A TRANSPORTATION ALGORITHM AND CODE

Merrill M. Flood
The University of Michigan
Ann Arbor, Michigan

*Preprint Number 44
April 1960
First Draft

Engn
UMR
1484

A TRANSPORTATION ALGORITHM AND CODE

Merrill M. Flood
The University of Michigan

Introduction

The transportation problem and the assignment problem are two alternative names for the Hitchcock distribution problem. This problem has received considerable attention in recent years because of its importance in a wide variety of practical situations, and especially for industrial operations research applications.

We refer the interested reader to other sources[1] for a discussion of practical applications, and for further references to the many techniques that have been offered for solving this problem.

Solution techniques seem to fall into two broad classes. One class consists of variants of the Simplex Method of Dantzig[2], including the earliest work by Hitchcock[3] and Kantorovich[4]; also including some previous work by the present author[5]. The other class consists of variants of the Hungarian Method of Kuhn[6], including the present paper. Generally speaking, the Hungarian Method seems to be superior to the Simplex Method, for this special case of the general linear programming problem, but efforts to extend the idea of the Hungarian Method to the general linear programming problem have not produced computational techniques superior to the Simplex Method for the more general case.

The Hungarian Method, and the Hitchcock distribution problem, seem important enough to merit further development; the present paper discusses a new variant that has proved to be effective computationally.*

---

*Acknowledgement. I am very grateful to the IBM Research Center for supporting my work on the computer program for this algorithm, and for the use of the IBM 704 at the Center in testing the computational quality of the program. This work was done largely during two summers at the Center, while serving as a Consultant to IBM, including the period of the Combinatorial Problems Institute during July of 1959. I am especially indebted to V. V. Van Ness, at the Center, for programming advice and for generous help in running test problems on the Center's IBM 704.

Problem

The assignment problem is stated as follows:

Devise an efficient computational algorithm for permuting the rows of a given square matrix so as to minimize its trace.

If the rows (or columns) of the matrix are not distinct then this special case is known as the transportation problem, and is exactly the product distribution problem of Hitchcock[3] when stated in this form.

The problem may be rewritten in linear programming form as follows:

Find non-negative integers x_{ij}, subject to the restrictions

$$\sum_{j=1}^{n} x_{ij} = r_i \text{ and } \sum_{i=1}^{m} x_{ij} = c_j,$$

that minimize the quantity $\sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij}x_{ij}$; where r_i, c_j, d_{ij}, m, and n are given positive integers. Under this formulation, the problem is usually called an assignment problem if $r_i = c_j = 1$, otherwise it is usually called a transportation problem.

Hungarian Method

It is evident that the solution to the assignment problem is not changed by adding a constant to each element of a "line" of the matrix, where a line is either a row or a column. It is also evident that if the addition of such constants yields a matrix of non-negative elements, but such that the trace is zero after some permutation of the rows, then this permutation is the solution we seek. The Hungarian Method proceeds in this fashion, always keeping the matrix non-negative while increasing the number of null elements that can be brought to the main diagonal by a row permutation.

The mathematical basis for the Hungarian Method is provided by a constructive proof of the following theorem of König[7] and Ergerváry[8]:

König-Ergerváry Theorem

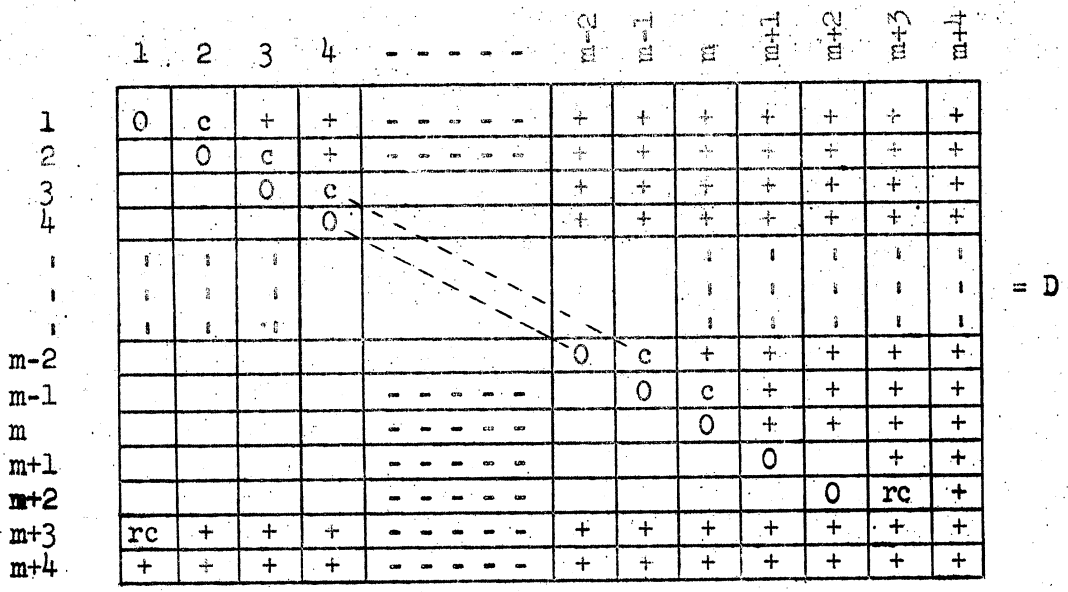
If D is a square array of two kinds of marks, say zeros and plusses, and if:

a) x is the maximum number of zeros that can be found in the array such that no two of them are in the same line, and

b) y is the minimum number of lines that can be found such that every zero of the array is contained in one of them,

then x = y.

The present author has shown elsewhere[2] how the rows and columns of a (0, +) matrix can be permuted into the following "standard" form.



The subarrays of the standard form have the following forms:

- ☐ denotes a subarray each of whose elements may be zero or plus,
- [0] denotes a square subarray all of whose main diagonal elements are zeros,
- [r] denotes a subarray with at least one zero in each row,
- [c] denotes a subarray with at least one zero in each column,
- [+] denotes a subarray with no zeros.

In this form, the zeros are all contained within the columns passing through D_{11}, D_{22}, ---, D_{mm} and the rows passing through $D_{m+1\ m+1}$ and $D_{m+2\ m+2}$. Consequently, if unity is added to each element of each of these lines, in turn,

the new matrix will have all elements positive; hence, unity can next be subtracted from every element of the matrix without producing any negative element. Since more was subtracted than was added, to the matrix as a whole, this process must eventually terminate; it does so when the zero elements fill the main diagonal in the standard form.

The Hungarian Method proceeds by two kinds of moves. One type of move involves finding a permutation that produces a standard form, so that the lines are identified for the additions to be made. The other type of move consists in adding and subtracting to this set of lines until no further such unit changes can be made.

Following Kuhn's[6] terminology, we shall say that two zeros are "independent" if they are not on the same line. Also, a set of lines is called a "cover" if every zero is on some line of the cover. A "minimal cover" is one containing the fewest lines. By the König-Ergerváry theorem, the maximal number of independent zeros is equal to the number of lines in a minimal cover. For further convenience, we shall call the number of elements in a set the "cardinal" of the set.

We shall use the phrase "trial set" to denote any set of independent zeros chosen to that every zero of the matrix is on line with at least one of the independent zeros of the trial set; a "trial zero" is any zero in a trial set. The phrase "trial cover" refers to a cover obtained using a specific trial set, and in the specific way described next.

Form the trial cover, for a given trial set, as follows:

1) Include in the trial cover each line containing a trial zero, that also includes a zero (called a "candidate") in some perpendicular line containing no trial zero. Consider these lines in the trial cover deleted from the matrix, and repeat this step until no further lines are added to the trial cover in this way.

2) Include in the trial cover each row containing a trial zero, in the matrix remaining after the deletions of step 1).

3) Include in a "candidate set" all candidate zeros noted during step 1).

It follows easily, by an inspection of the standard form, that the trial cover is a minimal cover if the trial set is a maximal set; also, if the cardinal of the trial cover exceeds the order of the trial set then the trial set is not maximal.

The Hungarian Method proceeds generally as follows:

a) Choose a trial set. If the order of this trial set is equal to the order of the matrix then the required permutation is the one that would permute the rows so as to place these zeros on the main diagonal. If not, go next to step b).

b) Consider the trial cover. If the cardinal of the trial cover is less than the order of the matrix then add a positive constant to all lines in the trial cover and subtract this constant from all lines of the matrix, choosing as this constant the largest integer that will leave all elements of the matrix non-negative; we call this constant the "fuse" for the cover and go next to step c). If the cardinal of the trial cover is not less that the order of the matrix, go next to step d).

c) Add to the trial set any new zeros that are eligible. If the cardinal of the new trial set is less than the order of the matrix then return to step b), otherwise the process ends.

d) The trial set is not maximal. Delete all zeros from the trial set that are on two lines of the minimal cover; such zeros are called "degenerates." Add to the trial set from the candidate set, forming a new trial set, using candidates on the paired lines (of the trial cover) that led to the deletion of zeros from the trial set at the start of step d). If the cardinal of the new trial set is less than the order of the matrix then return to step b), otherwise the process ends.

This brief description of the Hungarian Method, including one specific method for finding trial sets and trial covers, is intended only to indicate the general nature of the method. A more detailed and precise description of the steps actually used, in the variant presented in this paper, will be given in following sections; the variant actually used differs somewhat from the one described above, and choices left arbitrary above are made entirely definite in the computer code, but the differences are not significant for present purposes.

Initial Problem Preparation

It would be possible to solve an assignment problem, or a transportation problem, by use of the Hungarian Method from beginning to end. However, there are a number of improvements that can be made easily, and several of these are discussed in the present paper and used, to advantage, in the IBM 704 code. We first list some of these, and describe them briefly; then principal features will be illustrated by simple numerical examples.

Compression Subroutine. Lines of the matrix (row or column) having the same pattern of zero entries are combined. Except when determining fuses, compression leaves the procedures unchanged.

Reduction Subroutine. Each compressed line is made to have at least as many zeros as its frequency requires. Thus, if a compressed line includes χ of the original rows then it is made to have zeros in compressed columns that include at least χ of the original columns.

Isolation Subroutine. The trial set is chosen by first selecting "isolated" zeros that stand alone in some compressed line. One or both of the compressed lines through such an isolate are then deleted, after their zeros are exhausted, and new isolates are added to the trial set until no isolate remains.

Shortening Subroutine. If a compressed line has no more zeros than its frequency requires, then all these zeros are added to the trial set.

Zero Subroutine. A zero is chosen in a compressed line that contains zeros in the fewest perpendicular lines. When several such zeros exist, perhaps even in more than one compressed line, that one is chosen which has the most lines parallel to it through zeros in the compressed line perpendicular to it; if a tie still remains, the zero is chosen arbitrarily from among tied cases.

These Subroutines yield a trial set and also reduce the elements of the matrix.

Compression Example

	3	2	1	4	5
2	2	0	3	1	5
3	4	0	1	0	2
3	1	3	2	0	4
3	3	0	0	2	1
4	5	0	1	2	3

→

	8	2	1	4	
6	+	0	+	+	1, 5
3	+	0	+	0	
3	+	+	+	0	
3	+	0	0	+	
	1, 5				

The first and fifth rows, with frequencies 2 and 4, compress into a new first row with frequency 6; similarly the first and fifth columns compress.

Reduction Example

The first compressed row requires 6 zeros but has only 2; similarly the first compressed column requires 8 zeros and has none. The smallest amount that can be subtracted from the first compressed row, and yield a new zero, is 1 (the fuse); to retain the old zero in the first compressed row and second column, we must add 1 to the second column.

	3	2	1	4	5	
2	2	0	3	1	5	-1
3	4	0	1	0	2	
3	1	3	2	0	4	
3	3	0	0	2	1	
4	5	0	1	2	3	-1
		+1				

=

1	0	2	0	4	2
4	1	1	0	2	6
1	4	2	0	4	→ 3
3	1	0	2	1	3
4	0	0	1	2	

8	2	1	4	
+	0	+	0	
+	+	+	0	2,3
+	+	0	+	
+	0	0	+	
1,5				

Compressed Rows 2 and 3, and compressed Column 1, all require reduction; consider compressed Column 1 next, with fuse 1.

	3	2	1	4	5			3	2	1	4	5
2	1	0	2	0	4		2	0	0	2	0	3
3	4	1	1	0	2		3	3	1	1	0	1
3	1	4	2	0	4	=	3	0	4	2	0	3
3	3	1	0	2	1		3	2	1	0	2	0
4	4	0	0	1	2		4	3	0	0	1	1
	-1				-1							

Here, no compression is possible and only Row 5 and Column 5 require reduction; consider Row 5 next, with fuse 1.

	3	2	1	4	5				3	3	4	5	
2	0	0	2	0	3		0	1	3	0	3	5	0 + 0 + 1,3
3	3	1	1	0	1		3	2	2	0	1	3	+ + 0 +
3	0	4	2	0	3	=	0	5	3	0	3	→ 3	+ + + 0
3	2	1	0	2	0		2	2	1	2	0	4	+ 0 0 0
4	3	0	0	1	1	-1	2	0	0	0	0		2,3
		+1	+1										

No further compression is possible, so the reduction is complete.

Isolation Subroutine

The trial set is obtained by choosing isolated zeros (in the compressed matrix) as shown next, where the entry replacing the zero is the frequency for that trial zero and the subscripts indicate the order in which the isolated zeros were chosen.

	0	0	4	2			0	0	0	1
2	3_2	+	0	+		0	3_2	+	2_4	+
3	+	+	0	+		1	+	+	2_5	+
0	+	+	+	3_1	=	0	+	+	+	3_1
1	+	3_3	0	0		0	+	3_3	0	1_6

Note that lines are treated as though they were deleted after their frequency goes to 0.

This example does not stress the use of recompression during our choice of the trial set, but recompression normally occurs several times during the process as lines are deleted.

Shortening Subroutine

When the reduced and compressed matrix reaches a stage where no further isolated zeros are found, perhaps after several deletions of lines during construction of a trial set, then the Shortening Subroutine may add zeros to the trial set, as indicated for the following example.

	5	2	3			0	0	0
5	+	0	0		0	+	2	3
2	0	0	+	=	0	2	0	+
3	0	+	0		0	3	+	0

Zero Subroutine

When reduced and compressed matrix reaches a stage where further application of the Isolation and Shortening Subroutines yield no further trial zeros, then the Zero Subroutine takes effect. The following example illustrates this process.

	2	3	4			0	3	4
5	+	0	0		5	+	0	0
2	0	0	+	=	2	0	0	+
2	0	+	0		0	2	+	0

The trial zero in Row 3 and Column 1 is chosen because it is in the line having fewest zeros, namely 4 in Column 1, and is preferred over the zero in Row 2 because Row 3 has more zeros, with 6, than does Row 2, with only 5.

Fuse Rule. The Reduction Subroutine is usually improved if each uncompressed line is first reduced enough in one step to yield the required

number of zeros; this will sometimes mean the use of more than one fuse in a single step. The fuses are selected according to the "Fuse Rule," which states simply that just enough fuses are selected, in increasing order of numerical size, to yield the required number of zeros for the line being reduced. The following example illustrates application of the Fuse Rule.

	3	2	1	4	5				3	2	1	4	5
2	1	0	2	0	4			2	0	0	2	0	2
3	4	1	1	0	2			3	3	1	1	0	0
3	1	4	2	0	4		=	3	0	4	2	0	2
3	3	1	0	2	1	+1		3	3	2	1	3	0
4	4	0	0	1	2			4	3	0	0	1	0
	-1				-2								

The fuse for Column 1 is 1, because Rows 1 and 3 yield 5 zeros to meet the requirement of 3. The fuse for Column 5 is 2, because Row 4 (and fuse = 1) does not yield the 5 zeros required for Column 5; the next larger entry in Column 5 is 2, and this becomes the actual fuse used when the Fuse Rule is applied to Column 5. Actually, for this example, our reduction using the Fuse Rule has completed the problem, except for finding the trial set which then proves to be a solution.

Many problems are completely solved by the first trial set, after the initial preparation. When they are not completely solved, the reduction in the matrix is large enough to make a substantial cut in computational time required to complete the solution by some other procedure. For example, the Simplex Method could be used after this initial preparation with the trial set a part of the first trial basis. We shall return next to the Hungarian Method, and illustrate the use of the trial cover by a numerical example.

Trial Cover Example

We use the Isolation Subroutine example to illustrate the procedure for forming a trial cover. The trial set is shown as numerical entries in the compressed matrix, where a 0 denotes a zero in the compressed matrix that is not in the trial set.

	3	3	4	5
5	3	+	2	+
3	+	+	2	+
3	+	+	+	3
4	+	3	0	1

We first note that the trial zeros in positions (3, 4) and (4, 4) are candidates, and this leads immediately to the inclusion of compressed Rows 3 and 4 in the trial cover; this is so because compressed Column 4 includes five lines but only four trial zeros. Similarly, the trial zero in position (2, 3) is a candidate, and places Column 3 in the trial cover. After deleting compressed Rows 3 and 4, and compressed Column 3, the following array remains.

	3	3	5
5	3	+	+
3	+	+	+

In this array, the zero in position (1, 1) is a candidate, and the first column is added to the trial cover; thus, compressed Column 1 of the original matrix is in the trial cover. No zeros remain, after deleting Column 1, so the trial cover is complete; it consists of compressed lines (R3, R4, C1, C3). Note that the cardinal of the trial cover is 14, and is equal to the cardinal of the trial set, so it follows that the trial set is maximal.

Our next example shows how the trial cover is formed when the trial set is not maximal.

	3	3	4	5
5	3*	+	0*	+
3	+	+	0*	+
3	+	+	+	3*
4	+	0*	4	0*

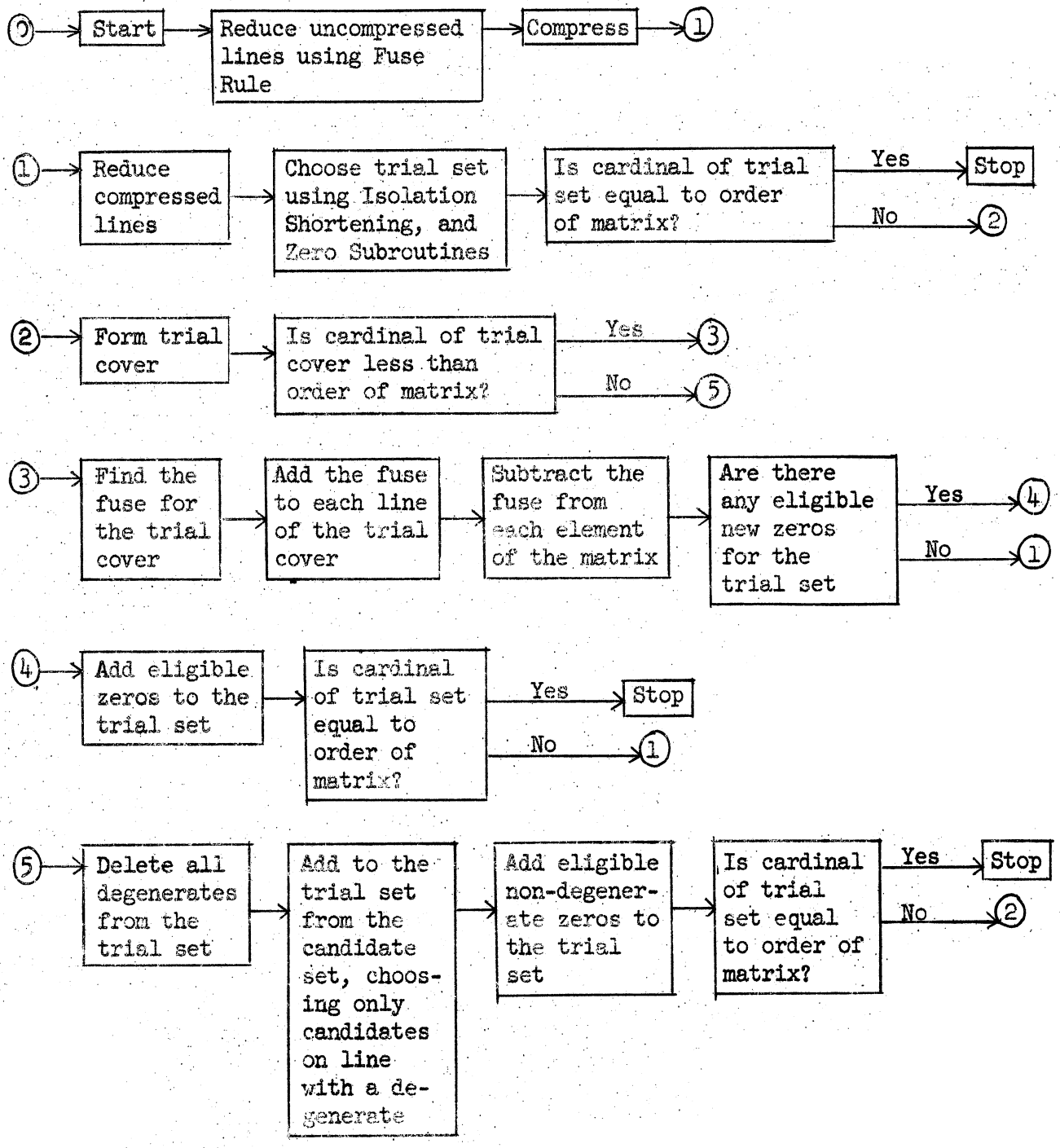
The trial cover includes compressed lines (R3, R4, C1, C3), with cardinal 14, while the cardinal of the trial set is 10. The candidate zeros are marked by asterisks. The zero in position (4, 3) must be removed from the trial set because it lies in intersecting paired lines of the trial cover; after its removal the trial set is increased by using candidate zeros in lines R4 and C3.

FORTRAN Codes

Two separate FORTRAN II codes were written and tested, based on the general principles of the algorithm. The earlier of these, called "CODE I," used recompression at intermediate stages of Step 1B but the later one, called "CODE II," did not. There were other important differences between CODES I and II, and neither code proceeds exactly as in the flow diagram for the algorithm We shall discuss only CODE II in the present paper; results with CODE I were reported previously[10].

CODE II was written to include provision for measuring the computing time required, in any particular run, for each of several stages within the whole run. This was made possible by the use of an internal clock that could be read and recorded, at any point during the calculation, when required by the code. Provision was made for reading the clock at 12 such points in the code, including 7 clock readings at points normally passed more than once during a problem; these repeated clock readings can be made the first 20 times the point is reached, or fewer at each point if desired. Clock readings may also be taken at the start and end of the computing cycle on a problem, and are always taken at the beginning and end of work on a problem.

Algorithm Flow Diagram



CODE II also provides for a count of the number of times each of 17 points is reached during the calculation of each problem.

CODE II includes a subroutine that may be called in to compose a problem of any desired size, up to the dimensional limit of 60x150 for the cost matrix. The cost matrix, and the row and column frequencies, are generated by the subroutine using a pseudorandom number technique.

Although CODE II was written and compiled to handle a transportation problem having at most 60 rows and 150 columns, it would be quite a simple matter to recompile the code to handle any problem having fewer than 24,000 elements in the cost matrix.

Other special features of CODE II, that will not be discussed in any detail in this paper, include provision for selecting among a few alternative algorithms and several options on control of output data.

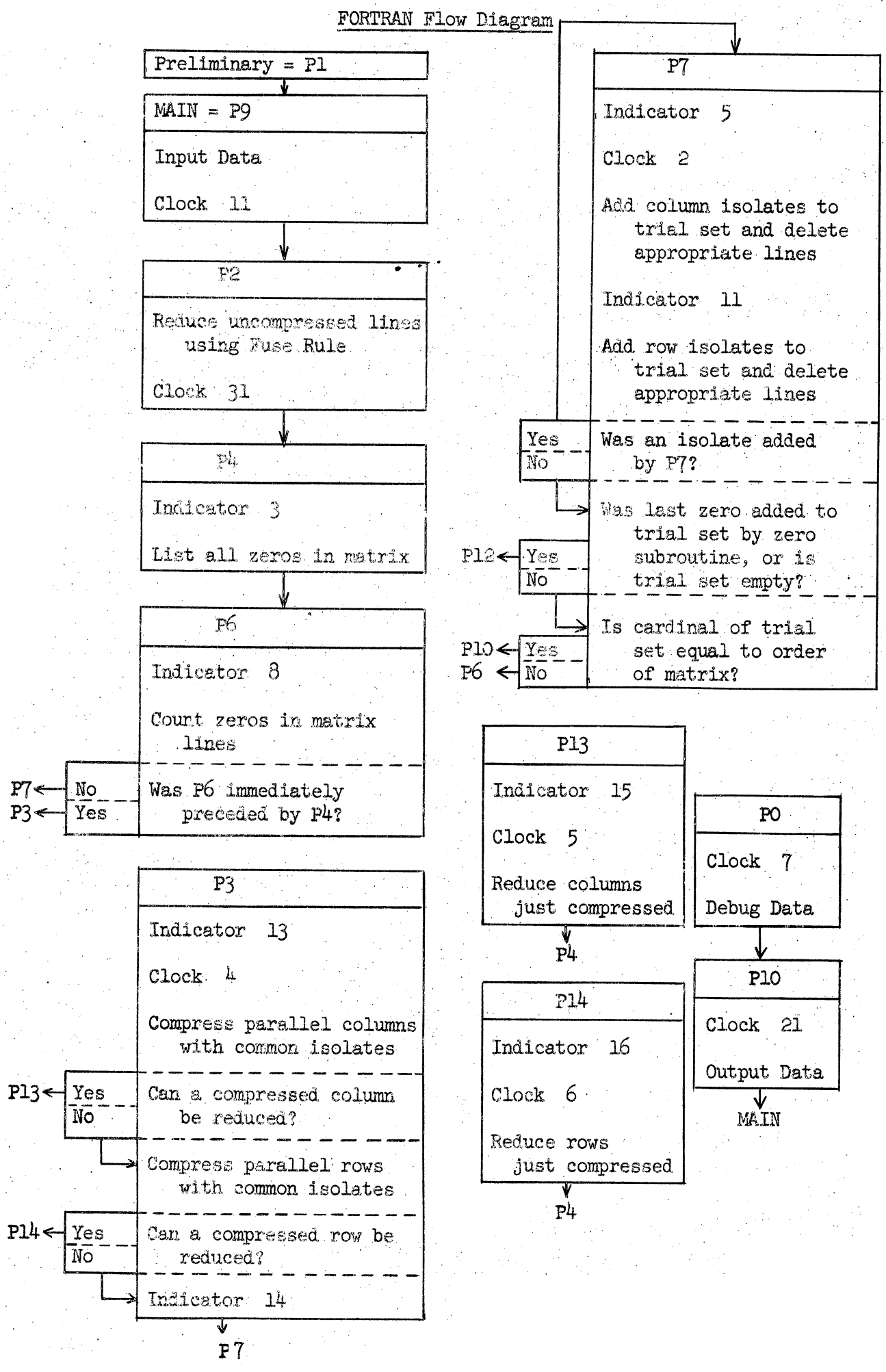
We next present a condensed flow diagram for the CODE II FORTRAN Subroutines, and show where CLOCK and INDICATOR readings are taken during a computation. This will be useful for our subsequent discussion of times and frequencies through various computing stages of actual problems run.

FORTRAN Flow Diagram

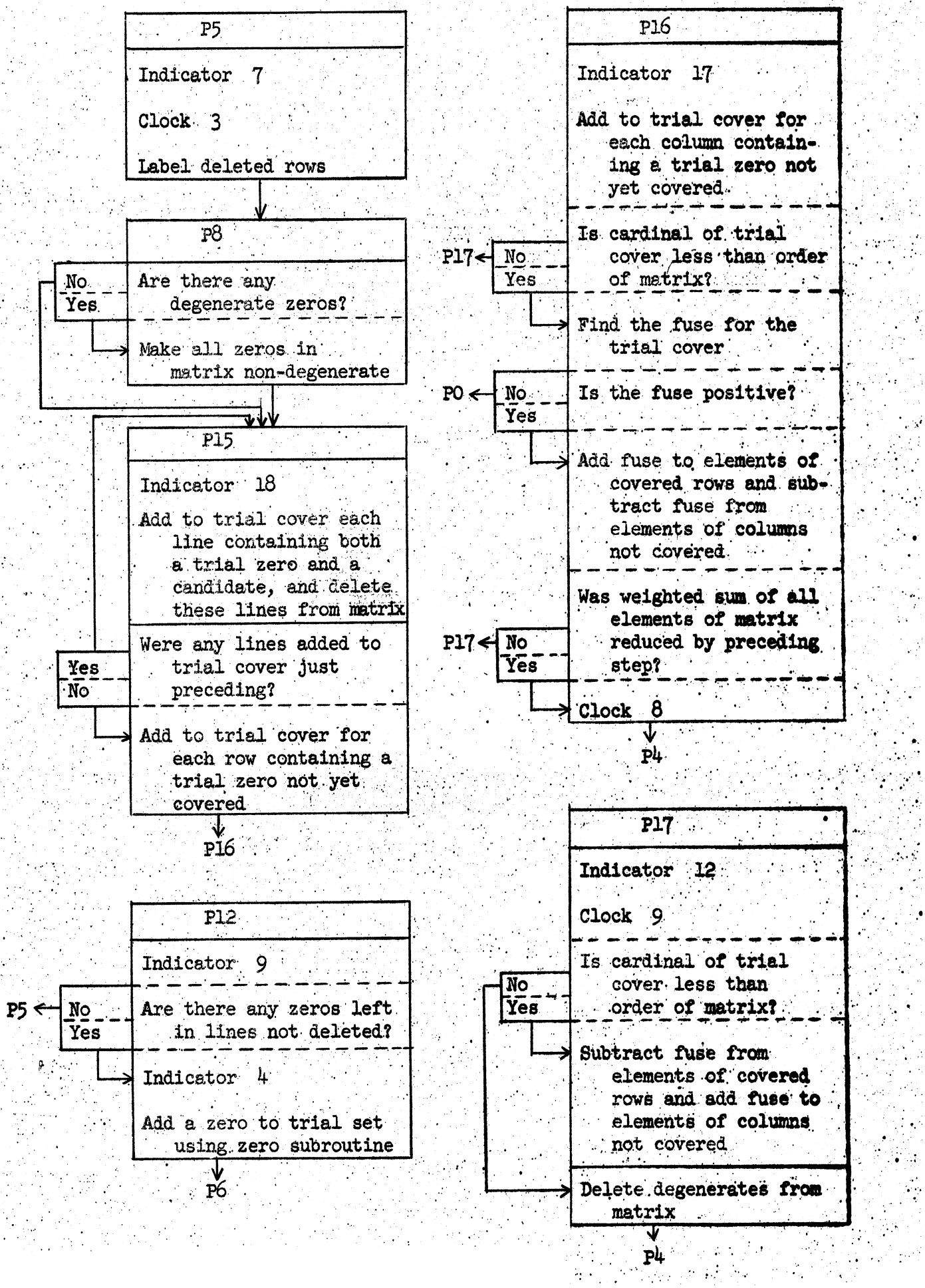
The name of the FORTRAN II Subroutine appears first in each box of the flow diagram. CLOCK X entries show the points during the computation at which clock readings may be taken. INDICATOR Y entries show the points during the computation at which counts are made of the frequencies of passing these points.

CODE II is represented correctly by this FORTRAN Flow Diagram, but differs in several respects from the Algorithm Flow Diagram. For example, the matrix is not recompressed at each step in CODE II that is indicated by the Algorithm Flow Diagram, and several other important changes have been made in CODE II to facilitate the use of FORTRAN.

FORTRAN Flow Diagram



FORTRAN Flow Diagram



There is also a coding error in CODE II, but this error results only in stopping a few problems before the computation is entirely completed. CODE II is arranged so that this case results in a special output, through Subroutine PO, that provides the data necessary to complete the problem easily. This output, like several others included in CODE II, automatically indicates the point of error whether due to machine difficulty or some other unexpected trouble. These possible sources of error will be ignored throughout the rest of our discussion, since they are rare and recognizable.

Computational Results

We present our computation results first in terms of CLOCK times. For example, T ≡ CLOCK 21-CLOCK 11 represents actual total computing time; if SUBROUTINE PO is used, then actual total computing time is T ≡ CLOCK 7-CLOCK 11. As another example, T_{83} ≡ CLOCK 8-CLOCK 3 represents actual computing time used in going once through SUBROUTINES P5, P8, P15, P16 if CLOCK 9 was not read meanwhile. This latter example illustrates one way in which individual CLOCK readings may be used to trace cycle times in various portions of the complete run. Similarly, if CLOCK 4 follows CLOCK 9 (or CLOCK 5, 6, 8 or 31) then SUBROUTINE P3 (rather than SUBROUTINE P7) followed SUBROUTINE P6; in this general manner the computing time taken for various segments of the run can be determined.

CODE II is written so that intermediate outputs can be obtained during a run, in very many different ways, but this feature will not be discussed in the present paper. Suffice it to say that this feature permits the use of INDICATOR readings, in conjunction with CLOCK readings, to follow the path and timing of computational steps in considerable detail when appropriate intermediate outputs are available.

We turn now to CLOCK results on one particular problem (#E116.1), in order to show the kind of timing data obtained. This is the only problem that

has been computed also by a competing code, called IB-TFL, perhaps the fastest standard IBM 704 code (SHARE 464)* available for the transportation problem. This problem was a 29x116 pseudorandom transportation problem that took 3.03 minutes with CODE II, in comparison with 3.17 minutes with IB-TFL; this difference is so small that the two codes may be considered equally efficient for this one 29x116 problem. On a second pseudorandom problem (#E116.2), of size 29x116, CODE II took 1.90 minutes.

The beginning of the flow through #E116.1 may be followed by noting CLOCK readings, as follows:

Time	0	24	25	25	26	26	28	28	30	31	33	34	36	37	39	40	etc.
CLOCK	11	31	4	5	4	5	4	5	4	5	4	5	4	5	4	5	
SUB-ROUTINE	P9	P2	P3	P13	P3	P13	P3	P13	P3	P13	P3	P13	P3	P13	P3	P13	

Time	58	60	60	60	60	60	60	65	70	72	72	etc.	73	79	88	95	etc.
CLOCK	2	2	2	2	2	2	3	8	2	2	2		3	8	3	8	
SUB-ROUTINE	P7	P7	P7	P7	P7	P7	P5	P16	P7	P7	P7		P5	P16	P5	P16	

In other words, the initial uncompressed reduction through P2 took 0.24 minutes. Then the P4→P6→P3→P13 cycle was repeated 9 times in 0.34 minutes, to complete the compressed isolate columns reduction. Next the P7 cycle was repeated 6 times, in 0.02 minutes, to list the isolated zeros. Next the P12→P5→P8→P15→P16 sequence took 0.05 minutes to find a trial cover, fuse, and reduction before returning to P4→P6→P7. Since we had arranged to have only 9 readings on CLOCK 2, we cannot trace the path through all the subroutines after .72 minutes. We do obtain the first 10 times at which CLOCKS 3 and 8 are read and, from these data, it is seen that: the P5→P8→P15→P16 portion takes about 0.06 minutes each

*SHARE 464 is a code based upon the Ford-Fulkerson[11] network flow algorithm, which is another variant of Kuhn's Hungarian Method.

time, and the portion from P16 back to P5 through P4→P6→P7→P12 takes about 0.09 minutes each time.

The Indicators not only show frequencies with which each of several points are passed during the computation, but also may be used to assist in tracing the actual path of the computation through the run. The Indicator readings for problem #E116.1 follow.

Indicator #	3	8	13	14	15	16	5	9	4	7	18	12
Frequency	27	42	27	16	11	0	99	15	0	15	79	0
Subroutine	P4	P6	P3		P13	P14	P7	P12		P5	P15	P17

For example, since I12 = I16 = 0 it is seen that P14 and P17 were not used; also since I4 = 0, the segment P12→P6 was never used. By continuing in this way, we find that the paths used were as follows.

Indicator:	3	5	7	8	13	14	15	18	Stop
Start	1								
3				27					
5		68	15	15					1
7								15	
8		15			27				
13						16	11		
14		16							
15	11								
18	15							64	

This shows that Subroutine P7 was entered 99 times; of these, 15 led to P12, 15 to P6, 68 back through P7, and 1 to Stop. It shows also that Subroutine P7 was entered 15 times from P6, 16 from P3, and 68 from itself.

Other pseudorandom problems were generated and solved by CODE II, with results as follows.

Dimension	Number Solved	Median Computing Time	Smallest Computing Time	Largest Computing Time	Input Output Time
		← In minutes →			
58x145	2	13.56*	12.72*	14.41*	?
29x116	2	2.47	1.90	3.03	.18
29x29	3	.53	.35	.82	.14
20x29	13	.25	.18	.36	.14
9x10	7	.45**	.13**	.65**	?

*Actually, these problems were not quite completed, but the output is adequate for easy completion.
**All times too high because intermediate outputs were obtained.

Several other small problems were run, for various tests, each of which was constructed for some particular purpose. Among these were one 9x9 assignment problem (unit rim totals) and 30 5x15 assignment problems (unit column rim totals). The 30 5x15 assignment problems each took less than 0.02 minutes computing time, except for one that was not quite completed--due to the same error in CODE II that kept the 58x145 problems from finishing. The 9x9 assignment problem was constructed carefully to stretch CODE II, and did in fact require 0.26 minutes computing plus system time.

Test runs with the recompression subroutines (P3, P13, P14) omitted in CODE II have shown that their inclusion shortens computation times appreciably for larger problems. Experience with CODE I, where complete recompression was done at each step rather than recompression only of lines having common isolates as in CODE II, indicates the desirability of more extensive recompression than is used in CODE II; however, the FORTRAN techniques used in CODE I are not effective in accomplishing more extensive recompression. Comparative tests of other alternatives have suggested various improvements that can easily be made in CODE II, but without deviating from the essential steps of our mathematical algorithm, and data on segmental computing times have helped in choosing among some of these alternatives.

Conclusions

CODE II makes effective use of the Hungarian Method for solving transportation and assignment problems on the IBM 704. Computing experience indicates that CODE II is now as fast as the IB-TFL code, or any other existing code, and CODE II would surely be improved further if rewritten to gain effeciency and speed.

The mathematical algorithm supporting CODE II makes essential use of several reduction steps, preliminary to the execution of zero covering and selection subroutines based on the present author's proof of the König-Ergervary Theorem. These preliminary reductions should be useful with other codes for the Hitchcock distribution problem, whether based upon the Hungarian Method, the Simplex Method, or any other method.

The techniques employed in CODE II, for observing timing and frequencies of passages through segments of a particular computation, are very useful in determining choices among alternative codes. Experience with CODE II has proveded timing and frequency data that shows where further substantial improvements may possibly be made in CODE II, while holding to the same mathematical algorithm.

References

1a. C. W. Churchman, R. L. Ackoff and E. L. Arnoff. Introduction to Operations Research, Wiley (1957), Chapters 11 and 12.

b. Merrill M. Flood, "The Traveling Salesman Problem," Operations Research, Vol. 4, No. 1, February 1956, pp. 61-75.

2. George B. Dantzig. "Application of the Simplex Method to a Transportation Problem." In: T. C. Koopmans (Ed.), Activity Analysis of Production and Allocation, Wiley (1951).

3. Frank L. Hitchcock. "The Distribution of a Product from Several Sources to Numerous Localities," Journal of Mathematics and Physics, Vol. 20, (1941), pp. 224-230.

4. L. Kantorovitch. "On the Translocation of Masses," Management Science, Vol. 5, No. 1, October 1958, pp. 1-4.

5. Merrill M. Flood. "On the Hitchcock Distribution Problem," Pacific Journal of Mathematics, Vol. 3, No. 2, June 1953, pp. 369-386.

6. H. W. Kuhn. "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, Vol. 2, Nos. 1, 2, March-June 1955, pp. 83-98.

7. Denes König. "Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre," Mathematisches Annalen, Vol. 77, (1916), pp. 453-465.

8. J. Egerváry. "Matrixok Kombinatorius Tulajdonsagairol," Mathematikai és Fizikai Lapok, Vol. 38, (1931), pp. 16-28.

9. Merrill M. Flood. "An Alternative Proof of a Theorem of König as an Algorithm for the Hitchcock Distribution Problem." In: R. Bellman (Ed.), Proceedings of the Symposia in Applied Mathematics, Vol. 10 (1960), to be published by the American Mathematical Society.

10. Merrill M. Flood. "A Transportation Code." In: Philip Wolfe (Ed.), The RAND Symposium on Mathematical Programming, The RAND Corporation (1960), pp. 79-80 (abstract).

11. L. R. Ford, Jr. and D. R. Fulkerson. "Solving the Transportation Problem," Management Science, Vol. 3, No. 1, October 1956, pp. 24-32.