Annals of Economic and Social Measurement, 3/3, 1974

GENERATING SUBMODELS OF A SIMULTANEOUS EQUATION MODEL: AN ALGORITHM USING FISHER'S CORRESPONDENCE PRINCIPLE

BY JAMES STEUERT*

Using Fisher's Correspondence Principle, a graph-theoretic algorithm is presented for generating all simultaneous submodels of a simultaneous equation model. Primitive vertex simple cycles of the directed graphs are combined to generate the simultaneous submodels. The algorithm is efficient and can be used to locate problems of specification, even in models for which the number of simultaneous submodels is prohibitively large. Fisher's convergence criterion is discussed, and the algorithm is applied to the 27-equation Klein-Goldberger model.

I. INTRODUCTION

This paper deals with the problem of applying Fisher's (1) Correspondence Principle to simultaneous equation models. Fisher argues that causality requires simultaneous equation models to be considered limiting approximations to non-simultaneous models of the same form, but with very small time lags. As a result, Fisher derives a set of conditions which well-specified models must satisfy. He suggests tests that can be easily used to locate problems of specification.

The tests are useful because they apply both to the full simultaneous equation model, and to *all* of its simultaneous submodels. A simultaneous submodel is a subset of the full model's set of equations; each variable of a given submodel depends either directly or indirectly on every other variable of the submodel. All variables that are not in the submodel are held fixed and are considered constants or parameters of the submodel.

Because these tests apply to all simultaneous submodels of the full model, they can be used to locate submodels which do not satisfy Fisher's Correspondence Principle and are thus misspecified. A submodel is misspecified if at least one equation in it is misspecified. More specifically, one can suspect the behavior of some equation as a function of a select few of its independent variables, because other variables of that equation that are not in the submodel are held constant. By identifying such failed submodels of the full model, one may be able to find the equations or parts of equations which cause the specification problem.

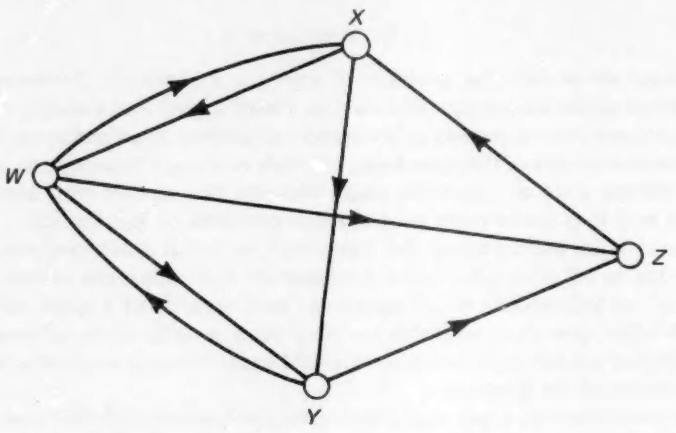
2. DIRECTED GRAPHS AND SIMULTANEOUS SUBMODELS

The application of Fisher's Correspondence Principle thus requires the generation and testing of all simultaneous submodels of the full model. Every variable x of a simultaneous submodel depends on every other variable y of that submodel, either directly through the determining equation of x, as in $x = x(\ldots, y, \ldots)$, or

* This research was supported by National Science Foundation Grant No. GS-2403 to Massachusetts Institute of Technology. I am indebted to Franklin M. Fisher for supervision and advice but retain responsibility for error. Referees for the *Annals* aided in the organization of this paper.

indirectly through the determining equations of other variables (say, z) of the submodel, as in $x = x(\ldots, z, \ldots)$ and $z = z(\ldots, y, \ldots)$. A submodel is simultaneous because of the functional dependencies of its equations, regardless of the specific form of those equations.

The dependencies of a simultaneous equation model may be represented as the set of vertices and arcs of a directed graph. In this representation, each vertex represents a variable (such as x, y, or z), and an arc of the form $x \leftarrow y$ implies that variable x depends directly on variable y via x's determining equation $x = x(\ldots, y, \ldots)$. Given this directed graph of the full model's functional dependencies, a simultaneous submodel is defined as a set of variables (say, w, x, y, z) and the equations that determine them, such that every variable x of that set is dependent on every other variable y of that set, either directly via x's determining equation $x \leftarrow y$, or indirectly through other equations of the submodel (say, z's



$$W = .352 \cdot X + Y^2$$
$$X = W \cdot Z$$
$$Y = 7.6 \cdot X + \text{LOG}_{10} W$$
$$Z = W + Y$$

Figure 1 Directed Graph of a Simultaneous Equation Model

equation, yielding $x \leftarrow z$ and $z \leftarrow y$). See Figure 1 for an example of the directed graph of a simultaneous equation model. The graph theory relevant to generating simultaneous submodels will be discussed later.

3. Generating Simultaneous Submodels

Locating all simultaneous submodels of an n-equation model by direct enumeration (i.e., by testing all 2^n combinations of equations to determine if that combination is simultaneous) is infeasible for all but the smallest models. This paper describes an algorithm which efficiently generates all simultaneous submodels of a large model ($n = 23$) in a small amount of computing time. Further, this algorithm first generates a small set of "primitive" simultaneous submodels which are later combined in special ways to generate all simultaneous submodels. (A simultaneous submodel will henceforth be referred to simply as a submodel.) A submodel is generated or "grown" by combining an already-generated submodel with one of these primitive submodels. This feature of the algorithm is useful in two respects.

First, a submodel generated by combining a failed submodel with a primitive submodel will also include the misspecified equations or functional dependencies of the original failed submodel. The failure (or success) of the combined submodel will not provide more specific information about the location of the problem than that provided by the original failed submodel. For a large simultaneous equation model the application of Fisher's test to all submodels may be infeasible because of the large number of such submodels. By not "growing" submodels from failed submodels, the number of submodels tested may be greatly reduced, without sacrificing any information about the location of misspecified equations and dependencies.

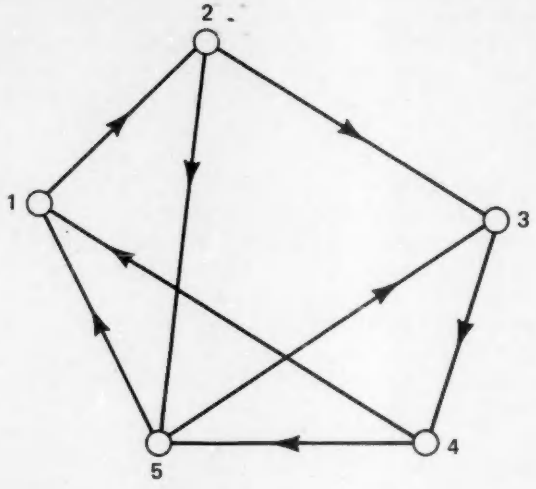
Second, if a successful submodel is combined with a primitive submodel, and if the combined submodel fails the tests, then further information may be derived. Variables (and dependencies) of the failed submodel that are constants of the successful submodel are to be suspected as causes of the failure. In other words, if the primitive submodel is added, some constants of the successful submodel become variables, and the behavior of those variables and their dependencies may be the cause of failure. This provides more specific information about the location of a misspecification than does simply identifying the failed submodel and its equations.

4. Graph Theory

The dependencies of a simultaneous equation model may be represented by a directed graph. In this representation, each vertex of the directed graph corresponds to an endogeneous variable of the model, and an arc corresponds to a functional dependency of the model. It is assumed that the model is consistently normalized with a different endogeneous variable on the left-hand side of each equation. The arcs directed inward toward variable x specify those other variables that x depends on in its determining equation. For example, if $x = x(a, b, c)$, then variable (vertex) x would have the arcs $x \leftarrow a$, $x \leftarrow b$, and $x \leftarrow c$ directed inward toward x.

A simultaneous submodel of a model is a set of variables, along with the equations that determine them, such that each variable is dependent on every other variable via some chain of functional dependencies. Thus, a simultaneous submodel corresponds uniquely to a set of vertices and their connecting arcs, such that there exists a directed sequence of arcs connecting every ordered pair of vertices (a start vertex and an end vertex). Such a directed sequence of arcs is known as a directed path, or simply a path. For example, $a \leftarrow b \leftarrow c \leftarrow d$ is a path between vertices d and a. This path specifies that variable a depends on variable d.

The directed graph of an m-equation model is a set of m vertices, and a set of arcs connecting those vertices. An arc of the form $a \rightarrow b$ is an ordered pair of vertices (a, b), where vertex a is called the *start vertex* of the arc, and vertex b is called the *end vertex* of the arc. A path is thus an ordered sequence of arcs $(A_1, A_2, A_3, A_4, \ldots, A_k)$ where the end vertex of A_i is the same as the start vertex of A_{i+1} (i.e., where the arcs are consecutive). A *simple cycle* is a closed path where no vertex is the start or end of more than one arc of that path; i.e., the start vertex of the first arc of the path is the same as the end vertex of the last arc of the path, and no vertex is repeated. For example, the path $(a \rightarrow b, b \rightarrow c, c \rightarrow a)$ is a simple cycle, and may be represented as a sequence of vertices $a \rightarrow b \rightarrow c \rightarrow a$. For a given set of vertices, there may exist more than one simple cycle which consists of those vertices (Figure 2).



THE SIMPLE CYCLE

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$

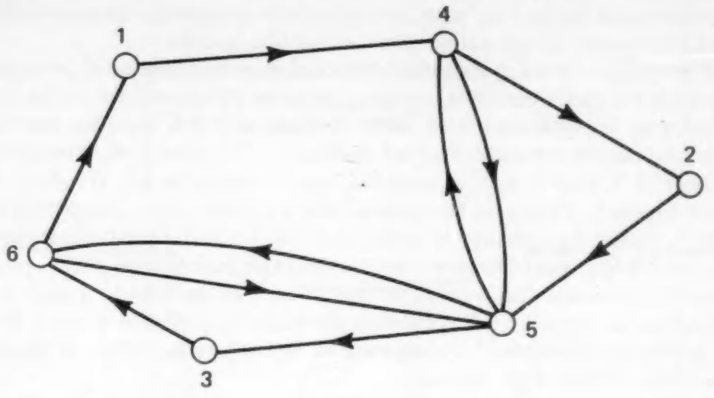
HAS THE SAME VERTICES AS

THE SIMPLE CYCLE

$1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 1$

Figure 2 Two Simple Cycles From One Set of Vertices

Thus, a *vertex simple cycle*, or *v-cycle* for short, is a set of vertices for which there exists at least one simple cycle as previously defined. Two *v*-cycles are said to be connected if they have a vertex in common. A connected sequence of *v*-cycles is an ordered sequence of *v*-cycles, each of which is connected to some previous member of the sequence. The primary result of this theory is that every simultaneous submodel may be represented as the vertices of a connected sequence of *v*-cycles of the full model's directed graph. The following discussion will demonstrate this.



PATHS:

$P_{12} = 1 \rightarrow 4 \rightarrow 2$

$P_{23} = 2 \rightarrow 5 \rightarrow 3$

$P_{34} = 3 \rightarrow 6 \rightarrow 5 \rightarrow 4$

$P_{45} = 4 \rightarrow 5$

$P_{56} = 5 \rightarrow 6$

$P_{61} = 6 \rightarrow 1$

TRAVERSE:

$1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 1$

SEQUENCE OF V-CYCLES:

$$\left\langle \{1\ 4\ 5\ 6\ 1\}, \{4\ 2\ 5\ 4\}, \{5\ 3\ 6\ 5\} \right\rangle$$

PINCH NUMBER $\quad 1 \leq 4 \leq 5$

Figure 3 Generation of Traverse From Paths P_{ii+1}

Let the vertices of a simultaneous submodel be numbered from 1 to m. Because the submodel is simultaneous, there is a directed *path* between every ordered pair of vertices of that submodel. In particular, there is a path between vertex i and vertex $i + 1$, for $i \neq m$; and there is a path from vertex m to vertex 1. Now create a large path by concatenating all these paths, to yield the sequence of paths $(P_{12}, P_{23}, P_{34}, P_{45}, \ldots, P_{M1})$. If each path of this sequence is replaced with its sequence of vertices, this large closed path may be reduced to a single closed path which is a sequence of many vertices connected by arcs. Vertices may occur more than once in this path. Such a closed path is called a *traverse*. Every simultaneous submodel corresponds to at least one traverse. For a given numbering (labeling) of the vertices of the full model's directed graph, every simultaneous submodel corresponds to a unique traverse, assuming that paths P_{ii+1} are chosen by a predetermined unique rule, such as shortest path of smallest numerical ranking. Figure 3 illustrates the generation of a traverse from paths P_{ii+1}.

The traverse of a simultaneous submodel may be represented as a unique sequence of v-cycles. Consider a traverse T, which is a sequence of k vertices, some of which may be duplicated. If no vertex is repeated in this sequence, then T is a v-cycle, and may be represented as just one v-cycle. If a vertex (say, 4) occurs more than once in T, then T may be split into two closed paths by "pinching" T at vertex 4 (Figure 3). T may now be represented as a sequence of two smaller traverses T_1 and T_2, which have vertex 4 in common. If this *pinching* algorithm successively applied to T's sequence of traverses, and if each traverse is replaced with its pinched sequence of traverses, the original traverse T may be reduced to a sequence of v-cycles because pinching will eventually eliminate all duplicate vertices. Thus a traverse may be represented by a sequence of v-cycles, each of which is connected to a previous v-cycle of the sequence.

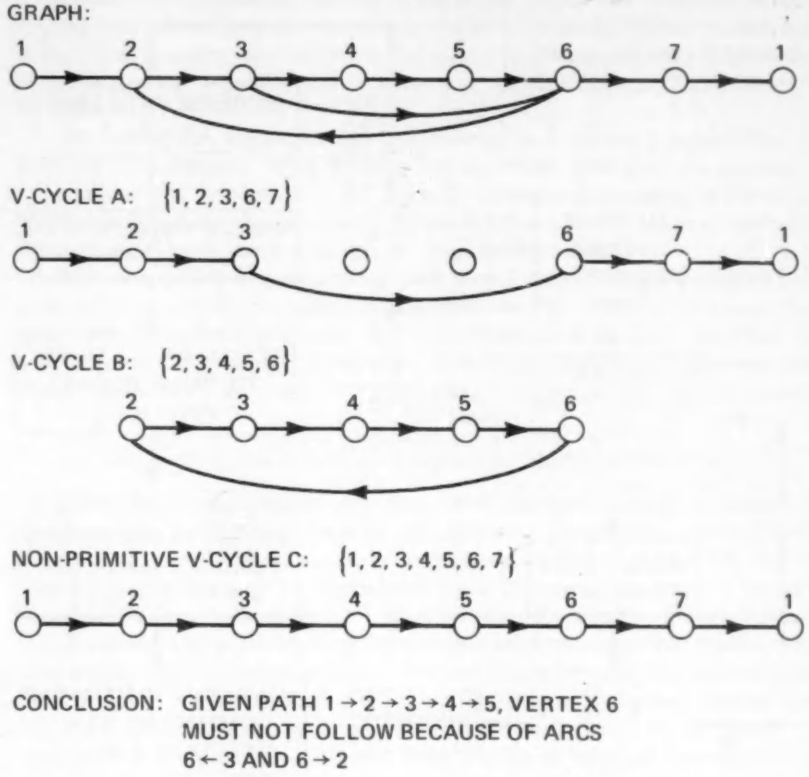
Pinch vertices may be chosen in order of the smallest duplicated vertex first. Therefore, associated with every v-cycle of a connected sequence, there is a "pinch number," which is the smallest numbered vertex that it has in common with any previous v-cycle of the sequence. (See Figure 3 for an illustration of pinch-numbers.) Pinch numbers are necessarily non-decreasing because that would imply an interval which would have been pinched previously, contrary to our rule of selecting smallest pinch vertices first. For example, in Figure 3, a v-cycle with vertex 2 cannot be appended to the current sequences of v-cycles, because the resulting expanded traverse would have included two occurrences of vertex 2 and thus could have been pinched with vertex 2 first rather than with vertex 4. A v-cycle of the sequence must be connected to a v-cycle of a smaller pinch-number.

A primitive v-cycle is a v-cycle which cannot be represented as the union of a connected set of smaller v-cycles. Thus, by definition, any v-cycle may be represented as the union of a connected sequence of primitive v-cycles. If any v-cycle of the above representation is replaced by its representation as a sequence of primitive v-cycles, and if the primitive v-cycles are then grouped by their pinch number, any traverse may be represented by a sequence of primitive v-cycles ordered by increasing pinch-number. The above results are fundamental constraints which make practical the algorithm for generating simultaneous submodels.

5. The Algorithm for Generating Primitive V-Cycles

The theory discussed above suggests an algorithm for generating simultaneous submodels. Basically, the technique involves two steps. First, generate the primitive v-cycles of the full model's directed graph. Second, combine these primitive v-cycles in sequences which generate all simultaneous submodels. The number of v-cycles of a directed graph may be very large. However, the number of primitive v-cycles should be significantly smaller. This turned out to be the case with the Klein–Goldberger model (Appendix A), which yields 111 primitive v-cycles.

The procedure for generating primitive v-cycles is known as a "tree-search". Branches of the tree correspond to the paths of the directed graph that are searched. When a path leads back to the starting vertex of the search, then a v-cycle is found because a closed directed path has been generated. When searching the directed graph, a path is extended by adding to it an arc which connects the end of the present


GRAPH:
1 2 3 4 5 6 7 1
V-CYCLE A: {1, 2, 3, 6, 7}
1 2 3 6 7 1
V-CYCLE B: {2, 3, 4, 5, 6}
2 3 4 5 6
NON-PRIMITIVE V-CYCLE C: {1, 2, 3, 4, 5, 6, 7}
1 2 3 4 5 6 7 1

CONCLUSION: GIVEN PATH 1 → 2 → 3 → 4 → 5, VERTEX 6 MUST NOT FOLLOW BECAUSE OF ARCS 6 ← 3 AND 6 → 2

Figure 4 Pruning: Inhibited Search for Non-Primitive *V*-Cycles

path to another vertex. This newly connected vertex might be the starting vertex, in which case a *v*-cycle is found. However, that vertex must not be a part of the current path, because a simple cycle is a closed path in which no vertex is repeated (the starting vertex is an exception to this restriction). This search procedure must rememeber its current path and the already-searched directions at each vertex of that path, so that it may "back up" and continue the search in another direction when the search is exhausted in a given direction.

The search just described would generate all simple cycles and would be impractical. However, an additional restriction is used to greatly reduce the search time for generating primitive *v*-cycles. This technique prevents the search from generating *v*-cycles which are the union of two smaller, connected *v*-cycles (Figure 4). The "pruning" technique shown in Figure 5 accomplishes this and makes the algorithm practical.

The above procedure, if applied to every vertex as starting vertex with no restrictions, would generate an *m*-vertex *v*-cycle *m* times, once for each of *m* vertices.

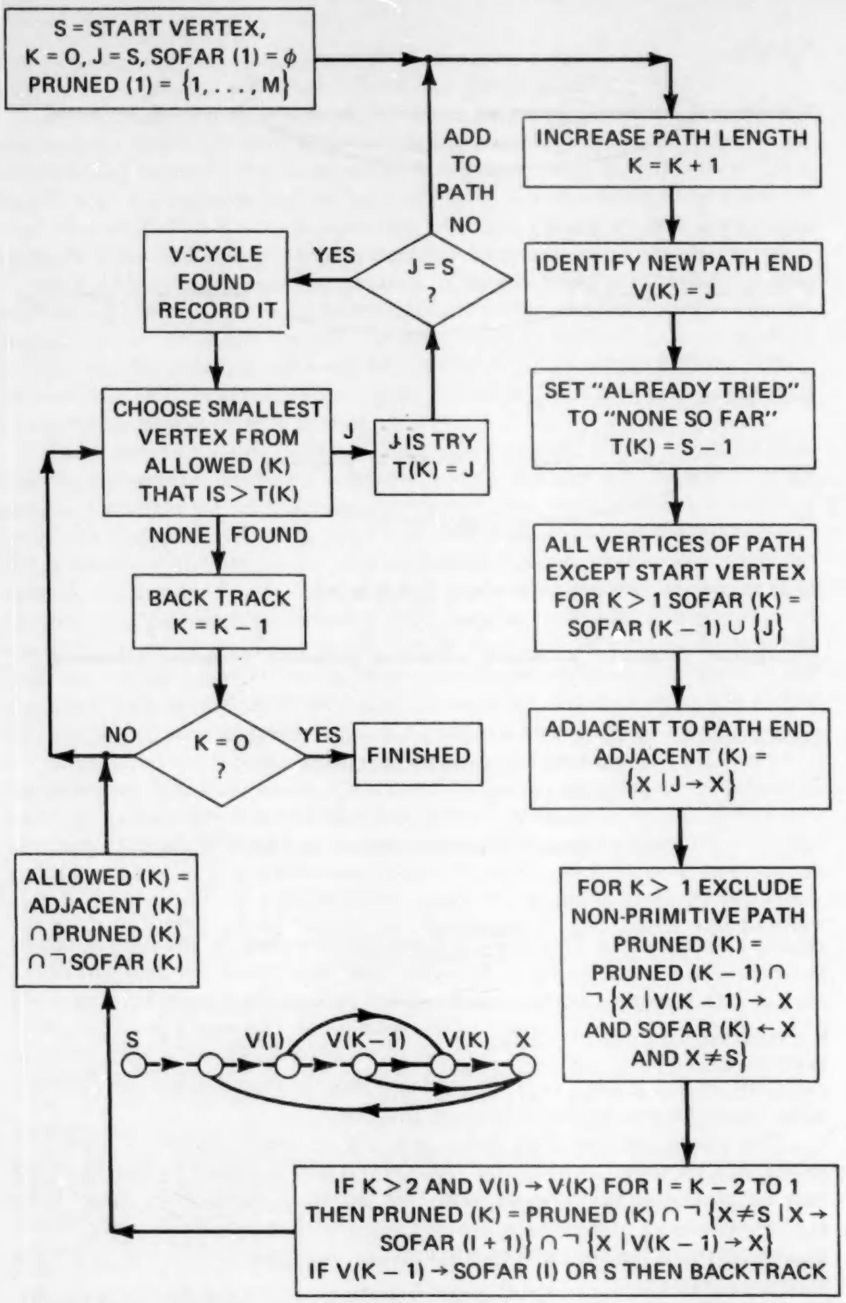

S = START VERTEX, K = O, J = S, SOFAR (1) = ϕ PRUNED (1) = $\{1, \ldots, M\}$
INCREASE PATH LENGTH K = K + 1
IDENTIFY NEW PATH END V(K) = J
SET "ALREADY TRIED" TO "NONE SO FAR" T(K) = S − 1
ALL VERTICES OF PATH EXCEPT START VERTEX FOR K > 1 SOFAR (K) = SOFAR (K − 1) ∪ {J}
ADJACENT TO PATH END ADJACENT (K) = {X | J → X}
FOR K > 1 EXCLUDE NON-PRIMITIVE PATH PRUNED (K) = PRUNED (K − 1) ∩ ¬ {X | V(K − 1) → X AND SOFAR (K) ← X AND X ≠ S}
IF K > 2 AND V(I) → V(K) FOR I = K − 2 TO 1 THEN PRUNED (K) = PRUNED (K) ∩ ¬ {X ≠ S | X → SOFAR (I + 1)} ∩ ¬ {X | V(K − 1) → X}
IF V(K − 1) → SOFAR (I) OR S THEN BACKTRACK
ALLOWED (K) = ADJACENT (K) ∩ PRUNED (K) ∩ ¬SOFAR (K)
NO
K = O ?
YES
FINISHED
BACK TRACK K = K − 1
NONE FOUND
CHOOSE SMALLEST VERTEX FROM ALLOWED (K) THAT IS > T(K)
J
J IS TRY T(K) = J
J = S ?
YES
V-CYCLE FOUND RECORD IT
NO
ADD TO PATH
S
V(I)
V(K−1)
V(K)
X

Figure 5 Generation of Primitive *V*-Cycles by a Depth-First Tree Search Assume a Directed Graph of *M* Vertices.

To prevent this redundancy, the algorithm is first applied to the entire directed graph G with vertex 1 as start vertex, then to G-(1) with vertex 2 as start vertex, then to G-(1, 2) with vertex 3 as start vertex, and so on. Thus every simple cycle generated by this algorithm is generated only once, with its minimum-numbered vertex as start vertex.

The tree-search algorithm was programmed in Assembly Language for the IBM 370/155 computer. With pruning the algorithm generated the v-cycles in 0.352 minutes for the version of the Klein–Goldberger model given in Fisher (1). Without pruning, the algorithm never finished, but would have taken an estimated 240 minutes. Clearly, pruning makes the algorithm practical.

Because a v-cycle may be derived from several distinct simple cycles with the same vertices (but different paths), the algorithm may generate a v-cycle more than once. Also, there is no guarantee that the v-cycles generated are primitive. The v-cycles generated by this algorithm are verified to be primitive by comparing them with all smaller v-cycles, and testing to determine whether or not they are primitive.

6. Generating Simultaneous Submodels from Primitive V-Cycles

Now that a set of primitive v-cycles have been generated, it is necessary to combine them to form simultaneous submodels. As the pinching process demonstrated, every simultaneous submodel may be represented by at least one traverse, and every traverse may be represented by a connected sequence of primitive v-cycles. This sequence of v-cycles must be grouped in order of non-decreasing pinch-number and each v-cycle of the sequence must be connected to some v-cycle of a smaller pinch-number group. In this manner, every simultaneous submodel may be generated by appending to an existing sequence of v-cycles (an already-generated submodel) a primitive v-cycle of pinch number equal to or greater than any previous v-cycle of the sequence. Note that the pinch-number property is not intrinsic to a primitive v-cycle; it is determined only as applied to a currently generated submodel. This procedure is also a tree search, but is of a somewhat different form. Backing up involves changing the primitive v-cycle at a previous element of the sequence, and then looking for all sequences generated by appending primitive v-cycles to that sequence.

The above procedures for generating simultaneous submodels take a total of 2.15 minutes on the IBM 370/155 to generate all the simultaneous submodels of the Klein–Goldberger model given in Fisher (1). There are 111 primitive v-cycles and 25,565 simultaneous submodels. The algorithm is successful mainly because of the pruning technique for generating primitive v-cycles.

7. The Convergence Test

The convergence criterion of Fisher (1) is applied to each of the simultaneous submodels specified by the above algorithm. The requirement is that the average of $G^n(YO)$ converges, where $Y = G(Y)$ is the simultaneous submodel, and YO is the initial vector of the variables endogeneous to that submodel (with all other variables held constant), and where $G^n(YO) = G(G^{n-1}(YO))$ with $G^O(YO) = YO$. In practice, this average is formed iteratively using the formula $Y(n + 1) =$

```
TOTAL NUMBER CF SUBMODELS WHICH HAVE
PASSED CONVERGENCE   = 960
FAILED Y(N+1)-Y(N)   = 111
FAILED Y(N)-G(Y(N))  = 0
BLOWN UP SCMEHCW     = 830

TOTAL NUMBER OF SUBMODELS
 WHICH CONTAIN EQUATION

                        1       WHICH
                                PASSED CONVERGENCE = 573
                                 FAILED Y(N+1)-Y(N) = 68
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SOMEHOW    = 431

                        2       WHICH
                                PASSED CCNVERGENCE = 556
                                 FAILED Y(N+1)-Y(N) = 64
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SOMEHOW    = 450

                        3       WHICH
                                PASSED CCNVERGENCE = 960
                                 FAILED Y(N+1)-Y(N) = 111
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SCMEHOW    = 830

                        4       WHICH
                                PASSED CCNVERGENCE = 534
                                 FAILED Y(N+1)-Y(N) = 55
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SCMEHOW    = 446

                        5       WHICH
                                PASSED CCNVERGENCE = 192
                                 FAILED Y(N+1)-Y(N) = 99
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SCMEHOW    = 824

                        6       WHICH
                                PASSED CCNVERGENCE = 783
                                 FAILED Y(N+1)-Y(N) = 12
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SCMEHOW    = 807

                        7       WHICH
                                PASSED CCNVERGENCE = 484
                                 FAILED Y(N+1)-Y(N) = 12
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SCMEHOW    = 688

                        8       WHICH
                                PASSED CCNVERGENCE = 551
                                 FAILED Y(N+1)-Y(N) = 108
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SCMEHOW    = 377

                        9       WHICH
                                PASSED CONVERGENCE = 175
                                 FAILED Y(N+1)-Y(N) = 9
                                 FAILED Y(N)-G(Y(N))= 0
                                 BLEW UP SCMEHOW    = 256
```

Figure 6 Summary Statistics on Convergence Tests of Submodels of the Klein-Goldberger Model

$(1/(n+1))(Y(n) + nG^n(YO))$ where the average at step n of the iteration is denoted by $Y(n)$. At each iteration, $G^n(YO)$ is formed as $G(X)$, where $X = G^{n-1}(YO)$ was formed at the previous iteration. The algorithm is determined to have converged if $|Y(n+1)_i - Y(n)_i| < \varepsilon_i$ for all $i = 1, 2, \ldots, r$ where r is the number of equations of the submodel; where Y_i is the i-th component of the vector Y of endogeneous variables of the submodel; and where the $\varepsilon_i = 0.001|YO_i|$. In practice, the $Y(n)$ often converged very rapidly to a solution $Y(n) = G(Y(n))$ of the model (within the ε_i given above) whereas the $|Y(n+1) - Y(n)|$ converged much more slowly. The total time taken by the convergence program is largely that of the convergence test since the generation of the submodels takes only about 2 minutes. With the Klein–Goldberger model (Appendix A) it was found that the $|Y(n+1) - Y(n)|$ test had failed for most of the submodels for over 500 iterations, whereas the $Y(n)$ had converged very quickly to $G(Y(n))$. The convergence of the average is apparently much slower than the convergence of the iteration based on G.

Thus the convergence algorithm in its final form is as follows:

(a) For each iteration $n = 1$ to 500, form $G^n(Y_0) = G(X)$ where X is the previously derived $G^{n-1}(Y_0)$.
(b) Form the running average of $G^n(Y_0)$ as $Y(n) = G^n(Y_0) + ((n-1)/n) \cdot Y(n-1)$.
(c) Test $|G^n(Y_0)_i - G^{n-1}(Y_0)_i| < \varepsilon_i$ for all components i. If this test succeeds for 5 consecutive values of n, the submodel passes the convergence test.
(d) If n is 500, do step (e); otherwise, go back to (a).
(e) Test convergence of average: if $|Y(500)_i - Y(499)_i| < \varepsilon_i$ for all components i, then do step (f); otherwise, submodel fails.
(f) Test requirement that average be a fixed point: if $|(G(Y(500))_i - Y(500)_i| < \varepsilon_i$ for all components i, then submodel passes the convergence test; otherwise, it fails.

When the test is applied, it is useful to store some statistical information about the submodels. The stored information includes: (1) the total number of submodels which have passed the convergence test; (2) the total number of submodels which have failed the $:Y(n+1) - Y(n):$ test (i.e., reached iteration 500 without passing the test); (3) the total number which failed the $:Y(n+1) - G(Y(n+1):$ test (i.e., passed $:Y(n+1) - Y(n):$ test and then immediately failed the $:Y(n+1) - G(Y(n+1)):$ test); (4) the total number of submodels which blew up somehow (e.g., some components of the vector $Y(n)$ got extremely large compared with the initial components of YO). In the first 2,000 submodels generated for the Klein–Goldberger model (Appendix A), only one submodel failed to pass test (3). Test (4) reported a blow-up condition when $|Y(n)|_i > 100{,}000\,|YO|_i$; for any component i.

More significant information was determined by storing the above information correlated with the number of submodels which contain equation K for each $k = 1, 2, \ldots, m$, where m is the number of equations of the full model. The print-out of these numbers made it possible to determine which equations of the full model were involved in events (1), (2), (3) and (4); and which equations are most likely misspecified according to Fisher's Correspondence principle. An example of this output for the Klein–Goldberger Model (Appendix A) is given in Figure 6. Note that equation 5 appears likely to be misspecified.

In the model tested, about half of the 2,000 submodels tested failed; and more models failed with $\varepsilon_i = 0.001\,|YO_i|$ than with $\varepsilon_i = 0.01|YO_i|$, which was expected. The test took 7.414 minutes of CPU time; if all 25,665 submodels had been tested, it would have taken an estimated 60 minutes of CPU time. An advantage of the algorithm as programmed is that the test may be stopped after a given number of submodels have been generated, and the relevant statistical information printed out. Useful results may be obtained without testing all submodels, although convergence of untested submodels cannot be guaranteed.

Another option of the program is the selection of submodels to be tested. The user can specify that only submodels which include a given subset of equation numbers will be generated and tested for convergence. Another option specifies that only submodels which do not include any of a given set of equation numbers will be generated and tested. Thus the algorithm facilitates the application of Fisher's Correspondence Principle to an econometric model and the discovery of misspecified equations.

Massachusetts Institute of Technology

Submitted May 1972
Revised September 1973

REFERENCES

[1] Fisher, Franklin M., "A Correspondence Principle for Simultaneous Equation Models," *Econometrica*, Vol. 38 No. 1 (January 1970), pp. 73–92.

[2] Purdom Jr., Paul, "A Transitive Closure Algorithm," *BIT*, No. 10 (1970), pp. 76–94.

[3] Tarjan, Robert, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, Vol. 1 No. 2 (June 1972), pp. 146–160.

[4] Tiernan, James C., "An Efficient Search Algorithm to Find the Elementary Circuits of a Graph," *Comm. A.C.M.*, Vol. 13 No. 12 (December 1970), pp. 722–726.

[5] Weinblatt, Herbert, "A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph," *Journal of the A.C.M.*, Vol. 19 No. 1 (January 1972), pp. 43–56.

APPENDIX A

The Klein–Goldberger Model as Used in Applying the Convergence Criterion

(1) $Z1 = A1 \cdot Y + CON1$
(2) $Z2 = A4 \cdot Y + CON2$
(3) $Z3 = A11 \cdot Y + CON3$
(4) $Z4 = CON4 \cdot X + CON5$
(5) $Z5 = A18 \cdot X - (A19 \cdot (Z9 \cdot Z6 \cdot Z7)/Z8) + CON6$
(6) $Z6 = (1/A24) \cdot (X - A22 \cdot Z3 - CON7 \cdot Z7 - A22 \cdot C21 - CON8)$
(7) $Z7 = CON9 \cdot Z9 - CON10 \cdot Z6 + CON11$
(8) $Z8 = A29 \cdot X + CON12$
(9) $Z9 = -A32 \cdot Z7 + CON13$
(10) $Z10 = (Z8/(Z9 \cdot Z6 \cdot Z7))(-Z18 + A38((Z9 \cdot Z6 \cdot Z7 \cdot Z11/Z8)$
$+ Z18 - Z14) + CON14)$
(11) $Z11 = PI - A41 \cdot X - (Z8/(Z9 \cdot Z6 \cdot Z7)) \cdot (Z18 + CON15)$
(12) $Z12 = (A44 \cdot (Z3 + C21)) + (Z8/(Z6 \cdot Z7 \cdot Z9)) \cdot A45 \cdot C24 + CON16)$
(13) $Z13 = C1 + (C2 \cdot X \cdot Z9 \cdot Z6 \cdot Z7/Z8)$

(14) $Z14 = C3 + (C4 \cdot Z11 \cdot Z9 \cdot Z6 \cdot Z7/Z8)$
(15) $Z15 = (CON17 \cdot Y \cdot Z9 \cdot Z6 \cdot Z7/Z8) + CT5$
(16) $Z16 = (X \cdot A57 \cdot Z9 \cdot Z6 \cdot Z7/Z8) + CON18$
(17) $Z17 = -C9 \cdot Z7 + CON19$
(18) $Z18 = (A55 \cdot Z6 \cdot Z9 \cdot Z7/Z8) + CON20$

The Z's are the endogeneous variables of the full model, while the X, Y, and PI are literal substitutions (identities) for the following expressions.

$$\begin{aligned}
X &= Z1 + Z2 + Z3 + Z4 - Z5 + C21 + FCON1 \\
Y &= Z1 + Z2 + Z3 + Z4 - Z5 + C21 + FCON1 - Z10 \\
&\quad + (Z8/(Z6 \cdot Z7 \cdot Z9)) \cdot (-Z13 - Z14 - Z15 - Z16 + Z17 - Z18 \\
&\quad - C23 + FCON2) \\
PI &= Z1 + Z2 + Z3 + Z4 - Z5 + C21 + FCON1 - Z8 - Z12 \\
&\quad + (Z8/(Z6 + Z7 + Z8)) - (-Z13 - Z16 - C23 + FCON3) \\
P &= (Z6 \cdot Z9/Z8)
\end{aligned}$$

All other symbols are constants (which includes lagged variables for the purpose of the convergence test). The correspondence between the Z's and the quantities they represent follows.

$Z1 = C$: consumption of durables, billions of 1954 dollars
$Z2 = C$: consumption of non-durables and services, billions of 1954 dollars
$Z3 = R$: residential construction, billions of 1954 dollars
$Z4 = H$: stock of inventories, billions of 1954 dollars
$Z5 = I_m$: imports, billions of 1954 dollars
$Z6 = h$: index of hours worked per week, 1954 = 1.00
$Z7 = N_w$: wage and salary workers, millions
$Z8 = W$: wages and salaries and supplements to wages and salaries, billions of 1954 dollars
$Z9 = w$: annual earnings, thousands of dollars
$Z10 = S_c$: corporate savings, billions of 1954 dollars
$Z11 = P_c$: corporate profits, billions of 1954 dollars
$Z12 = \pi_r$: rental income and net interest, billions of 1954 dollars
$Z13 = IT$: indirect taxes, billions of current dollars
$Z14 = T_c$: corporate profit taxes, billions of current dollars
$Z15 = PT$: personal taxes, billions of current dollars
$Z16 = BT$: business transfers, billions of current dollars
$Z17 = GT$: government transfers, billions of current dollars
$Z18 = IVA$: inventory valuation adjustment, billions of current dollars

The expressions of the following variables were substituted literally as they are identities of the model.

$X = X$: gross national product, billions of current dollars
$Y = Y$: personal disposable income, billions of 1954 dollars
$PI = PI$: proprietor's income, billions of 1954 dollars plus P and IVA
$P = Z6 \cdot Z7 - Z9/Z8$: implicit GNP deflator, 1954 = 1.00

The following variables depend solely on lagged or exogeneous variables and were not included in the incidence matrix of endogeneous variables.

r: yield on prime commercial paper, 4–6 months, percent
I: investment in plant and equipment, billions of 1954 dollars
D: capital consumption allowances, billions of current dollars
r: average yield on corporate bonds (Moody's), percent

The following variable was not included (except as a constant) because coefficients supplied in Fisher (1) caused it to be a constant.

SI: contributions for social insurance, billions of current dollars

The constants used in this model, as well as the initial values of both the endogeneous and exogeneous variable may be found in Fisher (1).

Appendix B

Incidence Matrix and V-Cycles of the Klein–Goldberger Model

Incidence Matrix

$$\begin{bmatrix}
0&1&1&1&1&0&0&0&0&1&0&0&1&1&1&0&1&1\\
1&0&1&1&1&0&0&0&0&1&0&0&1&1&1&0&1&1\\
1&1&0&1&1&0&0&0&0&1&0&0&1&1&1&0&1&1\\
1&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&1&1&1&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&0&1&0&0&1&0&0&0&1\\
1&1&1&1&1&1&1&1&1&0&0&1&1&0&0&1&0&1\\
0&0&1&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&0&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&0&0&1&1&0&0&1&1\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

The Primitive V-Cycles of the Graph of the Full Model

Primitive V-Cycles of 2 Vertices

(2, 13) (3, 15) (1, 2) (2, 5) (5, 6) (2, 15) (1, 4) (1, 13) (4, 5) (5, 8) (7, 9) (1, 5) (2, 3) (6, 7) (1, 15) (3, 4) (3,13) (1, 3) (2, 4) (3, 5)

Primitive V-Cycles of 3 Vertices

(1, 7, 15) (2, 10, 11) (1, 6, 18) (1, 11, 14) (3, 6, 13) (1, 8, 15) (1, 6, 10) (2, 6, 13) (3, 8, 13) (3, 6, 15) (2, 8, 13) (2, 6, 15) (3, 8, 15) (1, 6, 13) (2, 8, 15) (1, 8, 13)

Primitive V-Cycles of 4 Vertices

(1, 4, 10, 11) (1, 4, 8, 10) (2, 3, 6, 10) (1, 5, 8, 18) (1, 10, 11, 13) (2, 3, 8, 18) (3, 4, 6, 18) (1, 3, 6, 14) (3, 4, 11, 14) (2, 4, 6, 14) (1, 10, 11, 16) (3, 10, 11, 12) (3, 4, 8, 14) (3, 5, 6, 14) (1, 6, 7, 17) (3, 11, 13, 14) (1, 5, 10, 11) (1, 5, 8, 10) (2, 3, 8, 10) (3, 11, 14, 16) (3, 4, 6, 10) (2, 4, 6, 18) (1, 2, 6, 14) (3, 4, 8, 18) (3, 5, 6, 18) (1, 3, 8, 14) (2, 4, 11, 14) (3, 5, 11, 14) (1, 10, 11, 15) (2, 4, 8, 14) (2, 5, 6, 14) (3, 5, 8, 14) (2, 11, 13, 14) (3, 11, 14, 15) (1, 3, 8, 18) (2, 11, 15, 16) (3, 4, 10, 11) (2, 4, 6, 10) (3, 4, 8, 10) (3, 5, 6, 10) (2, 4, 8, 18) (2, 5, 6, 18) (1, 2, 8, 14) (3, 5, 8, 18) (3, 10, 11, 13) (1, 4, 6, 14) (2, 5, 11, 14) (2, 5, 8, 14) (3, 6, 7, 17) (1, 3, 10, 11) (2, 11, 14, 15) (1, 3, 8, 10) (1, 2, 8, 18) (3, 5, 10, 11) (2, 4, 8, 10) (2, 5, 6, 10) (3, 5, 8, 10) (2, 5, 8, 18) (1, 4, 8, 14) (1, 5, 6, 14) (2, 3, 6, 14) (3, 10, 11, 15) (2, 10, 11, 16) (2, 6, 7, 17) (1, 2, 8, 10) (1, 11, 14, 16) (3, 11, 12, 14) (1, 4, 8, 18) (2, 5, 8, 10) (2, 3, 6, 18) (2, 3, 11, 14) (1, 5, 8, 14) (2, 3, 8, 14) (3, 4, 6, 14)

Primitive V-Cycles of 5 Vertices

(3, 10, 11, 15, 16)