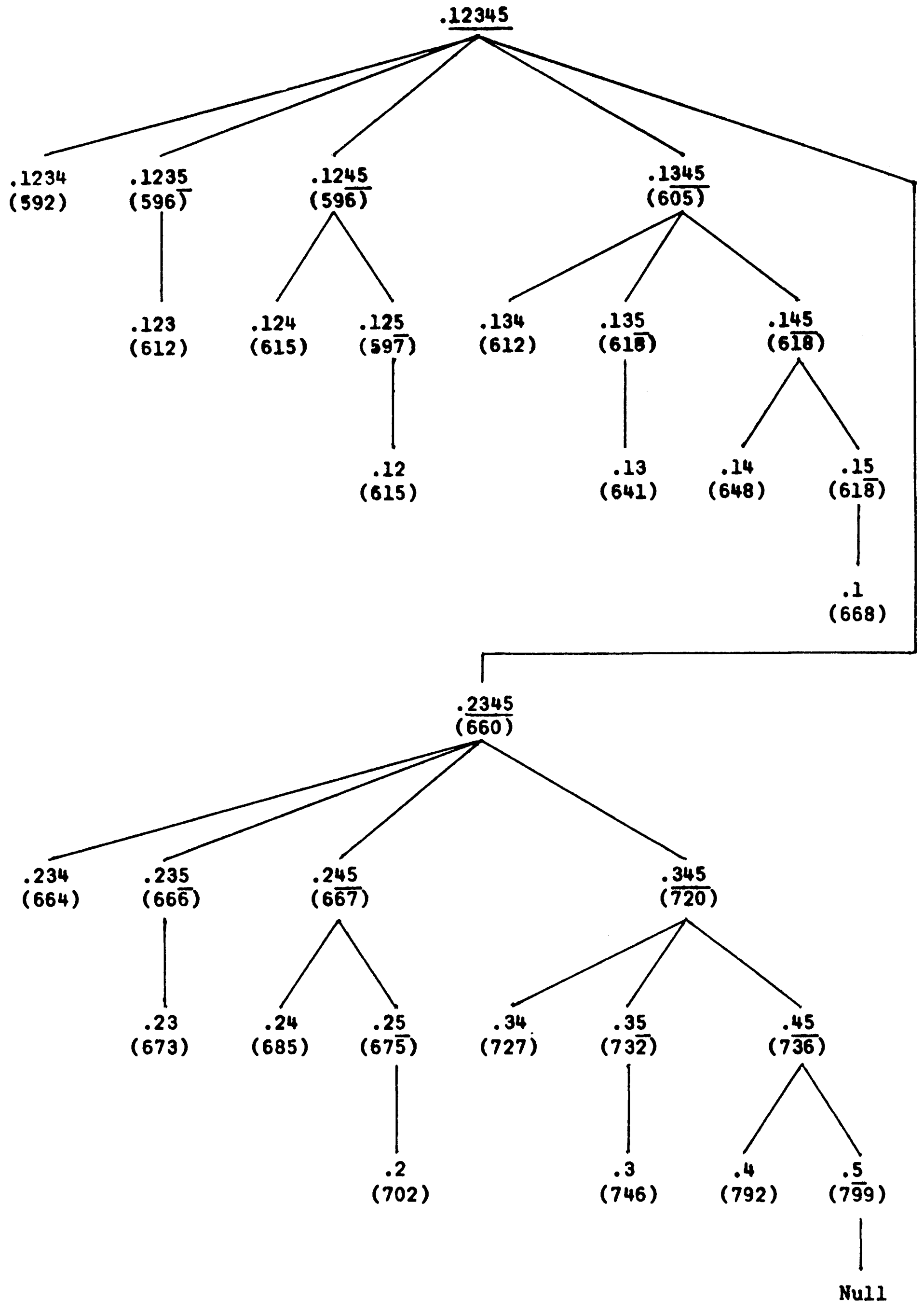

FIGURE 5—The inverse tree

current best one-variable equation fails this test. A reasonable procedure would be to evaluate .1 and ignore the other descendents of .1$\underline{345}$. Unfortunately, the evaluation of .1 requires the prior evaluation of .14$\underline{5}$ and .1$\underline{5}$ and these regressions are of little or no value to us.

The problem becomes more serious with larger trees. The likely candidates for the best regressions, like good fruit, occur at the tips of the branches and can be reached only by pivoting through a lot of dead wood. It is possible to move these good regressions to interior nodes by reversing the order of the variables, but another difficulty emerges immediately. The RSS for the interior nodes are now, of course, small and therefore practically useless as bounds. We assume here that the variables have been ordered by some measure of importance such as the magnitude of the t-ratios in the full k-variable equation.

It seems that we require an impossibility—two arrangements of the same tree. Our solution is two trees—one for bounds and one for regressions. The bound tree is obtained by eliminating all pivots on variable five from the original inverse tree. All the remaining regressions must then include variable five since this variable is never pivoted out of the regressions. The effect is simply to prune off the terminal nodes of the tree of Figure 5.

The other half of the regressions—those without variable five—are included in the regression tree and this need not be an inverse tree. A step-down procedure here would, in fact, defeat our purpose which is to cluster regressions with a small number of independent variables near the root of the tree where they can be reached early in the traverse. Furthermore, the forward approach involves fewer pivots and possibly less rounding error.

Our object is to work out the branches of the full inverse tree (Figure 5) by simultaneous traverses of the regression tree and the bound tree. We observe that each submatrix produced by the traverse of the bound tree will serve as the source for a sub-family or branch of regressions all of which contain X(K) and these regressions will make up half of the regressions in the corresponding branch of the full inverse tree. The other half of the regressions, those without X(K), must come from the regression tree and we must arrange our traverse so that a source subset will be available to produce them.

A particular branch or sub-family of the full inverse tree can be characterized or described by the presence or absence of certain variables and the presence or absence of these variables also determines the composition of the source sub-matrices. Variables which do not occur in any of the regressions of the sub-family must have been pivoted out of the source subset of the bound tree and the rows and columns associated with these variables need not be present in the regression source. On the other hand, variables that appear in all the regressions of a sub-family must have been pivoted into the regression source and must not have been pivoted out of the bound source. Finally, variables which are present in some regressions of a sub-family but not in others must be included in the bound source and (except for X(K)) must be represented in the regression source by rows and columns on which pivots have not been performed. The subscripts of this last class of variables are underlined in the nodes of the bound tree and appear to the left of the dot in the regression tree.

We next observe that the rows and columns associated with non-underlined variables can be deleted—not pivoted out but simply dropped—from the bound submatrices. These deletions are possible because we are interested only in residual sums of squares. We can, therefore, employ a procedure very similar to Gaussian elimination and we need not operate on rows and columns associated with variables which will not be pivoted out of the regression represented by the submatrix.

Symmetry now begins to emerge. Deletions on the bound side correspond to pivots on the regression side and vice-versa. Thus, for example, the bound source $.2\underline{4}5$ is obtained from $.\underline{12345}$ by deleting X(2) and pivoting out X(1) and X(3). The corresponding regression source 4.2 is developed from 1234, by deleting X(1) and X(3) and pivoting in X(2). We require, therefore, two traverses which are the complements of each other—one deletes where the other pivots. The most satisfactory pairing that we have found applies a lexicographic traverse to the regression tree and a slightly modified (the sons are listed from youngest to oldest) familial traverse to the bound tree.

Figures 1 and 2 show, respectively, the regression tree and the bound tree for a four-variable problem. We return to the full binary representation for these trees in order to emphasize the symmetry and must reduce the number of variables from five to four in order to stay within the confines of a page.

3.2 *The algorithm.*

Our simultaneous traverses produce both a regression RSS and a bound at each step in the computations by parallel pivots on submatrices from the product matrix and its inverse. With testing 'turned off', all possible regressions would be produced in the order shown in Table 2 for $k = 5$. However, with testing 'turned on' for the problem of Figure 5, only those pivots marked with asterisks are performed.

There is no difficulty in combining our lexicographic and familial algorithms to produce the sequences of Table 2; the problem arises in implementing the testing and skipping which permit us to perform, for the problem of Figure 5, only those pivots marked with asterisks. Before each pivot, we will test the RSS for some current best regression against some bound and then leap ahead if the test is successful. The questions are, of course: which regression, which bound and how much? The answers are quite simple. The appropriate best regression has the same number of variables as the regression which is about to be produced by the product traverse; the proper bound is the RSS from the source submatrix in the traverse of the inverse; and the increment for the stage counter is 2^{k-p-1} where p is the pivot index.

We illustrate the procedure with a step-by-step descent through Table 2. The pivots of stage zero must always be performed. At stage 1, the inverse

TABLE 2—*Order of computations for leap and bound algorithm*

Stage Number	Pivot Index	Product Traverse Source:	Regr'n	RSS	Inverse Traverse Source:	Regr'n	RSS	Stage Bound
0	* 1	1234.	234.1	668	$.\underline{1234}5$	$.\underline{234}5$	660	
	* 2	234.1	34.12	615	$.1\underline{234}5$	$.1\underline{34}5$	605	
	* 3	34.12	4.123	612	$.12\underline{34}5$	$.12\underline{4}5$	596	
	* 4	4.123	.1234	592	$.123\underline{4}5$	.1235	596	
1	* 4	4.12	.124	615	$.12\underline{4}5$	.125	597	596
2	* 3	34.1	4.13	641	$.1\underline{34}5$	$.1\underline{4}5$	618	605
	4	4.13	.134	612	$.13\underline{4}5$	.135	618	
3	4	4.1	.14	648	$.1\underline{4}5$	.15	618	618
4	* 2	234.	34.2	702	$.\underline{234}5$	$.\underline{34}5$	720	660
	3	34.2	4.23	673	$.2\underline{34}5$	$.2\underline{4}5$	667	
	4	4.23	.234	664	$.23\underline{4}5$	.235	666	
5	4	4.2	.24	685	$.2\underline{4}5$	.25	675	667
6	3	34.	4.3	746	$.\underline{34}5$	$.\underline{4}5$	736	720
	4	4.3	.34	727	$.3\underline{4}5$	.35	732	
7	4	4.	.4	792	$.\underline{4}5$	.5	799	736

source is $.12\underline{4}5$ and the product source is 4.12. The inverse source always bounds the regressions that can be produced from it and it also bounds the only regression, .124, that can be produced from the product source. It is clear, therefore, that our best three-variable regression should be tested against the RSS for $.12\underline{4}5$. The asterisk indicates failure; the 612 of .123 is larger than the 596 of $12\underline{4}5$. At stage two, the inverse source $.1\underline{34}5$ again clearly bounds the regressions that can be produced from the product source 34.1, and we test against our current best two-variable regression. We use the two-variable regression because further pivots on 34.1 will produce at least a two-variable regression and pivoting out one or more of the underlined variables from $.1\underline{34}5$ will leave at least a two-variable regression. Again the asterisk indicates that our test fails; the 615 of .12 is larger than the 605 of $.1\underline{34}5$.

Our first success occurs on the next pivot of stage 2. Reasoning similar to that described for the earlier pivots leads to a test of the best three-variable regression against the bound .1345 and the test is passed. The 597 of .125 is smaller than 605 and we can leap but not very far. Only one regression will be produced from 4.13 and only one from $.13\underline{4}5$ so we can skip only the last pivot of stage two. The stage increment is 2^{5-4-1} or 1 and moves us to stage 3. A further successful test of a two-variable regression—the 615 of .12 versus the 618 of $.1\underline{4}5$—moves us to stage 4 and our last failure occurs here on the first pivot where the 668 of .1 is larger than the 660 of $.\underline{234}5$.

At the second pivot, however, we are again successful. The 615 of .12 is smaller than the 660 of $.2\underline{34}5$ and the regressions from $.2\underline{34}5$ and 34.2 need not be produced. Each of the sources is effectively a two-variable matrix and each will produce three regressions in two stages. Hence, we skip to stage 6 where a successful test—the 668 of .1 versus the 720 of $.\underline{34}5$—again permits a leap of two stages and our search is completed after evaluating 6 of the 22 possible subsets of stages 1–7.

The source submatrices utilized at each pivot in our procedure include only the rows and columns associated with the dependent variable and those independent variables having subscripts equal to or larger than the pivot index. Each source is, therefore, effectively a matrix with $k - p$ independent variables from which $2^{k-p} - 1$ subset regressions can be formed and these regressions are produced in 2^{k-p-1} stages.

For the lexicographic traverse, a stage is equivalent to a branch of a tree and, for the familial traverse, a stage is all the sons of a father. In either case, it is easily shown that the traverse of an r-variable tree (or sub-tree) requires 2^{r-1} stages. There are 2^r nodes in a condensed tree of which half, 2^{r-1}, are interior nodes and half are exterior. We simply note that each interior node is the father of a family of sons and each exterior node is the tip of a branch.

Just prior to each pivot in our procedure we are effectively at a node of Figure 5 preparing to develop the regressions in the sub-family defined by the node. We have, however, already in hand, in the source submatrix on the product side, the regression at the terminal node of the longest branch. Our stage increment of 2^{k-p-1} simply skips the remaining regressions which could be derived from and are therefore bounded by our node of Figure 5.

3.3 *Programming.*

The initial step in our program is the inversion of the product matrix using a forward step-wise procedure with a tolerance check on the pivot elements. Next we re-order the variables by calculating the reduction in the regression sum of squares that would occur with the removal of each variable from the regression equation; the variable associated with the largest removal sum of squares becomes the new X(1) and so on. The reordered product and inverse matrices, with the new X(K) deleted, are loaded into two-dimensional working arrays labeled AA and BB—the product in AA, the inverse in BB. We complete the intialization phase by setting IP = 1, KM = K − 1, KMM = K − 2, NXS(1) = 0, LS = 0 and the array NK (I) to zero for I = 1,2 · · · KMM.

```
1 DO 3 L = IP, KM                              1
  IS = L + 1                                   2
  NXS(IS) = NXS(IP) + L - IP                   3
  IF(RM(NXS(IS) + 1).LE.BB(K, K))
   GO TO 4                                     4
  IADD(IS) = LS                                5
  DO 2 I = IS, K                               6
  DO 2 J = I, K                                7
  LS = LS + 1                                  8
  AN(LS) = AA(I, J)                            9
  AA(I, J) = AN(LS)
   - AA(L, I)*AA(L, J)/AA(L, L)               10
2 AI(LS) = BB(I, J)
   - BB(L, I)*BB(L, J)/BB(L, L)               11
3 CALL STORE                                  12
  L = KM                                      13
4 LEAP = K - L                                14
  DO 5 M = LEAP, KMM                          15
  IF(NK(M).EQ.0) GO TO 6                      16
5 NK(M) = 0                                   17
  CALL WRITE                                  18
  STOP                                        19
6 NK(M) = 1                                   20
  IP = K - M                                  21
  LS = IADD(IP)                               22
  CALL COPY                                   23
  GO TO 1                                     24
```

In order to save space without unduly complicating the indexing, we work out of the two-dimensional arrays AA and BB but store in the linear arrays AN and AI. This practice also permits us to lag the storage of the product submatrices (statement 9) so that the index of the source block for a stage is the same for both the product and the inverse side of the computations. We begin stage zero with AA and BB already loaded and subroutine COPY has the task of retrieving the source submatrices before the first pivot of each succeeding stage. Statements 13–17 and 20–21 increment the stage counter by the amount 2**(LEAP − 1) and compute a value for IP which is the index of the source blocks and also the index the first pivot of a stage. Statement 3 finds the number of variables in the regressions produced by the pivots on the product submatrix and statement 4 tests the best regression with this number of variables against the RSS from the source submatrix of the inverse. At stage zero, the array, RM, of best regressions contains the best subset regressions found in the initial step-wise inversion.

The pivots are simply Gaussian eliminations and are performed by statements 6–12. The computations, except for the storage lag just described, are identical for the product and inverse submatrices. This simplification is possible because the original inversion utilizes a sweep operator which returns a negative inverse and regression coefficients. Thus, at stage zero and thereafter, the RSS in BB(K,K) is positive but the elements of the inverse in rows and columns IP through K − 1 and the regression coefficients in column K have their signs reversed.

Subroutine STORE is assigned the housekeeping chores of labeling and saving the RSS for the best regressions; subroutine WRITE prints the final output.

Our operating program reorders the variables just prior to the first pivot of each stage using a procedure very similar to that described for the initial ordering. We 'look ahead' and calculate, for each underlined variable in the inverse source submatrix, the removal sum of squares

$$\text{RIN(L)} = -\text{BB(L,K)*BB(L,K)/BB(L,L)}$$

for L = IP,IP+1 · · · K − 1. The new X(IP) is the variable with the largest removal sum of squares.

As a by-product of the reordering, we also obtain sharper bounds. Referring again to the node $.\underline{2345}$ of Figure 5, we see that all one-variable regressions from this source come from the sub-sources $.2\underline{45}$ and $.\underline{345}$. Hence, the smaller RSS, the 667 from $.2\underline{45}$, bounds the one-variable regressions. A similar argument shows that the 666 of $.23\underline{5}$ bounds the two-variable regressions. For the three-variable bound and in general for the last bound of a stage, we use the usual bound BB(K,K). The RSS needed for the sharper bounds can be obtained, without performing the pivots that would normally produce them, by simply adding the appropriate RIN(L) to BB(K,K). Thus, with the RIN(L) ordered by size from largest to smallest and with RIN(K) = 0, the bound for the L-th pivot of a stage is BB(K,K) + RIN(L+1).

The program also offers the option of finding the best m regressions, rather than a single best regression, for each size of subset. The changes in the algorithm are minor. The tests preceding the pivots of a stage are made on the current m-th best, rather than the best, regression with the appropriate number of independent variables; and the test procedure is modified to ensure that no leap occurs if the test associated with any remaining pivot of the stage is not satisfied. This situation can never arise in the simpler version of the program because there the bounds remain constant within a stage and the current best r-variable regression always has a RSS at least as small as the current best regression with a smaller number of variables.

In the search for the m best regressions for each size subset, the program evaluates a number of additional regressions and the best of these are saved for each size subset. The program also saves for each size subset the smallest bound invoking a leap or skip and these bounds are, of course, lower bounds for the RSS of the subsets that have not been evaluated.

3.4 *Discussion.*

Our branch and bound algorithm appears to have some desirable features which are not present in others that have been proposed. First, we make no real distinction between an RSS computed as a bound and an RSS computed for a regression; we use both in working out the branches of the regression tree. Other algorithms treat their bounds and regressions separately and may compute the same RSS twice, once as a bound and again for a regression. In addition, we compute each RSS with a single pivot and never pivot more than twice (only once in product submatrices) on the same row and column of any submatrix. Other algorithms employ a horizontal traverse and form new regressions by pivoting an 'old' variable out and then a 'new' variable into the regression. Thus, the subsets are strung along a lengthy chain of pivots and two pivots are required to move from one subset to another. The results are an increase in computing time and perhaps an accumulation of rounding errors. Furthermore, except for the reordering within stages, we do very little housekeeping beyond that required to do all possible regressions. In fact, our leaps and bounds procedure with testing 'turned off' is a very efficient algorithm for computing all possible regressions. Finally, we can obtain the m-best regressions, rather than a single best regression, for each size of subset and, so far as we know, no other program offers this option.

Some idea of the number of operations required to find the best subsets with our program can be obtained from the following series of trials:

k	$m = 1$	$m = 10$
10	2,192	3,764
15	11,050	23,118
20	66,766	123,412
25	336,575	639,945
30	2,169,708	3,934,714
35	6,301,708	11,614,024

Again, the number of operations can be converted to approximate time in minutes on an IBM 370/158 by dividing by 2,000,000. However, timing and number of operations are strongly data dependent and may vary from that given above by as much as a factor of two in either direction.

We have not had an opportunity to test our program against the Beale, Kendall and Mann algorithm. However, in a series of trials with k varying from 15 to 27, our program was 15–50 times as fast as the LaMotte–Hocking program, and the difference in speed increased with k.

Program decks (300 cards), instructions for use, and a sample problem are available from the authors. The program is designed for use with an existing regression package and is in the form of a subroutine utilizing a correlation or product matrix as input.

References

[1] Allen, D. M. (1971). Mean square error of prediction as a criterion for selecting variables. *Technometrics 13*, 469–475.

[2] Beale, E. M. L., Kendall, M. G., and Mann, D. W. (1967). The discarding of variables in multivariate analysis. *Biometrika 54*, 357–365.

[3] Beaton, A. E. (1964). *The Use of Special Matrix Operators in Statistical Calculus.* Research Bulletin RB-64-51, Educational Testing Service, Princeton, New Jersey.

[4] Draper, N. R., and Smith, H. (1966). *Applied Regression Analysis.* John Wiley and Sons, Inc.

[5] Frayer, W. E., Wilson, R. W., Furnival, G. M. (1971). *FSCREEN (Fast SCREEN) A Computer Program for Screening All Combinations of Independent Variables in Univariate Multiple Linear Regressions.* Dept. of Forest and Wood Sciences. College of For. and Nat. Resources. Colorado State University.

[6] Furnival, G. M. (1958). *Regression Routines.* Mimeo. Yale School of Forestry, New Haven, Conn.

[7] Furnival, G. M. (1964). More on the elusive formula of best fit. *Proceedings, Society of American Foresters,* Washington, D. C.

[8] Furnival, G. M. (1971). All possible regressions with less computation. *Technometrics 13*, 403–408.

[9] Garside, M. J. (1965). The best subset in multiple regression analysis. *Applied Statist., J. R. Statist. Soc.,* Series C, *14*, 196–200.

[10] Grosenbaugh, L. R. (1967). *REX-Fortran-4 System for Combinational Analysis of Multivariate Regression.*

Berkeley, Calif., Pacific S. W. Forrest Exp. Sta. (U. S. Forest Serv. Res. Paper PSW-44).

[11] Hocking, R. R. (1972). Criteria for selection of a subset regression: which one should be used? *Technometrics 14*, 967–970.

[12] Hocking, R. R., and Leslie, R. N. (1967). Selection of the best subset in regression analysis. *Technometrics 9*, 531–540.

[13] Hoerl, A. E., and Kennard, R. W. (1970). Ridge regression: biased estimation for non-orthogonal problems. *Technometrics 12*, 55–68.

[14] LaMotte, L. R. and Hocking, R. R. (1970). Computational efficiency in the selection of regression variables. *Technometrics 12*, 83–93.

[15] Mallows, C. L. (1966). *Choosing a Subset Regression.* Presented at Joint Statistical Meetings, Los Angeles, California.

[16] Schatzoff, M., Fienberg, S. and Tsao, R. (1968). Efficient calculations of all possible regressions. *Technometrics 10, 768–779.*

[17] Ware, K. D., Bickford, C. A., Wilson, R. W. and Mayer, C. E. (1962). A program for regression analysis with the IBM 650 electronic computer. *Journal of Forestry 60*, 645–646.