ARTICLE

ALGORITHM 643: FEXACT: a FORTRAN subroutine for Fisher's exact test on unordered r×c contingency tables

CYRUS R. MEHTA, Harvard University, Cambridge, MA, United States

NITIN R. PATEL

Published: 01 June 1986

ALGORITHM 643
FEXACT: A FORTRAN Subroutine for Fisher's Exact Test on Unordered *r*×*c* Contingency Tables

CYRUS R. MEHTA
Harvard University
AND
NITIN R. PATEL
Indian Institute of Management

The computer code for Mehta and Patel's (1983) network algorithm for Fisher's exact test on unordered *r*×*c* contingency tables is provided. The code is written in double precision FORTRAN 77. This code provides the fastest currently available method for executing Fisher's exact test, and is shown to be orders of magnitude superior to any other available algorithm. Many important details of data structures and implementation that have contributed crucially to the success of the network algorithm are recorded here.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization—*nonlinear programming*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*statistical computing, statistical software*

General Terms: Algorithms

Additional Key Words and Phrases: Graph theory, networks, spanning trees

1. PURPOSE AND DESCRIPTION

1.1 Purpose

A network algorithm was developed by Mehta and Patel [7] for performing Fisher's exact test on *r*×*c* contingency tables. The present paper complements, and should be read in conjunction with, the earlier work (referred to hereafter as M–P). It provides the computer code, in double precision FORTRAN 77, along with many important details of implementation that contributed in crucial ways to the success of the algorithm.

This work was supported by the National Cancer Institute, DHHS grants CA-23415, CA-33019, and by a grant from the Mellon Foundation.
Contact author's address: C. R. Mehta, Department of Biostatistics, Harvard School of Public Health, 677 Huntington Avenue, Boston, MA 02115.

1.2 Background

Given an $r \times c$ contingency table $\mathbf{X}$, let x_{ij} denote the entry in row i and column j, and define row and column sums by

$$R_i = \sum_{j=1}^{c} X_{ij},$$

$$C_j = \sum_{i=1}^{r} x_{ij}.$$

Let

$$\xi = \left\{ \mathbf{Y}: \mathbf{Y} \text{ is } r \times c, \sum_{j=1}^{c} y_{ij} = R_i, \sum_{i=1}^{r} y_{ij} = C_j \right\}$$

denote the reference set of all possible $r \times c$ tables with the same marginal sums as the observed table $\mathbf{X}$. The probability of generating table $\mathbf{Y}$ from this reference set under the null hypothesis of row and column independence is the hypergeometric probability

$$P(\mathbf{Y}) = D^{-1} \prod_{j=1}^{c} C_j!/(y_{1j}!\, y_{2j}! \ldots y_{rj}!), \tag{1.1}$$

where

$$D = (R_1 + R_2 + \cdots + R_r)!/(R_1!\, R_2! \ldots R_r!).$$

If we define

$$\xi^* = \{\mathbf{Y}: \mathbf{Y} \epsilon \xi \text{ and } P(\mathbf{Y}) \leq P(\mathbf{X})\},$$

then the **p**-value associated with the observed table $\mathbf{X}$ is

$$\mathbf{p} = \sum_{Y \epsilon \xi^*} P(\mathbf{Y}).$$

Our problem is to compute **p**. To do so, we represent the reference set ξ as a network of nodes and arcs, and reduce the problem to one of finding all paths through the network whose lengths do not exceed $P(\mathbf{X})$. We refer the reader to Section 2 of M–P for details of network construction and analysis. In the remainder of this paper, we borrow heavily from the notation of Section 2 of M–P.

1.3 Subroutine FEXACT

FEXACT, a FORTRAN subroutine for computing Fisher's exact test, implements the network algorithm described in M–P. The subroutine accepts the observed $r \times c$ contingency table as an input parameter and returns the exact **p**-value. In the process, it calls seven other subroutines, LONGP, SHORTP, COL2, COL3, COL4, PUT, and GET, and one function, TABLE, whose respective roles are described below.

After initializing all arrays and accepting the observed $r \times c$ table $\mathbf{X}$, FEXACT processes each node of the form $(k, R_{1k}, R_{2k}, \ldots R_{rk})$, along with every possible

value of PAST up to that node. Specifically, it verifies whether either

$$\text{PAST} * \text{LP}(k, R_{1k}, R_{2k}, \ldots R_{rk}) \leq D * P(\mathbf{X}) \tag{1.2}$$

or

$$\text{PAST} * \text{SP}(k, R_{1k}, R_{2k}, \ldots R_{rk}) > D * P(\mathbf{X}). \tag{1.3}$$

Thus, one basic task is to compute LP and SP at each node. This is accomplished through calls to the two subroutines LONGP and SHORTP, respectively. These subroutines utilize Theorems 1, 2, and 3 of M–P, which assume that the $r \times k$ tables in the reference set over which we wish to optimize have equal column totals. Since this is not generally the case, Theorem 4 of M–P shows how one can alter the reference set to one containing $(r + 1) \times k$ tables with equal column totals. Subroutines LONGP and SHORTP optimize over the altered reference set.

The problem of determining the magnitude of the longest path, $\text{LP}(k, R_{1k}, R_{2k}, \ldots R_{rk})$, is straightforward. Theorem 1 of M–P provides a closed-form expression for the longest path, which is then computed by subroutine LONGP.

The shortest path, $\text{SP}(k, R_{1k}, R_{2k}, \ldots R_{rk})$ is more difficult to obtain. We must minimize the hypergeometric probability of an $r \times k$ matrix, (a concave function), subject to fixed row totals and fixed but equal column totals. Theorems 2 and 3 of M–P reduce the problem to enumerating the basic feasible solutions of a system of $r \times k'$ linear equations in k' variables, where $k' < r$. This enumeration is equivalent to finding all the spanning trees of a complete bipartite graph connecting k' sources to r destinations. When $k' = 1$, the problem is trivial. For $k' = 2$, 3, and 4, the spanning trees are enumerated by subroutines COL2, COL3, and COL4, respectively, which are all called from subroutine SHORTP. The enumeration is simplified considerably by exploiting the fact that the column sums are all equal in the original $r \times k$ matrix. For the present, we do not permit the number of rows, r, in the observed matrix to exceed 5. Hence, k cannot exceed 4, and we do not need additional subroutines beyond COL4 to enumerate the spanning trees. It is possible, however, to use COL4 even when r exceeds 5 by applying a result (not discussed in M–P) which first reduces the original minimization problem to an equivalent one involving only 5 rows. SHORTP exploits this result when $r = 5$ and the column totals are unequal, for then, as discussed in Theorem 4 of M–P, a new row must be added to the original $r \times c$ matrix to equalize the column totals.

The function $\text{TABLE}(I, J)$ returns the logarithm of the binomial coefficient, $I!/(J!\ (I - J)!)$, and is used in many different portions of the program to avoid machine overflow when these coefficients get too large.

The heart of FEXACT is the logic for processing the nodes in the network. A key feature of this logic is the processing of all stage k nodes before proceeding to stage $k - 1$. Such stagewise processing complicates the computer program considerably, compared to the more natural alternative of depth-wise processing. By *depth-wise processing* we mean probing through successive stages, along a specific path, into the interior of the network until a node satisfying either condition (1.2) or (1.3) is reached. There are, however, major gains in the efficiency of the network algorithm with stagewise processing that offset the added complexity of the programming effort.

We now outline the stagewise node processing logic used by FEXACT. There are five important data arrays in FEXACT; KEY(3001,2), IPOIN(3001,2), STP(10000,2), IFREQ(10000,2), and NPOIN(10000,2). The second subscript of each of these arrays indicates whether the data in that array pertain to an odd or even network stage. Suppose for example that an odd stage is currently being processed, $k = 3$, say. Then KEY(I,1) carries information about stage 3 nodes while KEY(I,2) carries information about stage 2 nodes. The stage 3 nodes (mother nodes) are accessed one at a time from KEY(I,1) by the subroutine GET. While processing each mother node, new nodes are generated for processing at stage 2 (daughter nodes). These are inserted into the array KEY(I, 2) by the subroutine PUT. This continues until stage 0 is reached.

The storage and retrieval of nodes discussed above needs further elaboration. First we note that many nodes can be merged into a single node by appealing to the following powerful theorem:

SYMMETRY THEOREM. *The contribution to the* **p***-value of all subpaths from node* $(k, R_{1k}, R_{2k}, \ldots R_{rk})$ *to node* $(0, 0 \ldots 0)$ *equals the contribution of all subpaths from node* $(k, R'_{1k}, R'_{2k}, \ldots R'_{rk})$ *to node* $(0, 0, \ldots 0)$, *provided* $(R'_{1k}, R'_{2k}, \ldots R'_{rk})$ *is a permutation of* $(R_{1k}, R_{2k}, \ldots R_{rk})$.

The above result, which is a consequence of the form of (1.1), permits us to assess the future contribution of all nodes that are permutations of each other once only, and to merely keep track of the PAST values associated with all such nodes.

The following example illustrates how each node, its various permutations, and all PAST values associated with it, are stored. Suppose $r = 5$, and we are currently processing nodes for stage 3. Consider the two nodes (3, 1, 2, 1, 0, 0) and (3, 1, 1, 2, 0, 0) both of which require processing. By appealing to the symmetry theorem, we can merge both nodes into a single node (3, 2, 1, 1, 0, 0), where the partial row sums, R_{ik}, are rearranged to be in descending order. This equivalent node is stored as the integer

$$\text{KEY} = (KY)^4 * 2 + (KY)^3 * 1 + (KY)^2 * 1 + (KY)^1 * 0 + (KY)^0 * 0,$$

where KY is chosen so that each $R_{ik} < KY$, thereby ensuring that the conversion of the vector $(R_{1k}, R_{2k}, \ldots R_{5k})$ to an integer will be unique. (In FEXACT, $KY = \max(R_{1k}, R_{2k}, \ldots R_{5k}) + 1$). Suppose for the present example that we choose $KY = 50$. Then KEY = 2552, which will be stored in the KEY array in, say, KEY(43,1). Suppose next that there are two PAST values, 30.0 and 70.0, associated with the node 2552. Finally, suppose that there are 5 paths with a PAST value of 30 and 15 paths with a PAST value of 70. The PAST information is saved as follows. There is a pointer in IPOIN(43,1) which locates the first PAST value, 30.0, in the STP array. The number of copies of that PAST value is stored in the IFREQ array and NPOIN points to the location in STP at which the next PAST value associated with node 2552 is located. Figure 1 illustrates the data structure for the above example.

Node 2552 in the above example is processed as follows. Its successor nodes are generated in sequence in accordance with M–P Eq. (2.1). Then SP and LP are computed for each successor node. Conditions (1.2) and (1.3) are verified, based on SP, LP, the current PAST value associated with node 2552, and the

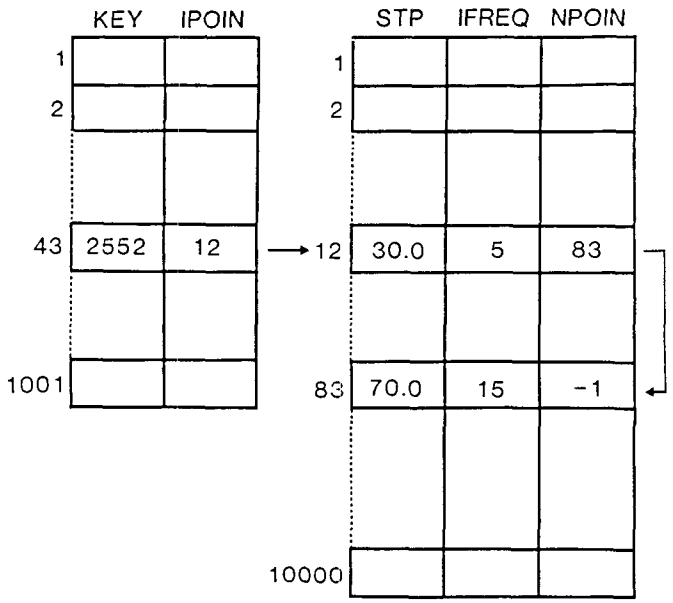

Fig. 1. Data structure for storing a typical node along with its distinct *PAST* values. (The second subscript for each of the arrays has been suppressed for greater clarity.)

length of the arc connecting it to its successor. If either condition (1.2) or (1.3) is satisfied, the next PAST value associated with node 2552 is retrieved from the STP array and conditions (1.2) and (1.3) are verified again. If conditions (1.2) and (1.3) both fail, the successor node is converted to an integer using base KY. The PAST value is updated by multiplying it with the length of the arc connecting 2552 to its successor. The integer version of the successor node is stored in KEY(I,2) and a pointer to the updated PAST value is stored in IPOIN(I,2), where I is determined by hashing using the prime number division method [4]. The updated PAST value is stored in the STP array and it is chained to other STP values (if any) associated with the successor node.

The flowchart given in Figure 2 summarizes the logic of FEXACT.

2. RELATED ALGORITHMS

Early algorithms by March [5], Hancock [3], and Baker [1] were based on exhaustive enumeration of all possible $r \times c$ contingency tables with fixed marginal sums. (Refer to Verbeek and Kroonenberg [11] for an excellent survey.) Unfortunately the number of tables in the reference set grows far too rapidly (Gail and Mantel [2]) for this approach to be feasible, except for trivially small data sets. Two recent algorithms by Mehta and Patel [6, 7] and Pagano and Halvorsen [10] represent a significant advance over earlier methods, because they circumvent the need to enumerate explicitly all the tables in the appropriate reference set. The network-based approach of Mehta and Patel has been particularly successful, dominating all other methods by its efficiency as well as its adaptability to other types of permutation tests [8, 9]. This is demonstrated in the next section.

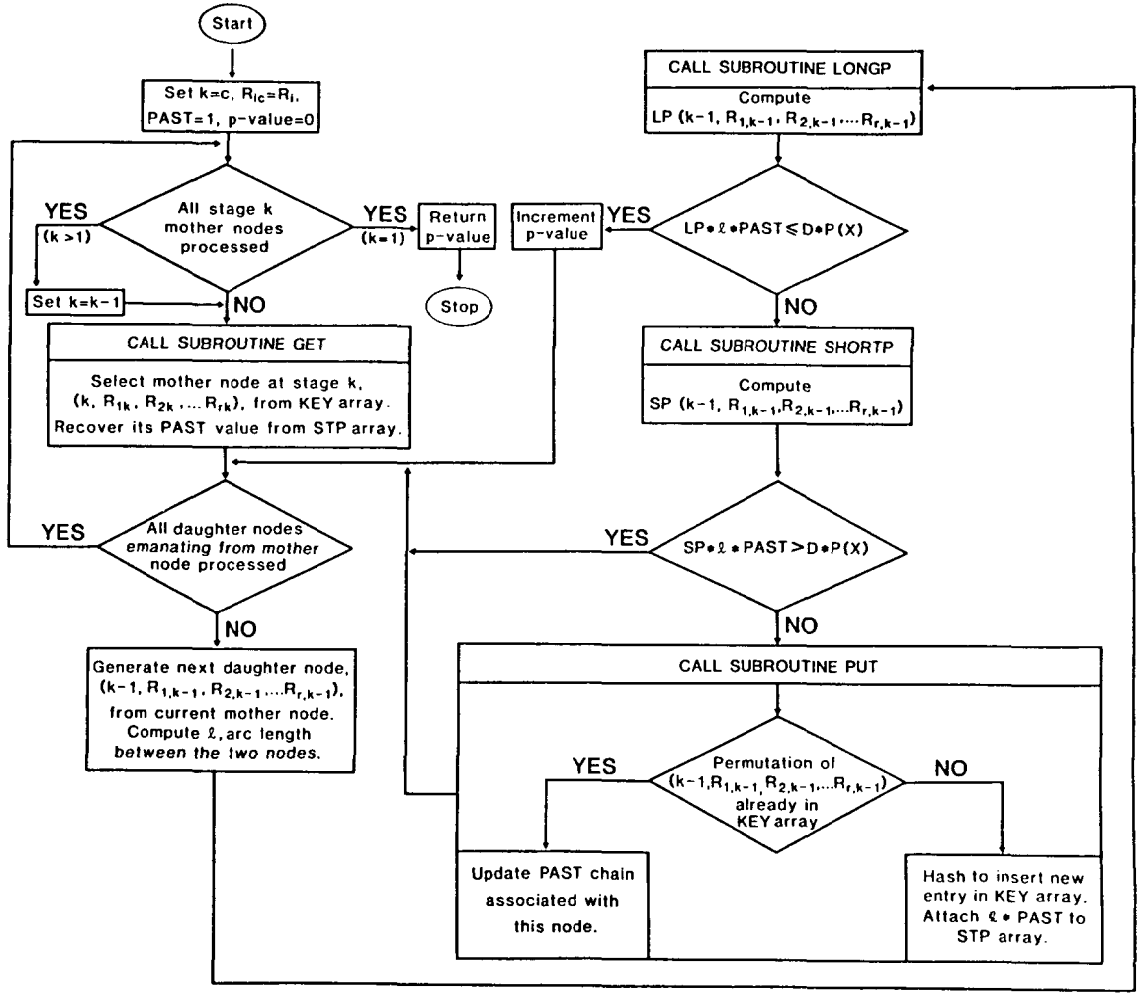

Fig. 2. Subroutine FEXACT.

Table I. Computational Experience with FEXACT on Problems of Varying Size

		p-value		CPU time (minutes)		
Problems	Contingency table	Exact	Approx.	FEXACT	Pagano/ Halvorsen	Number of tables in the reference set[a]
1	1 1 1 0 0 0 1 3 3 4 4 4 4 4 4 4 1 1	.0680	.0605	0.02	0.49	40,500
2	2 0 1 2 6 1 3 1 1 1 1 0 3 1 0 1 2 1 2 0	.0911	.0932	0.24	35.67	1.1×10^6
3	2 0 1 2 6 5 1 3 1 1 1 2 1 0 3 1 0 0 1 2 1 2 0 0	.0454	.0666	1.15	Infeasible[b]	68×10^6
4	1 1 1 0 0 0 1 2 4 4 4 4 5 5 5 6 5 0 1 1 1 0 0 0 1 2 4	.0354	.0935	5.34	Infeasible[b]	624×10^6
5	1 2 2 1 1 0 2 0 0 2 3 0 0 1 1 1 2 7 1 1 2 0 0 0 0 1 1 1 1 0	.0258	.0771	3.04	Infeasible[b]	1.6×10^9
6	1 2 2 1 1 0 1 2 0 0 2 3 0 0 0 1 1 1 2 7 3 1 1 2 0 0 0 1 0 1 1 1 1 0 0	.0393	.1213	14.09	Infeasible[b]	64×10^9

[a] Failed to compute the exact p-value within 180 CPU minutes.
[b] Estimated by the method of Gail and Mantel [2].

3. TEST RESULTS

Exact p-values were computed by FEXACT for problems of varying size. The results are displayed in Table I. Where possible the CPU minutes used by FEXACT have been compared with the corresponding CPU minutes used by the algorithm of Pagano and Halvorsen [10]. Both algorithms have been programmed on a DEC-2060 computer. The algorithm developed by Pagano and Halvorsen is most appropriate for these comparisons because their approach already yields substantial improvements over previously published approaches to this problem. Table I shows that FEXACT considerably extends the bounds of computational feasibility for exact tests of significance in $r \times c$ contingency tables. Sparse matrices with greater than 20 cells are infeasible using the algorithm of Pagano and Halvorsen (because the CPU times exceed 180 minutes), but are easily evaluated by FEXACT. The last column of Table I provides an estimate of the number of tables in the reference set ξ corresponding to each problem. This estimate was computed by a technique due to Gail and Mantel [2], and it explains why explicit enumeration soon becomes computationally infeasible. Note also that for many of these problems the exact and approximate p-value would lead to quite different inferences about row and column independence.

Table I does not provide a good estimate of the CPU ratio for the two algorithms because most of the problems considered were too large to be handled by the

Table II. A Comparison of FEXACT and the Pagano/Halvorsen Algorithms for a Variety of $r \times c$ Contingency Tables

Size of $r \times c$ table	CPU seconds FEXACT	CPU seconds Pagano/Halvorsen	CPU ratio
2 × 4	.32	.82	2.56
2 × 4	.27	.73	2.70
3 × 3	.32	1.05	3.28
3 × 3	.31	.81	2.61
2 × 5	.40	1.73	4.33
3 × 4	.36	1.25	3.47
3 × 4	.89	3.84	4.31
3 × 5	2.38	34.02	14.29
4 × 5	2.31	41.12	17.80
4 × 5	.35	4.39	12.54
4 × 5	6.10	1263.40	207.11
5 × 5	1.30	87.45	67.27
5 × 5	2.04	815.36	399.69
5 × 5	14.22	4910.74	345.34

Pagano–Halvorsen algorithm. Therefore, a number of smaller problems were also considered. Table II shows that FEXACT performs uniformly better for all these problems, and the CPU ratio increases dramatically with the size of the problem.

ACKNOWLEDGMENT

The authors are indebted to Rita Rama Rao for valuable assistance with the code.

REFERENCES

1. BAKER, R. J. Exact distributions derived from two-way tables. *J. Royal Stat. Soc.* Series C, *26*, 2 (1977), 199.–206.
2. GAIL, M., AND MANTEL, N. Counting the number of contingency tables with fixed margins. *J. Am. Stat. Assoc. 72* (1977), 859–862.
3. HANCOCK, T. W. Remark on algorithm 434 [G2]. Exact probabilties for $r \times c$ contingency tables. *Commun. ACM 18*, 2 (Feb. 1975), 117–119.
4. KNUTH, D. E. *The Art of Computer Programming. Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass, 1973.
5. MARCH, D. L. Exact probabilities for $r \times c$ contingency tables. *Commun. ACM 15*, 11 (Nov. 1972), 991–992.
6. MEHTA, C. R., AND PATEL, N. R. A network algorithm for the exact treatment of the $2 \times k$ contingency table. *Commun. Stat. B9*, 6 (1980), 649–664.
7. MEHTA, C. R., AND PATEL, N. R. A network algorithm for performing Fisher's exact test in $r \times c$ contingency tables. *J. Am. Stat. Assoc. 78*, 382 (1983), 427–434.
8. MEHTA, C. R., PATEL, N. R., AND TSIATIS, A. A. Exact significance testing to establish treatment equivalence with ordered categorical data. *Biometrics 40*, 3 (1984), 819–825.
9. MEHTA, C. R., PATEL, N. R., AND GRAY, R. On computing an exact confidence interval for the common odds ratio in several 2×2 contingency tables. *J. Am. Stat. Assoc. 80*, 392 (1985), 969–973.
10. PAGANO, M., AND HALVORSEN, K. An algorithm for finding the exact significance levels of $r \times c$ contingency tables. *J. Am. Stat. Assoc. 76* (1981), 931–934.
11. VERBEEK, A., AND KROONENBERG, P. A survey of algorithms for exact distributions of test statistics in $r \times c$ contingency tables with fixed margins. *Comput. Stat. Data Anal. 3* (1985), 159–185.

Received August 1985; accepted May 1986.