Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

A METHOD FOR RANDOMLY GENERATING CAPACITATED NETWORKS

Kwang Shin
Steve Corder

Department of Computer Information Systems
Arkansas State University
State University, Arkansas 72467

ABSTRACT

A description is given of an approach to randomly generating capacitated (s,t)-networks of 10 nodes or more. A generator program written in BASIC demonstrates the approach. The results of run time experiments conducted with the generator for various network sizes are presented as well. The generator, which can produce large networks very quickly, is designed for use with capacitated network optimization models.

1 INTRODUCTION

This paper offers an efficient yet relatively simple approach to randomly generating capacitated (s,t)-networks of 10 nodes or more. The generator program presented in this study accepts a single input value -- the number of nodes in a network, and outputs a stream of three integer numbers that represent the head node, tail node, and capacity of an arc in the network. The generator is designed for use with capacitated network optimization models as a supplier of large and diverse test problems.

The specific design decisions and the random process involved with the generator are described. Storage space for the arrays involved is analyzed and the variables used are briefly described. We then report results of run time experiments carried out with the generator for various sizes of networks.

2 THE APPROACH

The approach we use in this study is based on the concept of "layered network" borrowed from Dinic (1970). Assume that we want to construct a network of 20 nodes with node 1 designated as the source node (s) and node 20 as the sink node (t). Between s and t, layers of intermediate nodes are successively built from left to right, as illustrated in Figure 1.

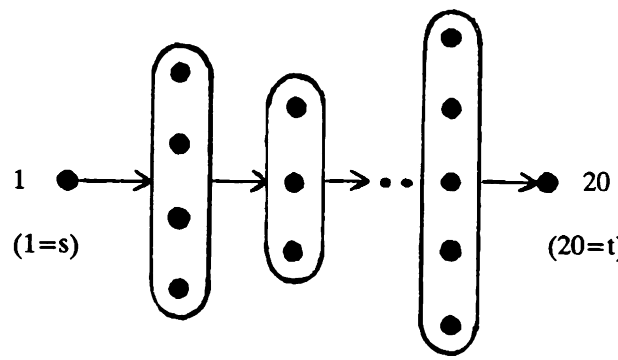

Figure 1: Constructing a Layered Network

The size of a layer is determined by the formula:

$$m = \text{integer}(2\sqrt{n}) \qquad (1)$$

where n is the number of nodes in the network and m is the layer size (the maximum number of nodes in a layer). After this upper bound is set, a uniform random process determines how many nodes are to be placed in the layer. The average size of layers, m/2, should be equal or approximately equal to the average number of layers in the network so that the network will be neither flat nor fat. Formula 1 provides a best-fit approximation to that objective, as the calculations shown in Table 1 indicate.

Table 1: Average Size of Layers vs. Average Number of Layers in Network of Size n

n	m	m/2	Avg No of Layers*
10	6	3	3
20	8	4	5
30	10	5	6
40	12	6	7
50	14	7	7
60	14	7	9
70	16	8	9
80	16	8	10
90	18	9	10
100	20	10	10
200	28	14	15
300	34	17	18
400	40	20	20
500	44	22	23
1,000	62	31	33
2,000	88	44	45
5,000	140	70	72
10,000	200	100	100

* based on m/2 and n

For the first layer after node s, a random number between 1 and m is drawn to indicate the number of nodes to be put in that layer. The source node is linked up with all of the nodes placed in the layer, which are numbered sequentially beginning with 2. The results are directed arcs 1→2, 1→3, and so on.

Nodes in a pair of adjacent layers are linked up as follows. Let layer j be a current layer and layer i its predecessor layer. Assume that layer j has three nodes in it, numbered 6, 7, and 8, and layer i has four nodes, numbered 2 through 5. For each node present in the current layer, a random number is drawn within the range of the numbers of nodes in the predecessor layer. For example, if the random number drawn for node 6 is 3, the link is 3→6. If a node in the predecessor layer remains unlinked after all nodes in the current layer have been linked, a random number is drawn from the current layer to connect the node in the forward direction. This ensures that all of the nodes in the two layers will be linked.

Let r be the number of remaining nodes to be generated for the network. For each successive pair of adjacent layers, the arc generation process is repeated until the condition $2 < r \leq m$ is encountered. In this case, the random variate for the current layer takes on a value between 1 and r-1. The random arc generation process is continued until r = 1 or 2. When r = 1 or 2, the random process is unnecessary. The sink node is linked from all nodes in the layer preceding it, forming directed arcs leading to node t.

3 IMPLEMENTATION

The head node i and tail node j of arcs are stored in two separate arrays in the order in which they are generated. At the completion of the arc generation process, the two arrays are sorted in sequence by node j within node i. This arrangement of arcs is consistent with the input requirements of maximum flow models -- a class of capacitated network optimization systems. Generating random capacities of the arcs is accomplished after the arc sorting is complete. Arc capacities are stored in a third array.

Three other arrays are needed for implementing the generator: one to contain nodes of a current layer, another to hold nodes of its predecessor layer, and still another to mark nodes of the predecessor layer. These three arrays take up relatively small amounts of memory space as they are dimensioned by m.

The three arrays that hold head nodes, tail nodes, and arc capacities are the most expensive in terms of memory space used. What should be the declared size of these arrays? To answer the question, first consider the case for n = 20. Among all of the network configurations possible, the one shown in Figure 2 yields the maximum number of arcs, 34, that is possible with n = 20.

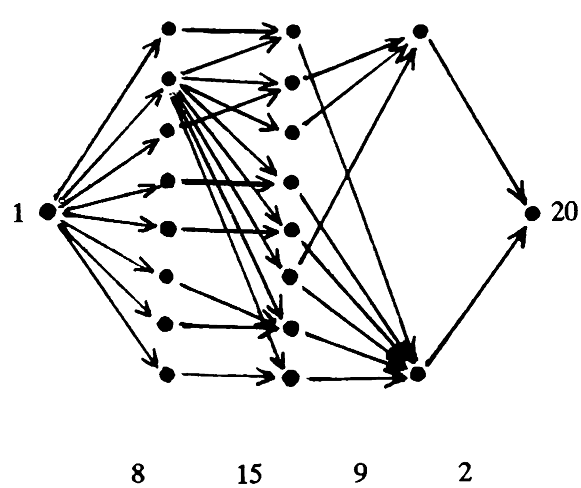

Figure 2: A 20-Node Network Configuration

Let k be the maximum number of arcs possible for a given n ($n \geq 10$). The values of k were calculated for various values of n. They are presented in Table 2.

Table 2: The Relation Between Network Size (n) and Maximum Number of Arcs Possible (k)

n	k
10	15
20	34
30	54
40	73
50	93
60	112
70	132
80	152
90	172
100	192
200	389
300	588
400	787
500	985
1,000	1,980
2,000	3,974
5,000	9,961
10,000	19,947

From these sufficient observations of the relation between n and k, it is clear that $k < 2n$ for any $n \geq 10$. (One can also verify that this inequality holds for any layer size m for any $n \geq 10$.) Therefore, the three principal arrays can be dimensioned with at most 2n elements.

A total of seven scalar variables, excluding loop control variables, are used for implementation. The names and descriptions of the variables are given in Table 3.

Table 3: Variable Descriptions

Variable	Description
NSIZE	Number of nodes in a network (minimum 10)
LSIZE	Maximum number of nodes for a layer
NODES	Accumulator for the number of nodes generated
ARCS	Accumulator for the number of arcs generated
RND1	Random variate between 1 and the size of a predecessor layer
RND2	Random variate between 1 and the size of a current layer
RND3	Random variate between the lowest and highest numbers of nodes in a layer.

The generator program, written in BASIC, is presented with sample output in the appendix. It uses BASIC's timer variable TIME$ to supply seeds for the internal random number generator RND. The program is easily modified so that the user can supply the seed in order to replicate a network previously created. The user is asked to enter a number for the size of a network to be generated. After some pause (the time lapse will vary depending on the network size), the screen will display three numbers on each line corresponding to the head node, tail node, and capacity of an arc. The same output is printed on the printer as well. Arc capacities generated by the program are integer values between 1 and 100, inclusive. The type and range of the random numbers can be modified by the user with a simple change on the program.

4 RUN TIME EXPERIMENTS

Run time experiments were conducted with the generator for various values of n using a Zenith PC-compatible based on the Intel 8088 processor running at 10 MHz. The results presented in Table 4 are average run times for the selected values of n. The figures enclosed in parentheses are average run times measured with the sort routine removed from the program.

Table 4: Run Times for Selected Network Sizes

n	Run Time (hh:mm:ss)	
10	02	(:01)
20	07	(:01)
30	13	(:03)
40	24	(:03)
50	42	(:04)
60	01:02	(:06)
70	01:26	(:06)
80	01:47	(:08)
90	02:09	(:09)
100	02:52	(:10)
150	05:54	(:16)
200	10:38	(:25)
250	17:44	(:31)
300	23:39	(:42)
350	30:25	(:43)
400	39:00	(:51)
450	55:01	(:01:06)
500	01:07:06	(:01:18)
550	01:21:57	(:01:30)
600	01:42:42	(:01:34)

An interesting outcome of the run time experiments is that as much as 98.5% of the run time is spent on sorting arcs after they are generated. This is not surprising because it is a well-known fact that sorting large-sized arrays (200 elements or more) takes a substantial amount of time. Besides, the interpretive BASIC used for the tests works relatively slowly on sorting. The generator without the sort routine, on the other hand, produces large networks very quickly. On our test machine, networks involving 600 nodes requires less than 1 minute and 40 seconds to generate.

5 CONCLUSIONS

This study provides an approach to quickly generating capacitated (s,t)-networks. The result of applying the method is a network with one source node, one sink node, and a succession of intermediate "layers" of nodes. The layer approach allows the resulting network to be more easily interpreted and graphically represented than the creation of a completely randomly connected network which can appear, once drawn, as little more than a barely interpretable jumble of lines and nodes.

The generator presented in this paper can produce very large networks very quickly. However, succeeding operations which might be required such as sorting the arcs may be very time consuming. That limitation is based on other models and is not the primary emphasis for this paper.

From a researcher's point of view, the implementation of the method allows the easy generation of many different examples of networks with a given number of nodes for empirical testing of capacitated network optimization algorithms.

From an educator's point of view, the ability to generate multiple examples of a network can be ideal for creating exercises to practice existing network analysis techniques.

We hope that the approach presented in this study may serve as a springboard to the development of new algorithms.

APPENDIX

Generator Program

```
100  INPUT "How many nodes does the network
     have (min 10) "; NSIZE
110  IF NSIZE < 10 THEN 100
120  LPRINT "Number of Nodes = "; NSIZE
130  LSIZE = INT(SQR(NSIZE)) * 2
140  DIM INODE(2*NSIZE),
     JNODE(2*NSIZE), CAP(2*NSIZE),
     CLAYER(LSIZE), PLAYER(LSIZE),
     MARK.PLAYER$(LSIZE)
150  RANDOMIZE VAL(RIGHT$(TIME$,2))
160  RND1 = INT(LSIZE * RND) + 1
170  FOR I = 1 TO RND1 'Link s to nodes of
     first layer
180     INODE(I) = 1
190     JNODE(I) = I+1
200     PLAYER(I) = I+1
210  NEXT I
220  ARCS = I - 1
230  NODES = RND1 + 1
240  WHILE NSIZE - NODES > LSIZE
250     RND2 = INT(LSIZE * RND) + 1
260     GOSUB 600
270  WEND
280  IF NSIZE - NODES > 2 THEN
        RND2 = NSIZE - NODES - 1:
        GOSUB 600
290  IF NSIZE - NODES = 2 THEN 360
300  FOR I = 1 TO RND1 't is the only
     remaining node
310     ARCS = ARCS + 1
320     INODE(ARCS) = PLAYER(I)
330     JNODE(ARCS) = NSIZE
340  NEXT I
350  GOTO 450
360  NODES = NODES + 1
370  FOR I = 1 TO RND1  'Only one node
     remains before t
380     ARCS = ARCS + 1
390     INODE(ARCS) = PLAYER(I)
400     JNODE(ARCS) = NODES
410  NEXT I
420  ARCS = ARCS + 1
430  INODE(ARCS) = NSIZE - 1
440  JNODE(ARCS) = NSIZE
450  FOR I = 2 TO ARCS - 2 'Sort arcs in
     sequence by node j within node i
460     FOR J = I + 1 TO ARCS - 1
470        IF INODE(I) > INODE(J) THEN
              SWAP INODE(I), INODE(J):
              SWAP JNODE(I), JNODE(J)
480        IF INODE(I) = INODE(J) AND
           JNODE(I) > JNODE(J) THEN
              SWAP JNODE(I), JNODE(J)
490     NEXT J
500  NEXT I
510  FOR I = 1 TO ARCS  'Print arcs and arc
     capacities
520     CAP(I) = INT(100 * RND) + 1
530     PRINT INODE(I), JNODE(I), CAP(I)
```

```
540     LPRINT INODE(I), JNODE(I), CAP(I)
550   NEXT I
560   END  'End of program
600   FOR I = 1 TO RND2  'Subroutine to link
      nodes of two adjacent layers
610     NODES = NODES + 1
620     ARCS = ARCS + 1
630     RND3 = INT(RND1 * RND) +
          PLAYER(1)
640     INODE(ARCS) = RND3
650     JNODE(ARCS) = NODES
660     CLAYER(I) = NODES
670     FOR J = 1 TO RND1
680       IF PLAYER(J) = RND3 THEN
            MARK.PLAYER$(J) = "Y"
690     NEXT J
700   NEXT I
710   FOR I = 1 TO RND1
720     IF MARK.PLAYER$(I) = "Y" THEN
          770
730     RND3 = INT(RND2*RND) +
          CLAYER(1)
740     ARCS = ARCS + 1
750     INODE(ARCS) = PLAYER(I)
760     JNODE(ARCS) = RND3
770   NEXT I
780   FOR I = 1 TO LSIZE
790     PLAYER(I) = CLAYER(I)
800     CLAYER(I) = 0
810     MARK.PLAYER$(I)=""
820   NEXT I
830   RND1 = RND2
840   RETURN   'End of subroutine
```

Number of Nodes = 20

1	2	4
1	3	17
1	4	78
1	5	89
1	6	95
1	7	1
1	8	83
2	9	36
3	10	98
3	15	92
4	11	42
4	12	89
5	13	73
5	14	86
6	15	22
7	12	77
8	15	43
9	17	98
10	17	42
11	16	79
12	18	76
13	19	8
14	19	13
15	17	91
16	20	90
17	20	42
18	20	12
19	20	15

REFERENCE

Dinic, E. A. 1970. Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. Soviet Math Dokl. 11: 1277-1280.

AUTHOR BIOGRAPHIES

Kwang Shin is an Assistant Professor of Computer Information Systems at Arkansas State University. His research interests are discrete system simulation and computer modeling. He received his Ph.D degree from Southern Illinois University-Carbondale.

Steve Corder is an Assistant Professor of Computer Information Systems at Arkansas State University. His major research interests are system development methodologies and software metrics. He received his Ph.D degree from Georgia State University.