MANAGEMENT SCIENCE
Vol. 20, No. 5, January 1974
Printed in U.S.A.

NETGEN: A PROGRAM FOR GENERATING LARGE SCALE CAPACITATED ASSIGNMENT, TRANSPORTATION, AND MINIMUM COST FLOW NETWORK PROBLEMS*

D. KLINGMAN,† A. NAPIER§ AND J. STUTZ¶

The purpose of this paper is to describe the development, implementation, and availability of a computer program for generating a variety of feasible network problems together with a set of benchmarked problems derived from it. The code "NETGEN" can generate capacitated and uncapacitated transportation and minimum cost flow network problems, and assignment problems. In addition to generating structurally different classes of network problems the code permits the user to vary structural characteristics within a class.

Problems benchmarked on several codes currently available are provided in this paper since NETGEN will also allow other researchers to generate identical problems. In particular, the latter part of the paper contains the solution time and objective function value of 40 assignment, transportation, and network problems varying in size from 200 nodes to 8,000 nodes and from 1,300 arcs to 35,000 arcs.

More Problems—Who Needs 'Em?

Since the earliest computational experiments with network computer codes both users and developers have been faced with the necessity of evaluating such codes as to their capacity, speed, effectiveness (with respect to problem structure), and accuracy. Initially most efforts were directed toward determining solution accuracy, however, more recently attention has been focused on the other attributes. This is due in large part to the availability of competing algorithms ([1], [2], [3], [6], [8], [10], [11], [12], [13], [14], [15], [16], [17], [19], [20]), the development of larger network models, the increased use of network models in government and industrial applications, and new techniques for converting problems which otherwise appear to be unamenable to network formulation. Current computational studies ([1], [2], [13], [14], [16], [20]) have concentrated on problem size and solution time (relative to a particular class of networks) because of the interest of potential users to minimize their computer costs.[1] Also some models are not being implemented in industry due to the prohibitive solution time and lack of computer codes to handle extremely large problems.[2] Further interest

* Received March 1973; revised May 1973.

† Associate Professor of Operations Research at the University of Texas, Austin, Texas.

§ Research Associate, Operations Research Division, Continental Oil Company, Houston, Texas.

¶ Assistant Professor of Operations Research at the University of Texas, Austin, Texas.

[1] Network codes and computational studies which are currently in progress include:

(a) a new out-of-kilter approach by M. Florian, Department of Information, Universite de Montreal;

(b) a special purpose primal simplex network code by G. W. Graves, and R. McBride, Graduate School of Management, University of California at Los Angeles;

(c) a minimum cost flow network code (the exact methodology to be employed is not known) by Burroughs Corporation;

(d) a new out-of-kilter approach by the Texas Water Development Board;

(e) a code for solving transportation problems with concave cost functions by R. Soland, Graduate School of Business, University of Texas at Austin;

(f) a computational study on a primal simplex network code by Glover, Karney, and Klingman, Graduate School of Business, University of Texas at Austin.

[2] To illustrate, Paul Randolph at New Mexico State University, in conjunction with the Department of Agriculture has developed a 3000 origin by 1200 destination transportation model with 2,478,000 admissible cells and twenty fixed charge variables for scheduling cotton to gins.

in reducing solution time is stimulated by the fact that network problems often occur as subproblems in larger problems such as warehouse location problems, multi-commodity network problems, fixed charge transportation problems, and constrained transportation problems.

The problem of adequately "benchmarking" even the most thoroughly debugged codes arises, of course, in a variety of applications of computers to mathematical and scientific problems. However, many network problems involve quite large node-arc incidence matrices (say 1500 by 10,000; that is, 1500 nodes and 10,000 arcs). Consequently, the data handling and generation of test problems become severe even for problems with well-known topological characteristics. For this reason, tasks like the comparison of performance of network codes must be carried out with the raw material (i.e., test problems) at hand.

The purpose of this paper is to describe the development, implementation, and availability[3] of a computer program for generating a variety of feasible large scale L.P. problems which are generally termed network problems. Among the most frequently discussed network problems which the code can generate are capacitated and uncapacitated transportation and minimum cost flow (pure) network problems; as well as the simpler forms such as assignment, shortest path, and maximum flow problems. In addition to generating structurally different classes of network problems the code permits the user to vary structural characteristics within a class. The user controls the size of the problem as well as various parameters. In particular, the user controls the size by specifying the number of pure sources (origins), pure sinks (destinations), transshipment nodes with supply (transshipment source nodes), transshipment nodes with demand (transshipment sink nodes), transshipment nodes with no supply or demand (pure transshipment nodes), and the total number of arcs (cells) in the problem. The user has additional control of the structure through the use of such parameters as: total supply, cost range, upper capacity range (the lower capacity of an arc is always zero), percentage of arcs to be capacitated, and another parameter which implicitly controls solution difficulty. Another feature of the program is the inclusion of a random number generator ([4], [22]) with a user supplied seed. This feature allows the code to regenerate the same problem if every input parameter is the same. Since other researchers can generate identical networks, a set of problems benchmarked on several currently available codes is provided in this paper.

Perhaps one of the most useful applications of our code is that of measuring the solution time and accuracy of some well-known and widely used network codes when employed to solve very large problems (e.g., the code could easily be used to create 10,000 node problems of any desired arc density). Also, the availability, in quantity, of a meaningful variety of test problems, may help to influence the implementation of new solution techniques for network problems. All too often with new algorithms, an elegant theory has been a substitute for, rather than gone hand-in-hand with, effective performance in practice.

Some other advantages of having such a code available are:

1. To permit codes developed for a more general class of problems to be easily tested on special subclasses. For instance, codes developed to solve general minimum cost flow network problems could also be easily benchmarked on transportation and assignment problems, thus providing ways to evaluate the relative worth of the more

[3] A complete description of the generator and user documentation is available from the authors. Also, a card deck is available for a nominal handling charge.

specialized codes. To illustrate, numerous algorithms exist for solving transportation problems and minimum cost flow network problems. Theoretically these problems are equivalent since any minimum cost flow network problem can be reformulated as a transportation problem. However, the question arises: "Is it worth developing and maintaining separate codes for each of these problem types, or should only one of the codes be developed? If only one, which type should be developed?" From a theoretical standpoint, the O.R. literature reflects the feeling that both types of algorithms should be pursued. Similar questions are relevant between algorithms for these problems and assignment problems or maximum flow problems.

2. To encourage standardization of data specification for all types of network problems. One of the problems we encountered in trying to benchmark codes based on different methodologies (e.g., codes based on circulation networks such as out-of-kilter ([2], [3], [17]) and codes based on the simplex method ([6], [12], [13], [14], [20])) and codes designed to solve different types of problems was their lack of uniformity for input specification. This nonstandardization of problem specification (in terms of input format) is most frustrating and has hampered benchmarking since researchers are reluctant to recode their input routines. Thus it is essential to establish a standard way of specifying all types of network problems as well as network problems within a class. In order to promote this standardization with minimum user inconvenience, we use a network specification which is compatible with SHARE ([3], [17]) out-of-kilter since this is probably the most widely used network format.

3. To facilitate computational studies on the effect of parameter variation—such as changing the cost range, total supply, percentage of arcs capacitated, number of arcs, and capacity range, etc.

The Creation Process

With these thoughts in mind we turn our attention to a more specific description of the methodology and other salient features of the generator. Having read the input parameters the size, type, and characteristics of the problem are fixed. The program then creates a network problem within this framework.

The overall program can be divided into two main parts. The first part creates what is called a skeleton network and is concerned with obtaining the proper number of nodes of each type, insuring the correct total supply, and guaranteeing that the resulting problem will be connected and feasible. The second part completes the problem while insuring that the remaining specifications are met, such as total number of arcs, cost range, upper bound range, and percentage of arcs capacitated.

First, all nodes are given a number (integer) between one and the number of nodes, and the nodes are grouped into sets by type (i.e., pure source, transshipment source, pure transshipment, pure sink, and transshipment sink). During this part of the program, transshipment sources and sinks are treated as pure sources and sinks, respectively. The total supply is then randomly distributed among the sources using random numbers from a uniform probability distribution ([4], [22]).

Next, a chain is randomly created from each source node by generating arcs involving the source node and a random number of pure transshipment nodes. These chains are pairwise disjoint and mutually exhaustive of all nodes except sinks. In each chain there is a single directed path from the source node to every pure transshipment node in the chain.

After the chains have been created, each chain is connected to a random number of randomly chosen sinks. Simultaneously, the demand amounts are accumulated for each sink by randomly distributing the supply of the unique source among the sinks connected to the chain.

In the case of transportation and assignment problems, since they contain no pure transshipment nodes, each chain contains only a source (origin) node. For transportation problems the origins are connected directly to a random number of sinks (destinations) and the demands are created by randomly distributing the supply. For assignment problems each origin is randomly connected to a unique destination and each origin is given a supply of one and each destination is given a demand of one.

At this point the network has the correct number of sources, sinks, transshipment nodes, and total supply. Also the network is guaranteed to be connected and feasible (without capacities). This partial network is called the *skeleton* and its generation completes the first part of the program. Observe that the skeleton will only contain a few more arcs than is required to have a connected graph. Thus, one of the major attributes of this procedure is its ability to create networks of extremely low density.

The second part of the program begins by determining the costs and capacities for the arcs in the skeleton. Using the percentage of arcs to be capacitated (supplied in the input), certain arcs in the skeleton are randomly chosen to be capacitated. The upper capacity of each randomly chosen arc is set equal to the larger of the supply of the unique source of the chain in which the arc appears and the user supplied minimum upper bound. The remaining arcs in the skeleton are left uncapacitated. (Note that the capacity of arcs in the skeleton may be higher than the largest upper capacity supplied by the user.) A percentage (one of the input parameters) of the arcs in the skeleton (the specific ones are chosen randomly) are assigned the maximum cost. Other arcs are assigned a cost randomly chosen between the lower and upper limits. The flexibility to set the costs of a percentage of the skeleton arcs large is intended to discourage the use of these arcs in an optimal solution, thus, (possibly) making the network more difficult to solve.

Once the skeleton is complete, the problem is guaranteed to be feasible regardless of the number, location, or characteristics of any additional arcs. Therefore, all that remains is to randomly generate and distribute the required number of additional arcs. (It is during this phase of the program that arcs are permitted to emanate from a transshipment sink and terminate at a transshipment source). For each node, except pure sink nodes, a random number of additional arcs are created from this node to other randomly selected nodes, while insuring that no duplicate arcs are created. The number of nodes to which a given node is connected, is randomly chosen from a range selected such that the final network will contain approximately the correct number of arcs. During this process a cost is randomly chosen (within the proper range) for each arc created and the given percentage of them are given an upper capacity randomly chosen within its range. This essentially completes the network except for adding a super-source (numbered one larger than the total number of nodes), and a super-sink (numbered two larger than the total number of nodes) in order to circularize the network. (The nodes and the arcs added for the purpose of creating a circulation network are not counted in the total number of nodes and arcs.)

One of the more salient features of the code and the one chiefly responsible for its ability to create large networks is the way the network is stored. The skeleton is the only part of the network stored in core. Further, it is stored as a set of linked lists, each list corresponding to a chain. The demand for each sink and all of the arcs in the skeleton, except for the arcs connecting the chains to the sinks, are contained in one node-length array. (All additional arcs are output to the problem file as they are cre-

TABLE 1
Problem Specifications

Number of Nodes	Number of Sources	Number of Sinks	Number of Arcs	Cost Range		Total Supply	Transshipments		Percent of High Cost	Percent of Arcs Capacitated	Upper Bound Range		Random No. Seed
				Min	Max		Sources	Sinks			Min	Max	
1. 200	100	100	1300	1	100	100,000	0	0	0	0	0	0	13502460
2. 200	100	100	1500	1	100	100,000	0	0	0	0	0	0	13502460
3. 200	100	100	2000	1	100	100,000	0	0	0	0	0	0	13502460
4. 200	100	100	2200	1	100	100,000	0	0	0	0	0	0	13502460
5. 200	100	100	2900	1	100	100,000	0	0	0	0	0	0	13502460
6. 300	150	150	3150	1	100	150,000	0	0	0	0	0	0	13502460
7. 300	150	150	4500	1	100	150,000	0	0	0	0	0	0	13502460
8. 300	150	150	5155	1	100	150,000	0	0	0	0	0	0	13502460
9. 300	150	150	6075	1	100	150,000	0	0	0	0	0	0	13502460
10. 300	150	150	6300	1	100	150,000	0	0	0	0	0	0	13502460
11. 400	200	200	1500	1	100	200	0	0	0	0	0	0	13502460
12. 400	200	200	2250	1	100	200	0	0	0	0	0	0	13502460
13. 400	200	200	3000	1	100	200	0	0	0	0	0	0	13502460
14. 400	200	200	3750	1	100	200	0	0	0	0	0	0	13502460
15. 400	200	200	4500	1	100	200	0	0	0	0	0	0	13502460
16. 400	8	60	1306	1	100	400,000	0	0	30.	20.	16,000	30,000	13502460
17. 400	8	60	2443	1	100	400,000	0	0	30.	20.	16,000	30,000	13502460
18. 400	8	60	1306	1	100	400,000	0	0	30.	20.	20,000	120,000	13502460
19. 400	8	60	2443	1	100	400,000	0	0	30.	20.	20,000	120,000	13502460
20. 400	8	60	1416	1	100	400,000	5	50	30.	40.	16,000	30,000	13502460
21. 400	8	60	2836	1	100	400,000	5	50	30.	40.	16,000	30,000	13502460
22. 400	8	60	1416	1	100	400,000	5	50	30.	40.	20,000	120,000	13502460
23. 400	8	60	2836	1	100	400,000	5	50	30.	40.	20,000	120,000	13502460
24. 400	4	12	1382	1	100	400,000	0	0	30.	80.	16,000	30,000	13502460
25. 400	4	12	2676	1	100	400,000	0	0	30.	80.	16,000	30,000	13502460
26. 400	4	12	1382	1	100	400,000	0	0	30.	80.	20,000	120,000	13502460
27. 400	4	12	2676	1	100	400,000	0	0	30.	80.	20,000	120,000	13502460
28. 1000	50	50	2900	1	100	1,000,000	0	0	0	0	0	0	13502460
29. 1000	50	50	3400	1	100	1,000,000	0	0	0	0	0	0	13502460
30. 1000	50	50	4400	1	100	1,000,000	0	0	0	0	0	0	13502460
31. 1000	50	50	4800	1	100	1,000,000	0	0	0	0	0	0	13502460
32. 1500	75	75	4342	1	100	1,500,000	0	0	0	0	0	0	13502460
33. 1500	75	75	4385	1	100	1,500,000	0	0	0	0	0	0	13502460
34. 1500	75	75	5107	1	100	1,500,000	0	0	0	0	0	0	13502460
35. 1500	75	75	5730	1	100	1,500,000	0	0	0	0	0	0	13502460
36. 8000	200	1000	15000	1	100	4,000,000	100	300	0	0	30	0	13502460
37. 5000	150	800	23000	1	100	4,000,000	50	100	0	0	0	0	13502460
38. 3000	125	500	35000	1	100	2,000,000	25	50	0	0	0	0	13502460
39. 5000	180	700	15000	1	100	4,000,000	100	300	0.1	0.7	3000	5000	13502460
40. 3000	100	300	23000	1	100	2,000,000	50	100	0.1	0.7	2000	4000	13502460

ated.) There are three other arrays (each one is at most node-length in size) whose sole purpose is to insure nonduplication of arcs. The program contains only two other arrays. One contains the supplies of each source; the other is an array whose size is equal to the number of sinks and is used in connecting chains to sinks. Thus, the total array core requirements of the code is at most 5 times the number of nodes, which accounts for our ability to generate problems with thousands of nodes and an unlimited number of arcs. This is in contrast to the rudimentary network generators which use a node-by-node incidence matrix. The improvement in the size capability is in addition to the flexibility and generality discussed earlier. Thus, the generator should prove to be quite helpful to practitioners and researchers for many years since no existing codes known to the authors are capable of solving such large networks.

Benchmarks

To help establish a small set of problems which researchers can use to benchmark their codes, we have generated and solved 40 network problems using 4 network codes.

TABLE 2

Solution Times (sec) and Optimal Objective Function Value

Problems	PNET	SUPERK	SHARE	Boeing	Objective Function Value
1	1.30	5.68	17.76	30.25	2,153,303
2	1.49	6.47	21.34	21.59	1,950,881
3	1.94	6.87	26.16	31.47	1,565,928
4	1.64	6.57	25.13	36.47	1,462,732
5	1.88	6.77	30.97	47.73	1,342,058
6	3.55	11.05	46.40	46.64	2,302,477
7	4.06	12.86	65.92	113.12	2,046,034
8	4.72	13.69	81.00	175.10	2,155,354
9	4.80	13.40	81.21	186.99	1,775,454
10	5.88	14.13	84.24	184.75	2,145,687
11	3.52	6.44	19.93	30.39	4563
12	4.87	6.47	21.17	22.08	3389
13	5.52	7.25	25.81	20.02	3070
14	6.02	6.95	24.95	23.11	2754
15	6.50	7.56	27.05	21.08	2721
16	2.02	5.22	21.51	42.13	80,300,844
17	3.23	8.47	32.40	54.10	44,683,151
18	2.38	4.77	20.06	20.52	79,820,813
19	3.17	8.20	31.75	53.29	44,683,151
20	2.36	5.59	22.00	20.21	72,099,415
21	3.71	7.59	27.36	67.28	39,471,683
22	1.97	5.42	21.53	21.36	71,534,527
23	3.20	7.34	26.83	65.36	38,552,093
24	2.68	5.51	23.46	20.81	79,368,634
25	3.26	8.14	31.34	56.85	56,967,178
26	2.33	3.43	17.01	12.60	68,152,192
27	3.30	5.79	20.84	40.24	46,440,505
28	6.35	13.91	53.87	83.98	122,582,531
29	7.39	14.51	52.55	117.83	105,050,119
30	9.08	16.00	61.33	152.21	86,331,458
31	9.59	17.05	61.33	135.73	82,561,499
32	15.70	22.88	78.63	553.93	174,279,219
33	20.20	25.89	101.92	210.14	195,931,070
34	17.10	25.42	92.25	248.16	160,007,929
35	19.39	29.96	DNR	DNR	162,270,303
36	384.08	NA	NA	NA	860,372,467
37	245.40	NA	NA	NA	351,773,733
38	140.98	NA	NA	NA	84,886,945
39	225.89	NA	NA	NA	512,076,450
40	129.69	NA	NA	NA	130,362,218

NA—Code and data would not fit in 104,000 words of memory.
DNR—Did not run.

Table 1 contains the specifications of these 40 problems as required on the input cards. Problems 1–5 are 100 × 100 transportation problems; problems 6–10 are 150 × 150 transportation problems. The parameter specifications of the first ten were picked to correspond with the problems in [1], [13] to provide a basis for comparison. Problems 11–15 are 200 × 200 assignment problems.

Problems 16–27 are 400 node capacitated network problems; problems 28–35 are uncapacitated 1000 and 1500 node network problems. The parameter specifications of these problems, like the transportation problems, were picked to correspond with the problems in [1]. (The problems in [1] were generated using a preliminary version of this network generator.)

To facilitate and encourage the development of large scale codes, we have included a few problems (problems 36–40) which are at the frontier of large scale solution code capability. These problems are small, however, compared with the capability of the network generator.

The four codes which we used to solve the problems are those of SHARE ([3], [17]), Boeing, SUPERK [1], and PNET. The first three codes are out-of-kilter codes while the last is the special purpose simplex network code referenced in footnote 1f. The Boeing code, obtained through Chris Witzgall, was developed at the Boeing research laboratories. Table 2 contains the solution times (not including input and output) and optimal objective function value for each of the problems. (The objective function values are included to help code developers verify the solution accuracy of their codes.)

The times reported in Table 2 were obtained on a CDC 6600 using the FORTRAN RUN compiler. The authors have solved some of these problems on a UNIVAC 1108 using the FORTRAN V compiler, and on the IBM 360/65 using the H compiler. The times on the UNIVAC 1108 were about 10% slower, while the times on the IBM 360/65 were about 12% slower.

A noteworthy feature of the computational results that pervades the entire study is that PNET and SUPERK are decidedly superior to the other codes. (Roughly, PNET and SUPERK are at least 4 times faster and in many cases 8–10 times faster than the other codes.) Furthermore PNET strictly dominates SUPERK. (PNET is roughly twice as fast as SUPERK.) This is a surprising result since a nonextreme point algorithm is generally believed to be faster especially on assignment problems. Another advantage of the primal simplex approach indicated by the computations is the core requirements of such a code. Whereas all of the out-of-kilter codes require at least 7 arc-length arrays and 4 node-length arrays, PNET only requires 3 arc-length and 8 node-length arrays. Thus PNET is capable of solving much larger problems than the other codes.

One of the unique findings obtained by a joint analysis of the three problem types is that assignment problems appear the easiest to solve, followed by general minimum cost flow network problems, and hardest to solve are the transportation problems. This finding is unexpected since it is a part of the folklore that transportation problems are easier to solve than network problems.

These computational results on the transportation problem demonstrate the need for researchers to use a standard problem generator since the same problem parameters were used to generate both the problems in [1], [13] and these problems with different generators. However, the transportation problem solution times on the out-of-kilter codes (particularly the SUPERK times) are substantially longer than (twice as long as) those reported in [1], [13]. Similarly, network problems 16–35 were generated using the same parameters and only different versions of our generator. However, the solution times reported in Table 2 are slightly longer than those in [1].

References

1. Barr, R. S., Glover, F. and Klingman, D., "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," C. S. Report 102, Center for Cybernetic Studies, University of Texas, BEB-512, Austin, 1972.
2. Bennington, G. E., "An Efficient Minimal Cost Flow Algorithm," O. R. Report 75, North Carolina State University, Raleigh, North Carolina, June 1972.
3. Clasen, R. J., "The Numerical Solution of Network Problems Using the Out-of-Kilter Algo-

rithm," RAND Corporation Memorandum RM-5456-PR, Santa Monica, California, March, 1968.

4. Canavas, G. C., "GETRAN—Random Number Generator," NASA-Langley, 1967.
5. Dantzig, G. B., *Linear Programming and Extensions*, Princeton, N. J.: Princeton University Press, 1963.
6. Dennis, J. B., "A High-Speed Computer Technique for the Transportation Problem," *Journal of Association for Computing Machinery*, Vol. 8, (1958), 132–153.
7. Flood, M. M., "A Transportation Algorithm and Code," *Naval Research Logistics Quarterly*, 8 (1961), 257-276.
8. Florian, M. and Klein, M., "An Experimental Evaluation of Some Methods of Solving Assignment Problem," *Can. Op. Res. Soc. Journal*, 8 (1970), 101–108.
9. Ford, L. R., Jr., and Fulkerson, D., *Flows in Networks*, Princeton, N. J.: Princeton University Press, 1962.
10. ——, "A Primal-Dual Algorithm for the Capacitated Hitchcock Problem," *Naval Research Logistics Quarterly*, 4, 1 (1957) 47–54.
11. Fulkerson, D. R., "An Out-of-Kilter Method for Solving Minimal Cost Flow Problems," *J. Soc. Indust. Appl. Math*, 9 (1961), 18–27.
12. Glickman, S., Johnson, L. and Eselson, L., "Coding the Transportation Problem," *Naval Research Logistics Quarterly*, 7 (1960), 169–183.
13. Glover, Fred, Karney, D., Klingman, D. and Napier, A., "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Management Science*, *20*, 5 (January 1974).
14. —— and Klingman, D., "Double-Pricing Dual and Feasible Start Algorithms for the Capacitated Transportation (distribution) Problem," University of Texas at Austin, 1970.
15. Grigoriadis, M. and Walker, W., "A Treatment of Transportation Problems by Primal Partition Programming," *Management Science*, *14*, 9 (May 1968), 565–599.
16. Lee, S., "An Experimental Study of the Transportation Algorithms", Master's Thesis, Graduate School of Business, University of California at Los Angeles, 1968.
17. "Out-of-Kilter Network Routine," SHARE Distribution 3536, SHARE Distribution Agency, Hawthorne, New York, 1967.
18. Shamma, M. M. and Williams, T. A., "Constructing Sample Networks for Analyzing and Comparing Shortest Route Algorithms," 1972 ORSA-TIMS-AIIE Joint National Meeting Atlantic City, New Jersey.
19. Spivey, W. A. and Thrall, R. M., *Linear Optimization*, New York: Holt, Rinehart and Winston, 1970.
20. Srinivasan, V. and Thompson, G. L., "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," to appear in JACM.
21. Swart, W. W., "A Course Scheduling Problem: Model Development, Analysis, and Solution Algorithm," Presented to the 41st National Meeting of the Operations Research Society of America, New Orleans, Louisiana April 26–28, 1972.
22. Transworthe, R. C., "Random Numbers Generated by Linear Recurrence Modulo Two," *Mathematics of Computation*, 19 (1965) 201–209.