Topological Sorting of Large Networks

A. B. Kahn

Westinghouse Electric Corporation, Baltimore, Maryland

Topological Sorting is a procedure required for many problems involving analysis of networks. An example of one such problem is PERT. The present paper presents a very general method for obtaining topological order. It permits treatment of larger networks than can be handled on present procedures and achieves this with greater efficiency. Although the procedure can be adapted to any machine, it is discussed in terms of the 7090. A PERT network of 30,000 activities can be ordered in less than one hour of machine time.

The method was developed as a byproduct of procedure needed by Westinghouse, Baltimore. It has not been programmed and at present there are no plans to implement it. In regard to the techniques described, Westinghouse's present and anticipated needs are completely served by the Lockheed program, which is in current use.

Introduction

In recent years much work has been done on the computational analysis of problems which can be formulated as networks with directed elements. In particular, this class of problems include Pert (Program Evaluation Review Technique) which is used as a management schedule tool. To a large extent the feasibility of network computations depends upon the ability to arrange the network information in topological order. A list in topological order has a special property. Simply expressed: proceeding from element to element along any path in the network, one passes through the list in one direction only. Stated another way, a list in topological order is such that no element appears in it until after all elements appearing on all paths leading to the particular element have been listed.

In the customary Pert computation the topological ordering is the most difficult aspect of the problem from the computational viewpoint. A simple solution is to number the elements so that the numbers along each path always ascend as one proceeds along the path. When this is done, topological order can be achieved with a simple sort. While this technique requires tedious effort, it has been used successfully on moderate-sized networks. However, for large networks two difficulties arise: (1) It is difficult to assign blocks of numbers to groups working independently on sections of the network, taking into account the problems of merging the sections. (2) A minor change in the network organization can require renumbering of large sections of the network. Since the Pert networks are not static but are undergoing revision on a periodic schedule, this tedious job is repetitive. In addition, the identity of elements is constantly changing raising administrative difficulties.

Thus it becomes vital that the computer be capable of reducing a randomly labeled network to topological order. Most of the existing programs are the result of earlier attempts with capacity limits of 5,000 to 10,000 elements. Only two papers on these approaches have been published [1, 2]. More recently, there have been several attempts at larger capacities, but no discussions of them have been published. Furthermore, although the lack of information prevents final judgment, it is felt that the current approach may be more efficient in machine time.

There have been numerous groups who have felt the need for much larger capacities. One of the difficulties is that the blocking and merging techniques of conventional sorts are not readily applicable here. The purpose of this paper is to disclose a technique which will order about 30,000 elements in about one hour of IBM 7090 time. This method is discussed with respect to the 7090. However, it can be readily adapted to make maximum use of the capacity of most medium or large scale computers. It is essentially based upon the ranking approach.

The method to be presented was not developed to meet a specific need. Rather it resulted as a byproduct of a technique developed for small networks. Thus in no way can this paper be taken as a recommendation for use of large networks. Rather it is intended solely to provide an efficient technique for those who have decided that it is to their advantage to handle large networks. In addition, there are many other network problems for which these techniques may be desirable. *The author has neither programmed nor at present plans to program the procedure described.*

The method conveniently breaks down into three parts. The middle part uses the basic algorithm which is the key to the procedure. The purpose of the preparatory part is solely to get from a convenient input format to the most efficient format for the basic algorithm. Similarly, the concluding part merely serves to get to the desired output format which can be subsequently used for computation. Although the first and last parts contain extensive processing they consist entirely of standard data processing techniques. Therefore, the discussion will commence with the basic algorithm followed by the auxiliary portions of the procedure. Before this, however, there will be a brief discussion of PERT terminology.

1. PERT Network Terminology

In general a network consists of a set of nodes or points connected by links which may or may not be directed. In PERT networks the nodes are points in time and are called events. Directed links connect the events and are called activities. They usually represent tasks which consume time. The events are uniquely, though arbitrarily labeled. The activities are designated by the pair of labels for its predecessor event, from which it proceeds, and the successor event at which it terminates. The terms "predecessor" and "successor" are used rather loosely to indicate either activities or events which proceed or succeed a specific activity or event. The terms may be used to designate either an immediate or a remote connection. In general the meaning is clear through context. For example, in

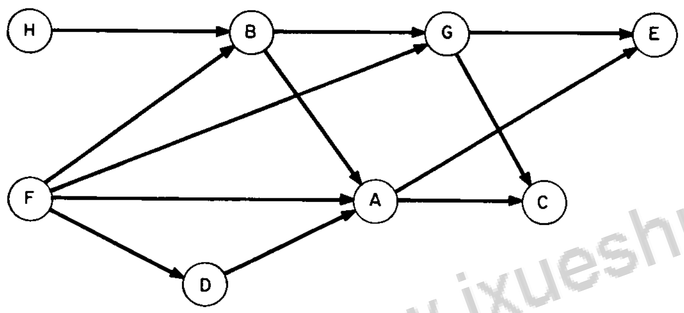

FIG. 1. Sample PERT network

Figure 1 events B and F are immediate predecessors of event G, and events E and C are immediate successors. Events H, B, D and F are remote predecessors to event E. Finally, G is a remote successor to event H. All events have predecessors except H and F which are initial events. Events E and C are terminal events with no successors. Events B and D are on parallel paths and are neither predecessors nor successors to each other. Other pairs of events related in this fashion are: H and F; H and D; G and A; and E and C.

Similarly, we can talk of activity A-C as being a successor to event H or activity D-A, and so forth.

2. The Basic Algorithm

In general there are two classes of techniques for topological ordering: threading and ranking. The basic algorithm to be presented makes use of a ranking procedure, which proceeds in parallel on all paths and therefore requires only as many iterations as the maximum number of reversals on any one path.

The basic procedure is to establish a list of all unique events with a count of the number of immediate predecessors of each event. To begin, the initial events can be assigned sequence numbers starting from one. The sequence numbers represent the sorting key which can ultimately be used to obtain the desired order.

Considerable processing is required to arrange the input into the most efficient form for the basic algorithm. For the present we will assume the data have been prepared for the basic algorithm. The preparatory procedures are discussed below.

The key to the efficiency of the algorithm is two lists: a list of activities and a list of events. In order to reduce the need for random searching certain features are present in these lists when the basic algorithm is executed. These are described below and illustrated in Figure 2. Figure 2 presents the two tables at the start of the algorithm. Only columns a, b, c and d are actually in core. The other columns are included solely to permit the reader to relate these tables to the sample network of Figure 1.

1. The activity list is sorted by predecessor. This puts all activities with the same predecessor event together in a block. Column a in the activity list is a one-bit flag to indicate the last activity in such a block.

2. The location of the first activity of the block mentioned above is placed opposite the predecessor in the event list (column d). In the special case of a terminal

Activity List

Tutorial			In Core	
Item	Event Labels		Predecessor Flag (a)	Successor Event Location (b)
	Predecessor	Successor		
1	A	C	0	3
2	A	E	1	5
3	B	A	0	1
4	B	G	1	7
5	D	A	1	1
6	F	B	0	2
7	F	A	0	1
8	F	D	0	4
9	F	G	1	7
10	G	C	0	3
11	G	E	1	5
12	H	B	1	2
			1 bit	15 bits

Event List

Tutorial		In Core	
Item	Label	Count (c)	Activity Location (d)
1	A	3	1
2	B	2	3
3	C	2	Z
4	D	1	5
5	E	2	Z
6	F	0	6*
7	G	2	10
8	H	0	12*
		5 bits	15 bits

FIG. 2. Tables at start of basic algorithm

event a special flag, Z, is used to indicate that there is no cross reference.

3. It is necessary to enter the event list from the activity list; therefore, the location (in the event list) of the successor event of each activity is available in column b of the activity list.

4. Column c in the event list is the count of the number of outstanding predecessors for those events which are not as yet ordered.

The procedure is indicated in the flow chart in Figure 3. It commences by searching the count list (column c) for zeros. (The loop through locations 10 and 20 shown with heavy lines in Figure 3.) At the start this count would be zero for all initial events. These events can be immediately assigned serial numbers; and the zero in column c is set to -1, indicating that the event has been ordered (location 11). Furthermore, their successors now have one less predecessor outstanding; therefore, the count for these successors can be decremented by one (loop through locations 30 and 40). Finally, the search for zeros continues down the list (location 20).

In the above process the count for some events will be reduced to zero. This means that all the predecessors of these events have been ordered and now they in turn can be ordered by assigning to them the next available serial numbers. Now the count for the successors of these events can in turn be decreased. Wherever a counter is decremented it is tested and if a zero was created a flag, F, is set (following location 30). When the complete list is searched a check is made (location 90), and if F is set it is reset and another iteration is made (location 1), starting the search again from the top of the count list (column c).

If the flag, F, is not set at the end of an iteration (location 100), one of three conditions exist: the process is completed, there is an illegal loop in the network, or segmenting procedures are being used. The first condition can be verified simply by comparing the number of events against the current value of the serial number. The latter two conditions are discussed in a later section.

Although shown in the event list for simplicity, it is actually more convenient to keep the initial events, marked with an * in Figure 2, in a separate list. For the case illustrated, this list consists simply of: 6, 12. This list obviates the need to search for the initial zeros.

3. Space Requirements

Assuming that the program for the basic algorithm is a separate link in a chain of programs, $2K$ should be ample for the basic algorithm. This leaves $30K$ (in a $32K$ machine) for the lists a, b, c and d in which about 30,000 activities can be accommodated in the following fashion.

The activity list items consist of two fields: a and b. It is clear that a is one bit while 15 bits for b would allow for 32,678 events. Similarly, 15 bits for d in the event list would allow the same number of activities. If the count c is limited to 5 bits this means that up to 30 activities can end at any single event. (Two of the 32 possible values are reserved for 0; and a flag to signify that an

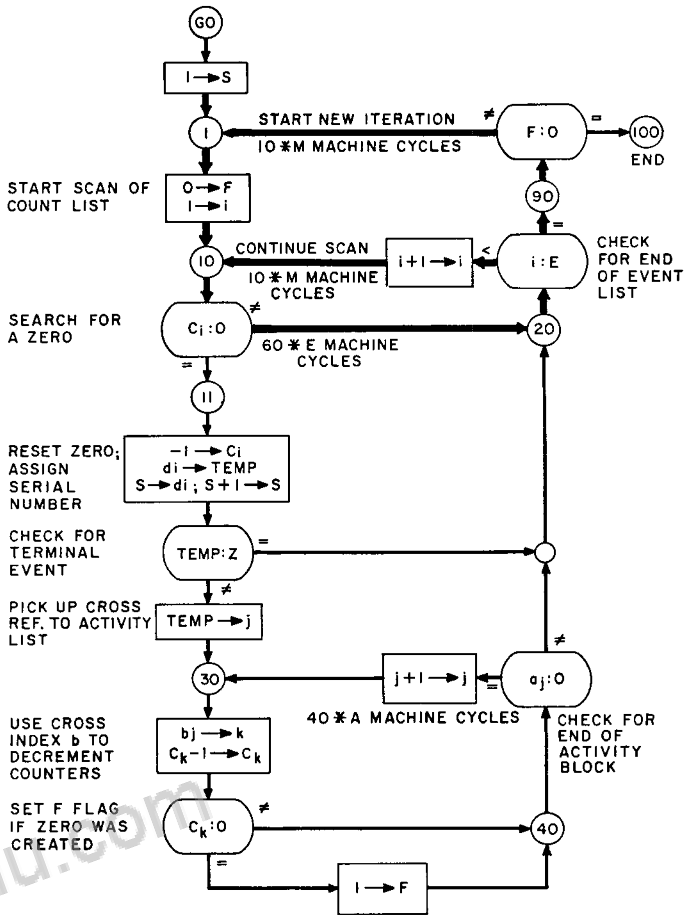

FIG. 3. Flow chart of basic algorithm

event has been ordered.) In rare cases where more than this number are required dummy events can be introduced. Furthermore, this can be done automatically in the early portions of the program.

In short, one 36-bit word can contain all the information required for one event and one activity. Since there are generally more activities than events some space is wasted by making both lists the same length. However, this space could be used if there is sufficient need to warrant the additional logic. If this were done the maximum number of activities could be 60,000 less the number of events. This would necessitate another bit in the d field which could be obtained by reducing the count field.

4. Analysis of Timing

One of the esthetic features of this procedure is that the timing can be readily analyzed. This method is fairly efficient for a number of reasons:

1. There is never any searching for an item, the location is always known.

2. If the network is labeled in sequence, then only a single iteration is made because the first zero count is encountered at the top of the list and as it is processed further zeros are created which will be encountered in the same pass. As these in turn are processed, additional zeros are created and all are processed in a single pass.

3. When an event is not in topological order, it means that when its count is reduced to zero the scan for zero counts will be below rather than above its location in the list. This means that this zero will not be picked up immediately but will wait until the next iteration. However, progress proceeds unhindered on parallel paths. Thus the number of iterations is not determined by the total number of events out of order, but only the maximum number of adverse[1] labels on any one path. Let this quantity be called M.

Let us now estimate an extreme value for M. Most PERT networks cover a period of less than 5 years with the average duration of activities greater than two weeks. Let us assume a ten-year network with activities of one week duration. Under these assumptions the longest possible chain would have less than 600 activities. It is very unlikely than an entire chain of 600 activities would be in adverse order. However, let us assume the worst which yields a maximum value for M of 600.

In the flow diagram of Figure 3 the lower portion of the diagram (location 11) is entered only once for each event as it is ordered. If we assume 30,000 events this loop should require less than 1 minute of IBM 7090 time. This represents a minimum total time which is required if the net is in order to begin with. The upper double loop which is drawn heavily is where the effect of adverse event labeling is felt. The outer loop (starting at location 1) is executed M times and the inner loop (starting at location 10) E (the number of events) times. Although M is the number of adverse labels in the network, the number of instructions in these loops is small. The longer loops in the lower part of the flow chart are dependent only on the size of the network and not its organization. The portion from 11 to 30 is executed once for each event and the portion from 30 to 40 once for each activity. Using 2.18 microseconds for the cycle time of the 7090, and estimating the number of instructions in each loop, the time, T, is:

$T = 2.18 \times 10^{-6} \times 10(M + M \times E + 4A + 6E)$ seconds

M = max number of events labeled out of order on any one path

A = number of activities

E = number of events

Assuming $A = E = 30{,}000$, (*a*) for the most favorable case $M = 0$; $T = 6.5$ seconds; (*b*) for the worst case $M = 600$, $T = 402$ seconds.

Thus the time ranges from under 10 seconds in the most favorable case to under 7 minutes in the worst case.

The value of 600 is a safe upper bound for M since even a PERT network with 30,000 activities is likely to last less than 10 years and to have activity duration of more than a week. Furthermore, it is unlikely to be labeled in worst order. A more reasonable expected value for M would be about 100 which would result in a machine time of less than 2 minutes.

5. Termination of the Basic Algorithm

The basic algorithm is terminated when a complete iteration is made with no new zeros created and location 100 on the flow chart is reached. At this point one of the following three conditions exists.

(1) All events have been ordered.

(2) A loop exists in the network.

(3) If segmenting has been used to increase capacity it is necessary to go to other segments.

The first condition is verified by simply comparing the number of events ordered to the total number of events present (within the segment).

In the case of a loop in the PERT network it is important to not only determine its existence but also to obtain some indication of its location.[2] The simplest diagnostic to implement is to list all events which have not been ordered but have had their predecessor count decremented at least once. Such events are either in a loop or have a predecessor in a loop. (This requires saving the event labels and counts on tape in the preparatory portion of the program.) A second procedure is to start over again from the input data working backwards. This can be achieved by simply reversing the role of predecessor and successor of each activity as they are read in. Any event which remains unordered on both the forward and the backward passes is either in a loop or lies between two loops. A third diagnostic, using precedence matrices, has been developed which will specify a loop exactly and probably even isolate two or more independent loops. However, it is too lengthy to report here and it would be difficult to apply to more than 1,000 events.

The third condition which can arise occurs when the technique is expanded to handle larger networks which must be segmented. Here again there is an inherent advantage in the method in that it is always known precisely what segment each event is in. However, rather than switch segments each time a request is generated, the requests can be collected and effort continued with the current segment until an impasse is met. Just as in the case of adversely numbered events, here too the fact that progress continues along parallel paths increases efficiency. The number of segment exchanges required will not be equal to the number of segment interfaces but probably to the maximum number of interfaces encountered in any single path.

[1] The term "adverse label" is used here to mean an event which has a label greater than the label on an immediate successor event. The more usual case of an event label which is less than all its immediate successors is referred to as a "favorable label." In both cases the terms "greater than" and "less than" are in reference to the sorting sequence used.

[2] Since it is sometimes difficult to isolate loops, and several of the popular programs do not provide such diagnostics, it is worth mentioning a helpful procedure. The method is based upon the fact that any loop must contain at least one activity which is adversely labeled. Thus, it is useful to list all activities which have a predecessor event label which is higher in the sorting sequence than the successor event label. In many networks this list will contain only a small percentage of the activities, and thus the search is considerably narrowed.

6. Preparatory and Concluding Processing

There is, of course, much processing required in addition to the basic algorithm. In fact, in general, this processing will require more time than the basic algorithm. However, there are two distinguishing advantages here: (1) all the processing other than the basic algorithm has open-ended capacity; (2) for the most part standard, available routines such as conventional sorts are used. In general, the input will consist of a record for each activity with predecessor event label, successor event label, and other information.

It is assumed that with the volume of data involved there will be some form of file maintenance. In this case we can assume or require the input to be sorted by successor event label.

The first step is to abstract from each record the required information: predecessor event label; successor event label; location of record in the input. This is the working activity file.

Since this is sorted by successor it is a simple matter to develop an event file from it with successor event labels and the counts of immediate predecessors (column *c* of Figure 2). (These two field items should be saved on tape for the loop diagnostic.) At the same time the location of the successor in the event file is appended to the working activity file (column *b*).

At this point the working activity file is resorted by predecessor, following which the location of the input records may be placed on tape to be merged back when needed. With this sort the predecessor block flag (column *a*) can be readily established. In addition, since the predecessor labels of the activity file are in the same order as the successor labels in the event file, it is possible to compare the two sets of labels with a matching operation. Three conditions arise for each event:

	Predecessor	*Successor*	*Type of Event*	*Action*
1.	Present	Absent	Initial	Place location of activity block in starting list
2.	Present	Present	Middle	Place location of activity event list (column *d*)
3.	Absent	Present	Terminal	Place terminal flag, *Z*, in event list (column *d*)

At this point the event labels are no longer required. Columns *a*, *b*, *c* and *d* are all that remain in core for the use of the basic algorithm which may now proceed. At first the starting list is exhausted and then the iterative searching for zeros commences. As the serial number for each event is found several procedures can be used according to whether the topological order is to be by predecessor or successor and whether the serial numbers are to serve solely as a sorting key or whether they are also to be used in further computation. In the first two cases the serial numbers are overlaid on column *b* or *d*, respectively. In the last case they are written in a separate file (on tape if maximum capacity is required) with the location of the event in the event file. The activity records with the serial numbers are then assembled in a separate pass.

Following the basic algorithm the location of the input records are merged with serialized activity records. This is sorted by input file location to permit merging with the rest of the input. Finally, the serial numbers are used as a key to sort the full input records into topological order.

All the processing other than the basic algorithm involves 3 sorting passes plus about 7 passes which are essentially tape limited. Assuming that an efficient record size is adapted at an early stage, all the tape passes should require less than 20 minutes. Another 20 minutes should be sufficient for the sorts (using, for example, IB sort).

Thus the overall time to topologically order 30,000 activities would be 40 to 50 minutes on an IBM 7090 depending on the structure of the network.

Summary

Most of the existing procedures for topological ordering are time consuming and limited in capacity. The method presented permits a considerable increase in capacity over other available methods. The use of available internal storage is economized to the maximum extent possible to achieve this capacity with minimum machine time.

The author has neither experienced nor anticipated a need to deal with large networks. This paper is not intended as an endorsement of large PERT networks, but rather as technical assistance to those who have such needs. For this reason, the author does not plan to implement this technique, although he would be happy to aid anyone who wishes to do so.

An intriguing application, which would require the disclosed technique is analysis of the structure of a glossary. Here each definition can be represented by a group of activities with a common successor event representing the term being defined. The predecessors would be the terms used within the definition. Topological order would then be equivalent to arranging the definitions in textbook fashion such that each term is defined in terms of previously defined terms. The current technique could order a specialized glossary including about 3,000 terms with each term defined by use of about 10 other terms. The author is currently pursuing this as well as other applications.

ACKNOWLEDGMENTS

Appreciation is expressed to Homer L. Smith of Boeing Aircraft Company, Seattle for stimulating the preparation of this paper by raising the question of how far a small capacity version of the algorithm could be extended. Appreciation is also extended to J. W. Froggatt and N. Kleiner of Westinghouse Baltimore for critical review of the technique.

REFERENCES

1. Anonymous (1958), Summary Report, Phases 1 and 2, Program Evaluation Research Task. U. S. Government Printing Office, Washington, D. C.
2. D. J. LASSER (1961), Topological ordering of a list of randomly-numbered elements of a network. *Comm. ACM 4*, 12 (1961).