To be presented at NASA/Industry
PERT Computer Conference in
Houston, Texas, July 22, 1964

LEWIS-GODDARD NASA PERT PROGRAMS IN COMPILER LANGUAGE

by Elizabeth Ryan and Ross C. Bainbridge

Lewis Research Center
Cleveland, Ohio

INTRODUCTION

The program to be described is a PERT time program written entirely in compiler language and with a capacity in excess of 30,000 activities. The program was written at Lewis Research Center with the assistance of Hans Bremer and N. H. Dillard of Goddard Space Flight Center. (Topological ordering by the pushdown technique is described in the conference paper by Hans Bremer.)

To best understand the reasons for production of this new program, a brief review of the history for writing the program and also of the programing philosophy at Lewis Research Center will be presented.

The project was proposed by Lewis in March of last year as a solution to several problems that had arisen in using the machine-coded programs. Briefly, these problems can be summarized as follows:

(1) The machine-coded program could be run only on one manufacturer's equipment. This required sole source replacement of the computing equipment when replacement was required for only 5 percent of the load.

(2) A great deal of time of systems personnel was being spent in maintaining machine-coded programs and in modifying them each time a systems or hardware change was implemented.

(3) The adoption of new hardware by an installation without a change of manufacturer often requires extensive rewriting of the machine-coded programs. For example, the substitution of disks or drums for tapes requires considerable program revision.

It appeared that these problems were typical to the exclusive use of machine-language programs and that compiler-written programs designed to run under a typical monitor system would eliminate these problems. A compiler-written program would have the added advantage of discouraging the use of undocumented binary patching to the program decks. Since modifications to a compiler-written program can much more easily be made simply by recompiling a source deck, language documentation would automatically be provided for all modifications.

Lewis proposed to write this PERT program in FORTRAN IV because it is the compiler that has been implemented by most computer manufacturers, and it is the compiler language most used by industry as well as NASA. It is known that for a given algorithm an optimum machine-coded program is faster than an optimum compiler program. But as the algorithm becomes larger or more complex, practical considerations of time and personnel prevent the production of an

optimum machine language program whereas an optimum compiler-written program can still be obtained. For example, Lewis has written several large systems entirely in FORTRAN with excellent results. These include a production IBM 1401 SPS assembler written originally in FORTRAN II and a FORTRAN II compiler and assembler written in FORTRAN II, which were more efficient in running time than the FORTRAN II compiler and assembler on the IBM 7090. In fact, this compiler and assembler have since been converted to FORTRAN IV and are still heavily used in production status producing data reduction programs. A final example of compiler-written programs is the SIFT program written in FORTRAN II, which made most FORTRAN II programs FORTRAN IV compatible.

PROGRAM DEVELOPMENT

In June 1963 the Lewis proposal was approved with the following restrictions:

(1) The program should be written in compiler language. Machine language would be permitted only where large gains could be made in efficiency.

(2) The running time of the new program should not exceed running times of the existing program.

(3) The program should have an ultimate capacity of 30,000 activities. The use of a modular or skeletonizing technique to achieve this would be considered.

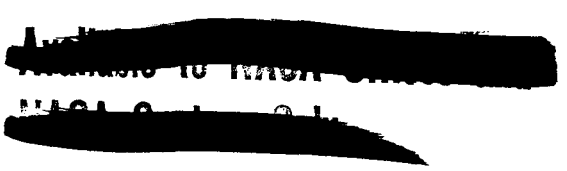
(4) The format of input and output data would not be altered.

(5) The program should be compatible with the data processing equipment then being used for PERT throughout NASA.

The first phase of the project - the production of a limited capacity PERT time program entirely in FORTRAN IV - was completed in October. The program, Lewis-Goddard PERT TIME I, has a capacity of 3500 activities and has been distributed to the following installations and manufacturers:

NASA
- Goddard Space Flight Center
- Langley Research Center
- Ames Research Center
- Lewis Research Center
- George C. Marshall Space Flight Center

Manufacturers
- CDC
- UNIVAC
- Honeywell



Industrial
- Bellcomm
- Aerojet General
- Westinghouse
- Goodrich

SHARE

The program is in exclusive production use at Lewis, Ames, Goddard, and Langley. The program is in operation at Aerojet and Bellcomm on an IBM 7040 and IBM 7044. It has been submitted to and is available through SHARE. The number of installations that have received the program through SHARE is not known.

PERFORMANCE DATA

Run times have been considered favorable on all machines used thus far. The following performance data are for the first phase, PERT TIME I, as recorded on an IBM 7094, running tape to tape using 729V tape drives at 800 BPI on two data channels. Times are exclusive of load time and reflect some time savings obtained by the blocking of output at 5 lines per record.

Activities	Outputs	Time, min
200	3	0.2
200	5	.3
300	4	.4
1000	5	2.5
1600	3	2.5
2830	4	7.5

Time studies were run using the configuration against the NASA PERT Mod-B machine-coded program that was then still in production at Lewis. Comparative timings on the machine indicate that the program is 50 percent faster than the Mod-B PERT machine-coded program for networks under 1100 activities, requiring no output merging; equal in speed for networks between 1100 and 2100 activities, requiring a single output merge; and within 10 percent for networks over 2100 activities and requiring multiple output merges.

Since the original time study was conducted, computer configurations have been switched and the following times for a directly coupled IBM 7094 model II IBM 7040 with a disk, drum, and four model VI tape units can be reported:

Activities	Outputs	Time, min
0 to 1000	3 to 5	Under 0.5
1000 to 2000	3 to 5	Under 2.0
2000 to 3000	3 to 5	Under 5.0

Also on subnetted jobs where no more than a single merge is ever needed, subnetted jobs have been run with a total of 4000 activities in under 6 minutes.

The next set of performance data was provided by Mr. E. Kilroy of Computer Usage Company subcontracted to Bellcomm of Washington, D.C., from runs made on a 7040/44 direct couple system with partial use of disks in place of tape. The overlay feature utilized when running on our 7094 was not necessary at Bellcomm.

Activities	Outputs	Time, min
1839	3	7.4
204	3	.2
1330	2	3.2

Again times do not include load time. Mr. Kilroy also estimated that this one time conversion cost to an IBM 7040/44 system was approximately $3800. This cost included personnel and computing. A complete rewrite of the program would have an estimated cost in excess of $50,000. We feel this is a good illustration of the cost savings of a compiler-written program.

Running times from Aerojet using an IBM 7044 with 10 tape drives, 4 disks, 2 channels, and a 1401 off-line are as follows:

Activities	Outputs	Time, min
81	2	0.8
270	3	1.5
325	4	2.9
1300	4	3.0
2000	4	4.4

This actually shows a 24-percent reduction in running cost over the NASA PERT B, which runs on Aerojet's IBM 7094. These data were supplied by Mr. T. C. Adams, a systems analyst at Aerojet General, Sacramento. The appendix is an internal memorandum written by Mr. Adams in which he evaluated the Lewis-Goddard PERT time program summary on the IBM 7044 versus the Mod-B program on the IBM 7094. We found this evaluation to be very informative as to the use of an IBM 7044 as a PERT management production tool. The ease of modifying the program is attested not only by the variety of machines on which it is running but also by the many features that have been added by individual installations.

EXTENSION OF PROGRAM TO HANDLE LARGER NETWORKS

The second phase of the project was begun in December 1963. Its aim was to build the PERT TIME I program into a program of much greater capacity with several new features. This was done while still retaining ability to process all smaller networks already using the program. That program, Lewis-Goddard PERT TIME II, has been in production and will be made available to NASA

installations along with a detailed system manual and a separate looseleaf users manual that can be updated.

The capacity of the PERT TIME II program is in excess of 30,000 activities. The increased capacity is obtained using a subnet technique. It is of interest to note that it is possible to maintain the size of the basic subnet at 2200 activities, which is felt to be more than generous enough for these people having experience with the IBM Cost-Time program with the restriction of a basic subnet size of 750 activities.

At this point it is best to define what is meant by a subnet. A subnet is simply any collection of interrelated activities belonging to a PERT network. In the PERT network shown in figure 1, where the circles represent event points and the connecting lines represent activities, the shaded activities make up a subnet, as do the activities enclosed in the broken lines. Note that there are three event points (with crosses in them) common to both subnets. These events are called interface points between the two subnets. Subnets can be connected only in this way, that is, by one or more interface events. In practical terms, a subnet is often a logical entity of some kind. For instance, in a network representing a project involving four contractors (A, B, C, and D) there could be four subnets each representing the work assigned to one of the four contractors (fig. 2).

To facilitate this usage, it is not required that the interface points have the same event number interior to each subnet in which they appear. This eliminates the necessity of coordinating numbering of common events among many contractors each of whom may be maintaining his own network. To eliminate this, each interface point is given an alphabetic name, the interface label, when its subnet is to be integrated. In figure 2, for example, the interface point that has been labeled I1 may be known in contractor A's subnet as event 5000 and as event 4 in contract B's subnet. With reference to the network as a whole, however, it is simply interface event I1. An equivalence card concept was used to show this relationship. Lewis experience finds these very flexible without adding excessive card input to the program. This will be discussed later in more detail.

PROGRAM FEATURES

A useful feature of the program is the provision for a different type of subnet, the summary network. Suppose the subnet shown in figure 3 below the dotted line is being maintained by a department for its own use. It may be that only those events with upward pointing broken arrows need to be reported to higher management. These events can then be made interfaces to a subnet, which consists only of the interface events. The resulting subnet can be represented by the figure above the dotted line. It is a summary of the original subnet or subnets and shows not only event relationship but also PERT network logical flow as indicated by the solid arrows. The program will compute time estimates along each path of the summary network using the detailed paths from the original. If requested, activity cards for the summary network can be punched out with delta time estimates. This deck can then be sent on to higher management to be run as this department's subnet in a larger network.

For truly effective management reporting, for example, summarized reports are necessary as high levels of management are reached. This is illustrated in figure 4 as a pyramid of PERT reports.

Now with the ability to place time estimates into an output form (the same form as the standard NASA input) with activity times that truly reflect the interrelationships of the base network, a method of even further upward reporting is established. As shown in figure 5, the program could support basic networks of 30,000 activities at Lewis, Goddard, and Langley. In turn, various summaries are sent as basic subnets to the program running on a computer used by NASA Headquarters. The program could support a base in the example of 90,000 activities and still present top management only the few hundred activities needed at the top level of command.

People who have had much contact with PERT networks are well acquainted with dummy activities. There are several kinds of dummy activities, but this is one of the most popular (see fig. 6).

The activity connecting events 7 and 17 is a dummy inserted for the express purpose of inventing a place to hang the label END TESTING. What is actually needed here is a way of identifying event 7 as the end of testing. The insertion of the dummy has added an extra event and activity to the network. This practice as a substitute for event nomenclature is quite common and can cause a significant increase to the size of a network. While large PERT networks may be regarded as a sort of status symbol, they can be expensive.

By using the PERT TIME II program, the event 7 can be named directly by the use of an event card:

00000070000007 END TESTING

The event number is entered in both the predecessor and successor columns, and nomenclature appears in the normal field. At report time, the event will appear in normal sort order with its expected and allowed dates and slack. The event card does not in any way enter into the PERT calculation and so does not increase the size of the network.

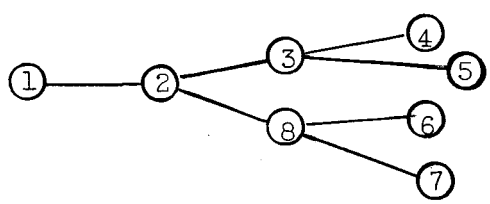
The updating or file maintenance technique used in PERT TIME II also represents a new approach. Previously the master file has been nothing more than a tape bearing the activity cards for a given network. When it was desired to change the network, the tape was first updated to obtain a new master file, and the new master file was used as input for a complete reexecution of the network. The PERT TIME II program performs updating as a part of the normal PERT run, thus eliminating duplication of operations. A tape developed as part of the PERT calculation is used as the master file. This tape contains not only the activity cards (in blocked form) but also all other information needed to make reports directly from the tape without recalculation. All this information is separated by subnet. Since many times not all subnets need be changed on a given update run, only those that are changed need be recalculated. The master tape is read only once as updating and recalculation of a subnet are overlapped. In addition to providing a fast and efficient update,

this technique eliminates dependence on the availability of a second computer.

As a further aid to maintaining networks, completed activities can be automatically deleted from the master file as an option. This feature has two important results. First, by eliminating past activities that no longer alter the project schedule, it reduces the effective in-core size of the network. Secondly, it has been found that a great deal of updating is done for the purpose of removing completed activities. This type of routine updating can now be completely eliminated.

Topological Procedures

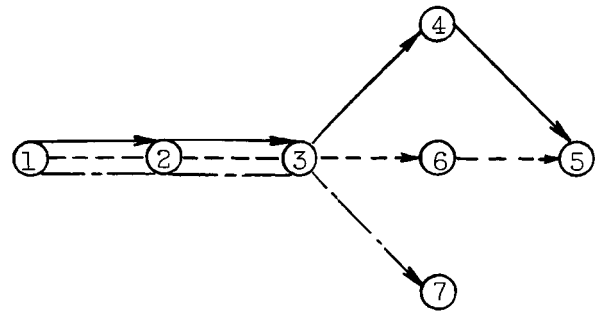
The topological or network analyzing procedures used in the Lewis-Goddard PERT time program are not the familiar topological sorting techniques used in other PERT programs. The technique used here is an application of pushdown lists or tables more commonly used in compilers and recursive routines. The pushdown table is actually a memory device used to remember decision points or alternate decision routes. For example, in the game or decision tree



it is often desirable to know the branches (paths or decisions in a game tree sense) not taken at points 2, 3, and 8. A pushdown table is used to do this by placing the alternate routes not taken at 2, 3, and 8 into a table. Upon retracing the path that actually was followed, the last nonentered branch would appear at the last entered position of the pushdown table. By extracting this branch, the alternative decision or path can also be analyzed. For example, in the figure taking path 1 to 2 to 3 to 4 would result in entering in order in a pushdown table 2 to 8 and 3 to 5. Working back from the end point 4 would result in looking at branch 3 to 5 and then the path from branch 2 to 8.

It now becomes apparent that the game tree figure actually can represent a PERT time or cost network with the decision points 1, 2, 3, etc. as events and the decisions or paths between them 1 to 2, 2 to 3, etc. as activities. Program adaptation of the pushdown table to analysis of PERT networks can actually be divided into three steps. The first step is getting the branch activities into the pushdown table. The second step with taking the last activity from the pushdown table and placing it into another table (the path list) that in final form contains all the activities on a particular network path. The third step then reduces the path list until a branch or start event is located on the pushdown table.

This procedure can be illustrated as follows:



The activities in a table are as follows:

1 to 2
2 to 3
3 to 4
3 to 6
3 to 7
4 to 5
6 to 5

The following analysis is performed:

Step	Pushdown table	Path list
(1)	1 - 2	-----
(2)	-----	1 - 2
(1)	2 - 3	1 - 2
(2)	-----	1 - 2 2 - 3
(1)	3 - 4 3 - 6 3 - 7	1 - 2 2 - 3
(2)	3 - 4 3 - 6	1 - 2 2 - 3 3 - 7

End event 7 encountered with path ——- - ——→ is located in the path list. Expected and allowed times can be calculated at this point.

Step	Pushdown table	Path list
(3)	3 - 4 3 - 6	1 - 2 2 - 3
(2)	3 - 4	1 - 2 2 - 3 3 - 6
(1)	3 - 4 6 - 5	1 - 2 2 - 3 3 - 6
(2)	3 - 4	1 - 2 2 - 3 3 - 6 6 - 5

End event 5 encountered with path - - - - - → is located in the path list. Expected and allowed times can be calculated at this point.

Step	Pushdown table	Path list
(3)	3 - 4	1 - 2 2 - 3 3 - 6
(3)	3 - 4	1 - 2 2 - 3
(2)	-----	1 - 2 2 - 3 3 - 4
(1)	4 - 5	1 - 2 2 - 3 3 - 4
(2)	-----	1 - 2 2 - 3 3 - 4 4 - 5

End event 5 again encountered with path ——→ is located in path list. Expected and allowed times can be calculated at this point. Step (3) now finds the activity list complete and will then go on to another start or if no other starts exist into another program phase.

The expected times are calculated as a path is completed by the use of a table of events. These event tables are used to keep expected and allowed

times for each network event. The expected times are forward calculated and replace the previously calculated event expected time in the events table (TSUPE) only when the expected time now being calculated is greater. Allowed times are calculated from the last to the first event with replacement of the previously calculated allowed time in the events table (TSUPL) only when the currently calculated allowed time is smaller.

When output reports are required, it becomes a simple matter to interrogate the events tables to get the predecessor and successor event times.

The following methods were used to modify the basic topological procedure and to increase its efficiency:

(1) Sequential numbering of the events eliminated events table searching. This sequential numbering also eliminated the necessity for retaining internally the actual event numbers.

(2) Retention of the activity position counter in an events table eliminates any activity table searches. This in turn eliminates the necessity for retaining internally the predecessor event of an activity.

(3) Experimentation with the expected and allowed time calculations resulted in the determination that if an expected time was less than or an allowed time greater than the previous value found in the events tables, then the path analysis could be terminated at that point. This innovation results in a considerable savings in actual internal computing times.

By introducing the pushdown table techniques into the processing of PERT networks, by doing as much in core processing as possible, and by limiting table searching and unnecessary calculation, the FORTRAN IV program developed into a highly efficient topological technique. This technique also makes modular networks easier to analyze and gives flexibility in experimentation with faster methods.

REPORTING AND SORTING TECHNIQUES

In preparing reports it is necessary to determine the order, with respect to several possible formats, of the activity records that make up each subnet. Because this ordering must be performed many times during the execution of any network, the procedure used must be as efficient as possible. The ordering method developed for use in Lewis-Goddard PERT time is now described.

The activity buffer into which the activity records have been placed constitutes a table of activities and their associated information. For each activity on the network there is an activity record and each record contains several storage words of information about its activity. Each item of activity information (predecessor and event numbers, expected and allowed dates, slack, department code, etc.) is assigned a fixed position in the activity record. With each item of information, then, can be associated two subscripts; the first refers to the position of its activity record in relation

to all other records, and the second to the particular item's position relative to all other activity information in the record.

An item of information pertaining to the 10th activity and which was assigned the 4th word in the activity record would have subscripts 10 and 4. That same item of information about activity 25 would have subscripts 25 and 4. Rather than rearranging the activity records themselves, which would be costly both in terms of execution time and core storage usage, the ordering routine rearranges their associated subscripts. At the termination of the ordering procedure there will have been produced a list of subscripts whose order indicates the order of their associated activity records with respect to the given key.

The initial phase of the process is a scanning of the activity keys to determine the extent of natural order as the records lie in core; both ascending and descending order is detected. The following list is constructed. Position 1 of the list contains the number of activity records that make up the first sequence of ordered records - the sign is made negative to indicate ascending order or positive to indicate descending order. The second position refers in the same way to the second sequence and so on, so that if the activity buffer consists of n such sequences, there will be n entries in the list. (If the activities lie in the buffer as shown in step 1, the list produced would be as shown in $LIST_1$. The first four activities are in ascending order as are the next 3. The four activities following the second sequence, however, are in descending order so that the entry is positive. The 25 activity records consist of 7 sequences as described in $LIST_1$.)

The remainder of the ordering procedure consists of combining consecutive pairs of sequences to form half as many sequences of combined length. The smaller activity key from the first sequence is compared to the smaller from the second sequence. The subscript of the activity whose key is smaller is placed in the first position of a second list (depicted in step 2 as $LIST_2$). If the smaller key came from sequence 1, the key for the next activity in sequence 1 is compared to the first activity's key in sequence 2. The subscript of the smaller is placed in the second position of $LIST_2$. Comparisons continue until subscripts of all activities in one of the sequences have been placed in $LIST_2$. The subscripts from the remaining sequence are then placed in $LIST_2$ and the combining process is repeated for the next two sequences. As each pair is combined, $LIST_1$ is revised to reflect the combined length of the sequences. (Step 2 shows $LIST_1$ and $LIST_2$ following the first stage of sorting whereby the 7 original sequences were reduced to 4. $LIST_1$ then indicates that the activities associated with the first 7 subscripts form a sequence as do the activities associated with the next 10, etc. All entries in $LIST_1$ are now left positive since after the first combination pass all sequences have been constructed in ascending order.)

The four sequences given by $LIST_2$ are now combined in the same manner to produce two sequences that are described by a list of subscripts in $LIST_3$. (The conclusion of this pass is represented in step 3.) Ordinarily at this point, $LIST_3$ together with $LIST_1$ would be used to produce a new list that will be placed in $LIST_2$, so that $LIST_2$ and $LIST_3$ are alternately used and overwritten. In practice, however, once the number of sequences has been reduced

to 2 the activity whose key is smaller is simply written as output after each compare.

TABLE I. - ORDERING PROCEDURE

Activities	Step 1		Step 2		Step 3	
	Keys	$LIST_1$	$LIST_1$	$LIST_2$	$LIST_1$	$LIST_3$
1	3	-4	7	1	17	11
2	5	-3	10	5	8	1
3	7	+4	6	2		12
4	8	-6	2	6		5
5	4	+3		3		10
6	6	+3		4		13
7	9	+2		7		2
8	7			11		9
9	5			12		6
10	4			10		14
11	1			13		3
12	3			9		8
13	4			14		15
14	6			8		4
15	7			15		7
16	9			16		16
17	11			17		17
18	10			23		23
19	8			22		25
20	6			20		24
21	7			21		22
22	5			19		20
23	1			18		21
24	4			25		19
25	3			24		18

SUMMARY

With the completion of PERT TIME II Lewis is confident that it has fulfilled its original goals.

1. The program is written in the FORTRAN IV compiler language.

2. The running times do not exceed those of the NASA Mod-B program.

3. The enlarging of the subnet technique has led to extended capabilities. The use of summary networks and the pyramid use of the program at varying levels of management are two examples of this. It is felt that both these features add greatly to the power of a PERT time program and will prove extremely useful to management groups at all levels. It is possible to handle over 30,000 activities with a maximum limit on subnets of 2200 activities.

4. Formats and outputs are compatible with existing standards.

5. The program is capable of running on data processing equipment within NASA. It has been found that the program is easy to use, as evidenced by the number of installations presently using PERT TIME I, and correspondingly easy to implement at individual installations. Additions and modifications can easily be made to adapt the program to the differing requirements of these installations.

6. The entire project was completed by only two programers, each devoting less than 1 man-year.

In short, it is felt that the Lewis-Goddard PERT TIME II is an efficient and powerful program that is easy to use and can easily be implemented. The program should prove valuable especially to computer systems groups who have long needed a standard PERT time program that can be used regardless of hardware or system demands.

APPENDIX

To: Distribution Date: 19 May 1964
TCA:mit:Ext.3090
From: T. C. Adams 2001A:2360

Subject: Lewis-Goddard PERT TIME-I for the 7040/44

Distribution: S. A. Chappell, F. W. Eagen, J. D. Poulsen, J. C. Richardson, J. V. Rizzo, J. R. Soll, R. E. Stienmuller, E. C. Wolf

Copies to: Sacto: A. Feinberg, R. T. La Sarge, R. W. Lee, R. J. Machado, File
Glendale: D. B. Cyrog

Enclosure: (1) Lewis-Goddard PERT TIME-I for the 7040/44
(2) Lewis-Goddard PERT TIME-I versus PERT System 'B'
(3) 'ASPERT,' NASA PERT TIME System on the 7040/44

I. BACKGROUND

A. The LG-PERT program (Job 24040) for the 7044 was compared to the existing NASA PERT TIME program (Job 1041AA) for running time comparisons. The test data consisted of three 'stacked' networks approximately 400 total activities. The results are below:

	1041AA (7094)	LG-PERT (7044)
Load time	0.16	1.20
Execute time	1.33	1.62
Total time	1.49	2.82
Cost	$295 - $7.35	$180 - $8.45

The results indicate the 1041AA is less expensive to run than the new '44' version. The obvious reason for this is the excessive load time cost which results while using the '44' version of LG-PERT.

B. Based on a statement by us, that this load time could be reduced significantly, Project M-1 has requested Job 24040 be put into production. The request also made provisions for the three (3) minor modifications to LG-PERT proposed in the Lewis-Goddard PERT status memo dated 3 April 1964.

II. STATUS TO DATE

A. The program is now in production with the binary program on the systems library taps. The program is called for by a "$EXECUTE LGPERT" card. The results with the previous test case data, run under the production status conditions, are as follows:

	LG-PERT (7044)
Load time	0.24
Execute time	1.62
Total time	1.86
Cost	$180 - $5.58

B. We encourage you to use this new system. We have demonstrated it is less expensive to run and expect you to take advantage of its new features. (A cost savings of 24 percent was realized in this test case.) Job reports explaining the program can be obtained from the Divisional Librarian (ask for Job 24040). Additional information can be obtained from us (refer to COSMOS Ref. v-II, p-7). An outline of the differences between Job 1041AA and Job 24040 have been included with this letter (Enclosure (2)).

C. An outline of LG-PERT's features and capabilities has been enclosed with this memo. The outline is organized by categories recommended by Corporate Systems for PERT software evaluation (Enclosure (1)).

III. FUTURE ACTIONS

A. For the first time since AGC 5-digit PERT was used by today's NASA PERT users, Computing Sciences is capable of providing programer support on the NASA PERT system. This means the requests for modifications, new features, etc., proposed by you can now be processed.

B. Project M-1 has for some time requested the capability of non-sequential data input to a NASA PERT program. Also they have been interested in Master File Maintenance for NASA PERT. The enclosed paper called 'ASPERT' (COSMOS Ref. v-II, p-9) is an overall PERT systems plan for NASA PERT users. The plan provides an overall framework for PERT user's needs; it is designed on the subsystem concept and thus could be constructed in parts if so desired (Enclosure (3)).

T. C. Adams, Analyst
Management Analysis & Programing
Computing Sciences Division

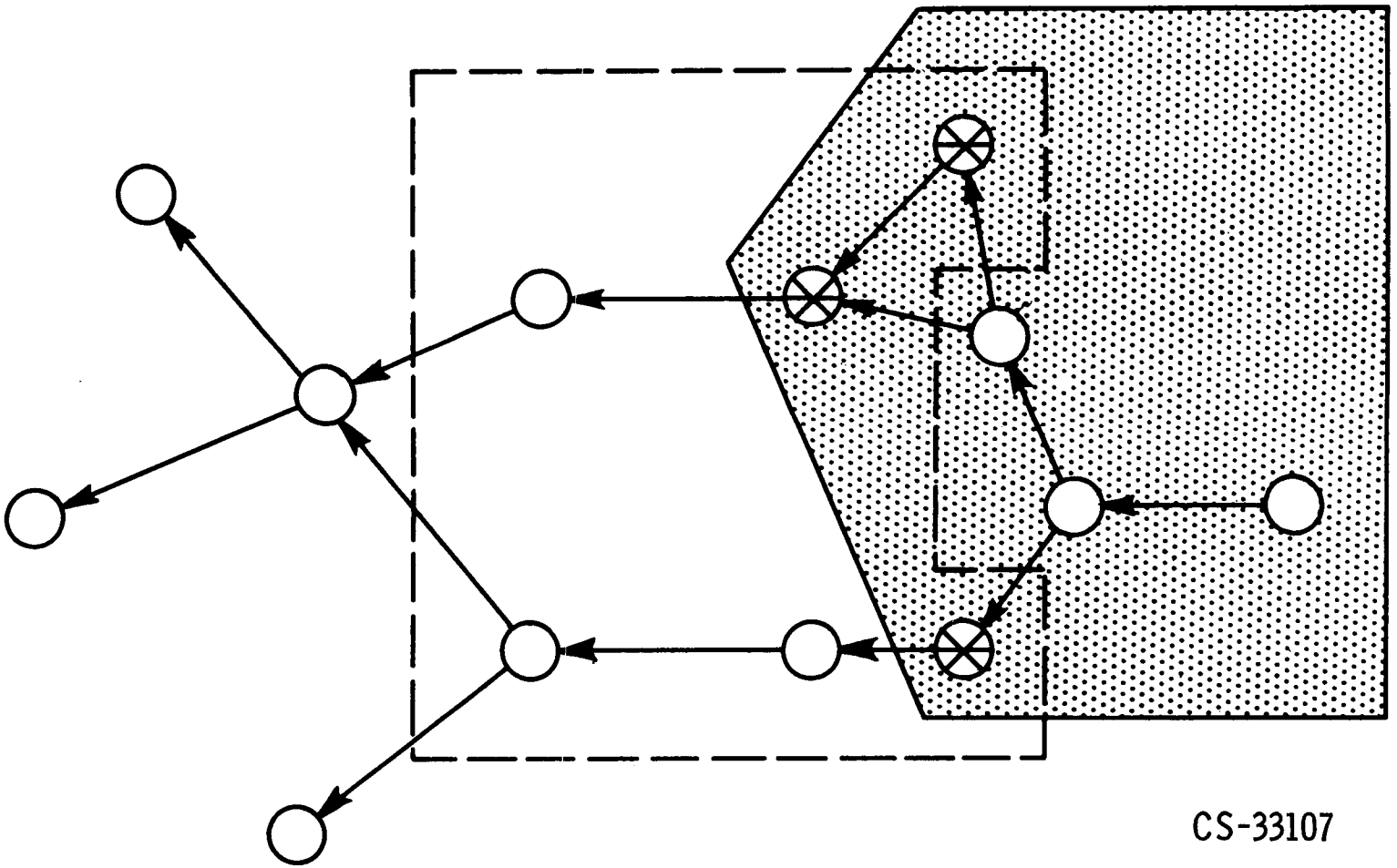


Figure 1. - PERT network.

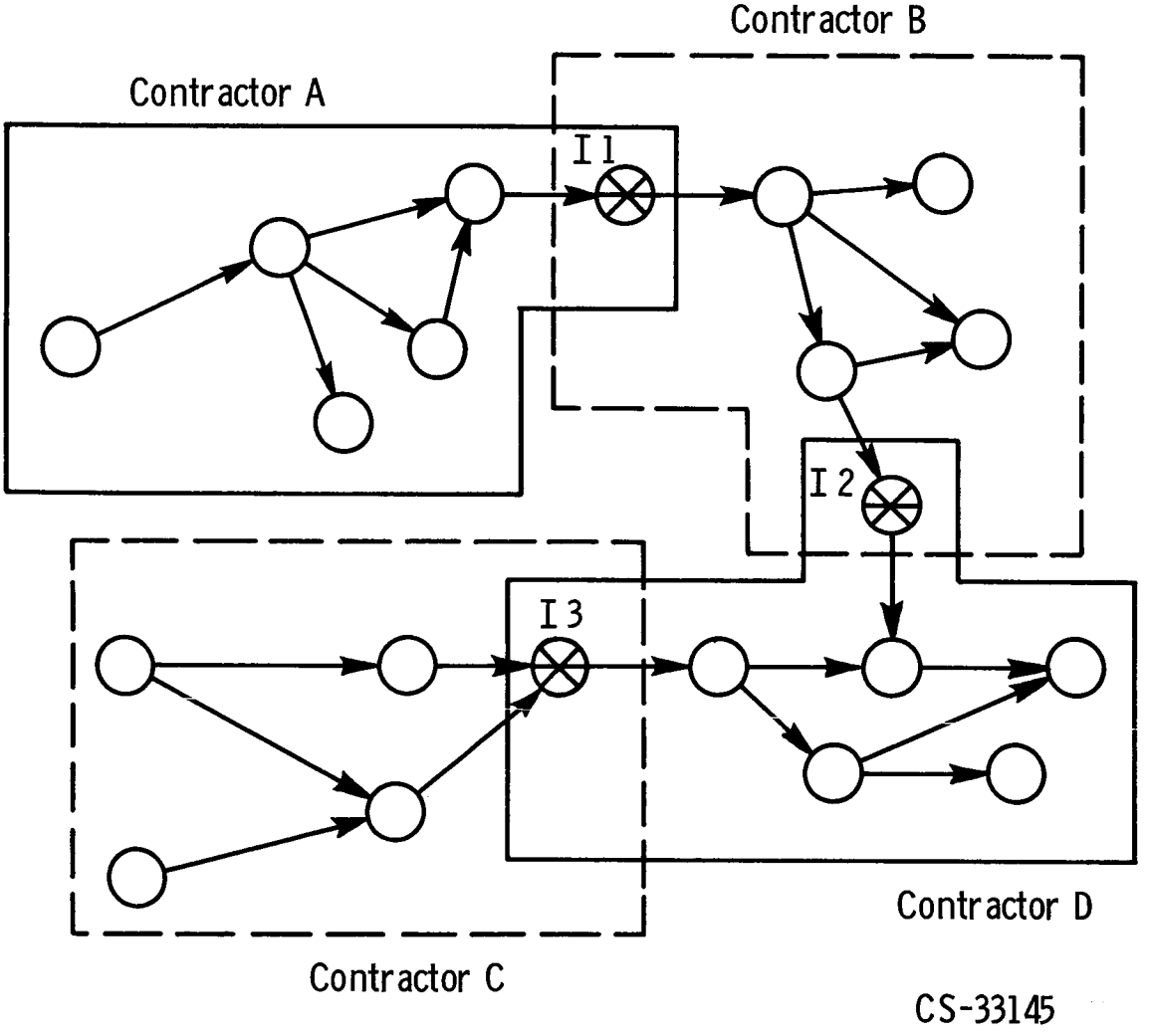


Figure 2. - Project network.

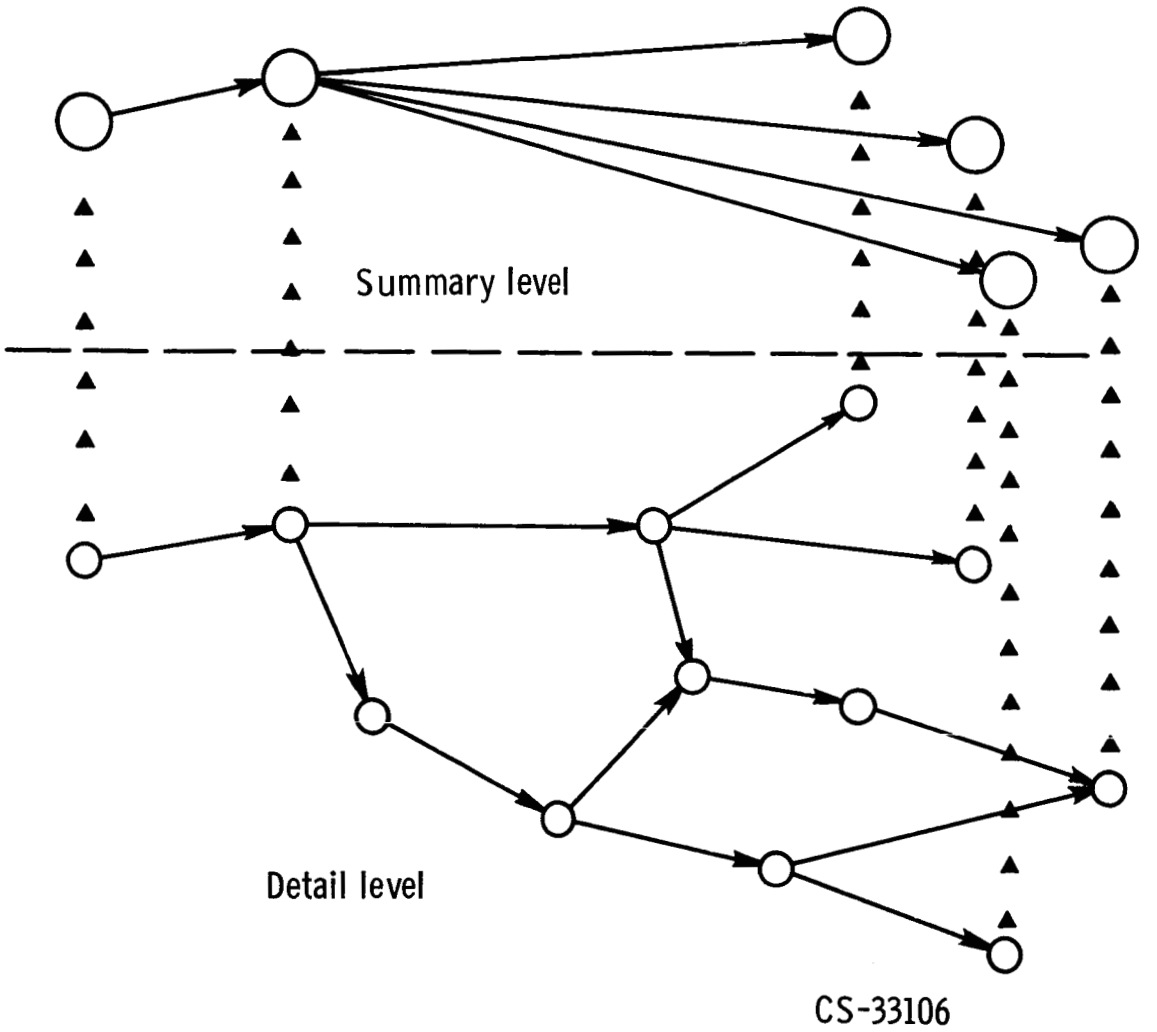


Figure 3. - Summary subnet.

E-2714

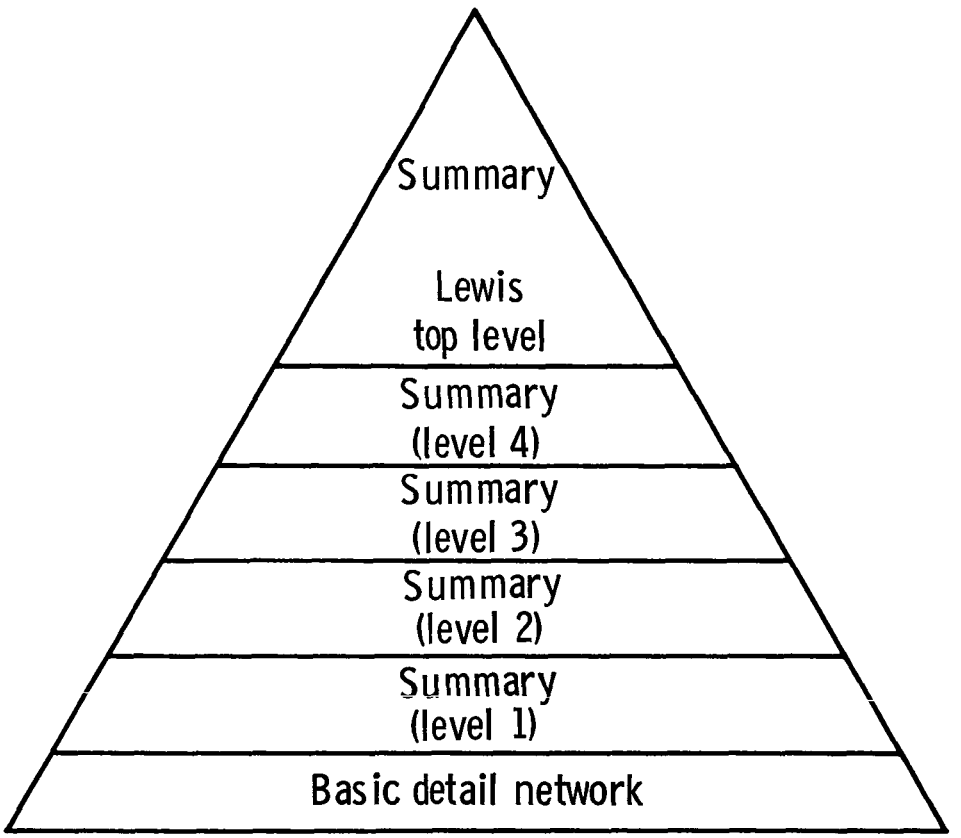


Figure 4. - Lewis PERT summary for management.

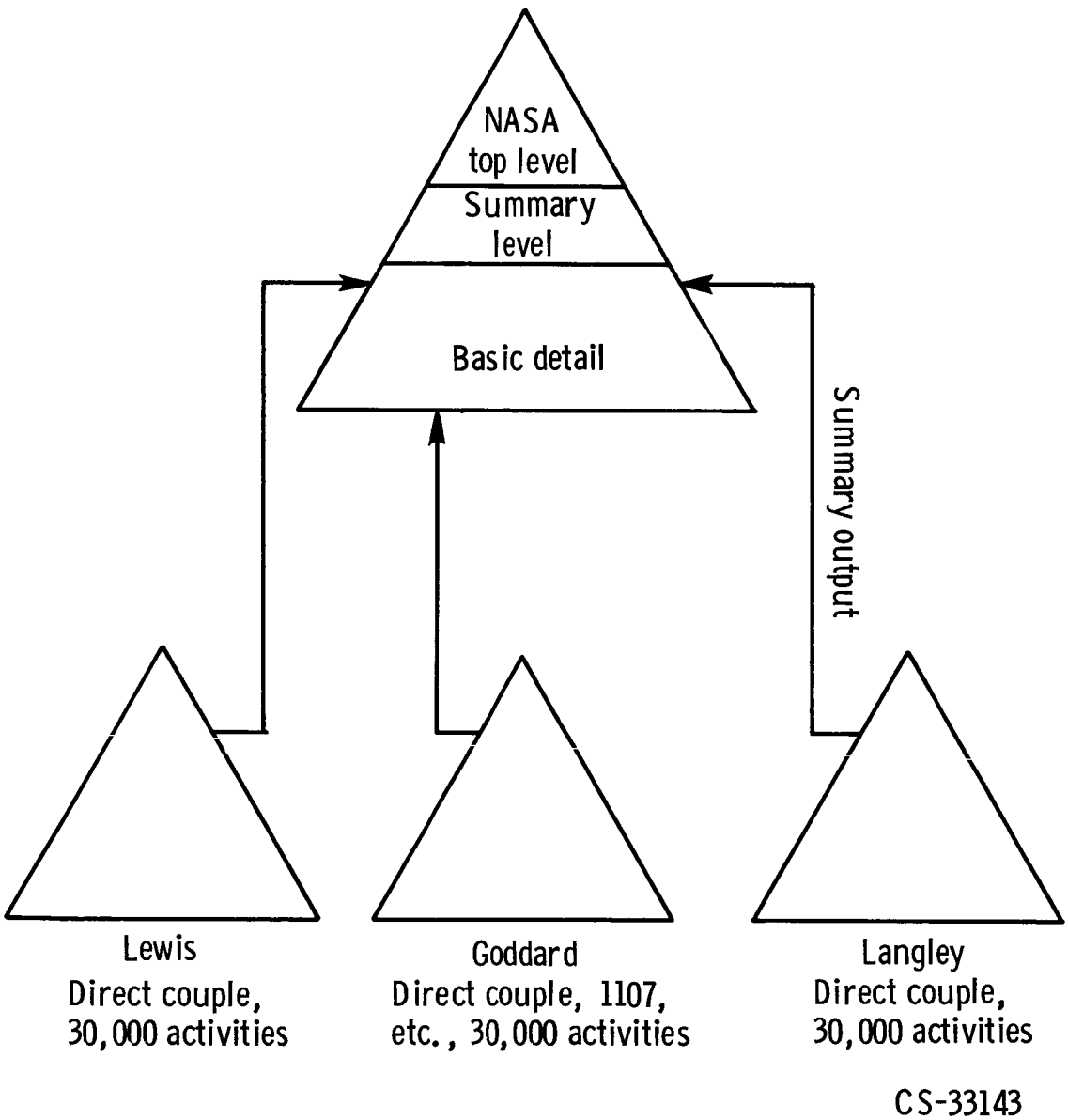


Figure 5. - Summary output becomes detail level input.

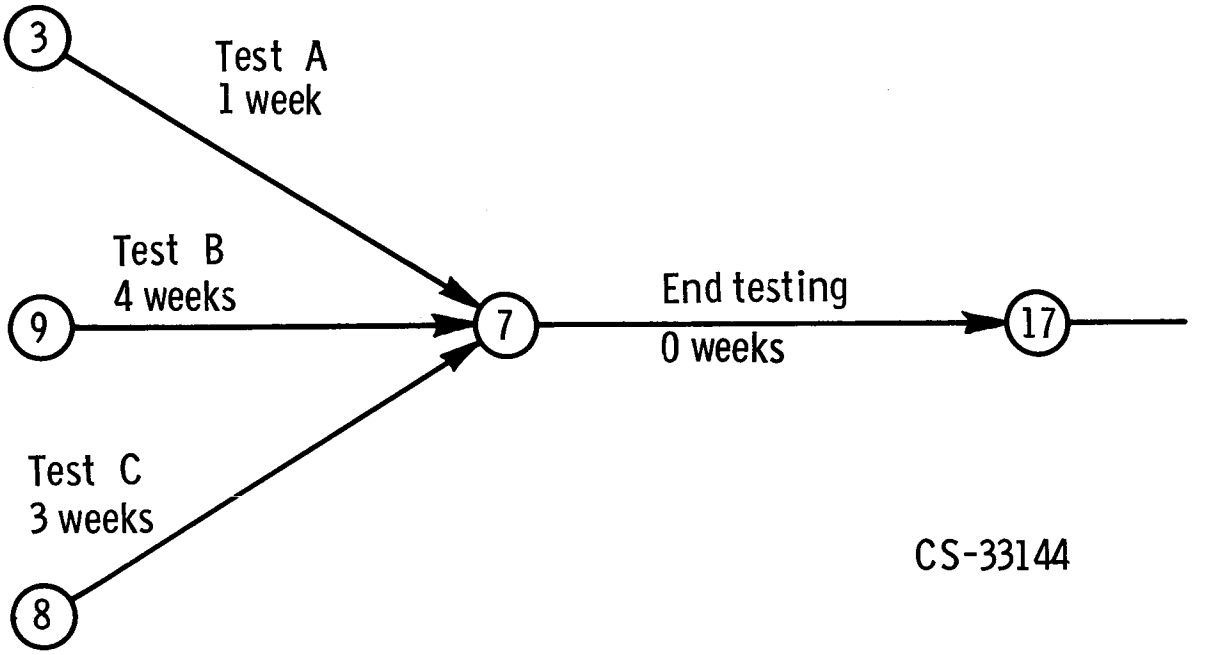


Figure 6. - Dummy activity.