Analytical techniques for the statistical evaluation of program running time

by BORIS BEIZER

Data Systems Analysts, Inc.
Pennsauken, New Jersey

INTRODUCTION

The design of large software systems or real-time systems imposes several constraints on the designer. Predominant among these are the running time of the programs, the amount of memory used by these programs, and the input/output channel utilization. A well considered design not only runs, but has optimum efficiency. Efficiency is often measured by the running time of the program.

If the designer must wait till the program is running to evaluate its running time, important design decisions will have been made, and cannot realistically be changed. Consequently, trades that could have improved the efficiency of the programs will not have been made. This will result in higher unit processing cost, increased hardware, or a reduction of available capacity. In real-time programs, the difference may be that of working or not working at all. For these reasons, the system analyst and programmer require techniques that allow the evaluation of such trades and the early estimation of running time.

Simulation is one method that has been used for timing analysis. The major blocks of the program are described in a simulation language that must be learned like any other programming language. The program simulator is run, statistics gathered, and the efficiency of the program judged thereby.

Analytical techniques, on the other hand, have not been extensively used for several reasons: the analysis has been too tedious for the value of the results obtained; such analyses have required a greater knowledge of mathematics than typical for a programmer; the solutions can be overly complicated (e.g., including transient behavior). In short, both analytical methods and simulation have been effectively inaccessible to the one who needs it most—the programmer. Yet this need not be, if we are willing to make a few analytical assumptions.

PROGRAM MODELS

Simulation or analysis both require a model. The model that we shall use for a program is based on the flow chart of the program. The model consists of junctions, decisions, and processes, as depicted in Figure 1. Associated with each process is an execution time, obtained by counting the instructions (with appropriate modification for indexing and indirect operations) within that process. Associated with each decision is a probability for each of the several exident branches. The sum of such probabilities must equal 1. Furthermore, the probabilities are assumed to be fixed and not to depend upon how the program got to the particular branch. This will be recognized as a Markov model assumption. Though this assumption is not always valid,* the number of cases in which it does not hold are sufficiently rare to allow us to ignore them. Furthermore, if we do not assume a Markov model, the resulting analysis is overly complex.

There is one difficulty with this model; the processing that takes place at a decision. It can be readily shown, that for analytical purposes, we can transform a decision to a new decision followed or preceded by processes, such that there is no work done at the decision itself. This is depicted in Figure 2.

Having done this, we can simplify the model further, by eliminating the distinction between junctions and decisions. The new model consists of nodes and links. That is, the model is a graph. Associated with the outways of each node, there is a probability that that outway will be taken. Associated with each link, there is work required for the execution of the instructions represented by that link. A link then, can represent a

* As an example, consider a program switch within a loop whose position is determined by a count of the number of passages through the loop. While the mean is unaffected, the variance will depend on the way the program got to that point.
sequence of instructions, a subroutine, or a whole program, depending upon what level we do our analysis.

Having gone this far, we can introduce a further generalization into the model. Rather than assuming that the execution time for a link is fixed, we can assume that it is really the mean value of a distribution of running times for the link. We can characterize that distribution by its mean value ($\mu$) and standard deviation ($\sigma$). In practice, we shall find it more convenient to use the variance ($\lambda = \sigma^2$) rather than the standard deviation. The resulting model is shown in Figure 3.

It is clear that any reasonable flow chart and, hence, any reasonable program operating within a single computer at one priority level, can be readily modeled in this manner.

Our problem is then: given the graph corresponding to the flow chart of a program, properly annotated with the required link properties ($\mu, \lambda, p$), determine the mean value, standard deviation, and the probability associated with every combination of flow chart entrance and exit; there is after all, no need to restrict ourselves to programs that have only a single entrance and exit—that would not be realistic.
Before we do this, however, it pays to go into the question of how we obtain these numbers.

ESTIMATION

The running times for individual links are obtained by an estimated count of the instructions in that link. This can be done precisely without programming. The real program must run, but its estimated version need not. We need not take the meticulous care that is mandatory for a real program. Furthermore, since almost all instructions are equivalent, we can replace the real instructions by estimators of these instructions. For most problems, the repertoire can be cut down to about 10 different generic instruction types. Similarly in indexing and indirect operations, we need not be concerned with which index register is used, and so forth.

The standard deviation is either externally supplied or it results from an intermediate step in the analysis. In most computers, the variation in the running time of individual instructions is small and can be ignored.

The difficult part of the analysis is the evaluation of the probabilities associated with the links. Some of these probabilities are externally supplied—that is, they are inherent in the job mix for which the program is written. How these are estimated depends upon the application. Many other probabilities, while difficult to estimate can be ignored. Consider the example shown in Figure 4. We have shown a decision which is followed by two radically different processes that take almost the same amount of time. It is clear that the probability in question is not important. Therefore, a crude estimate will suffice. The third type of probabilities are those which are inherent in the structure of the program. Thus, switches which are set on the first pass through the program and reset on the next pass, the number of times a program will go through a certain loop, etc., fall into this category. These are also readily obtained. Our pragmatic experience has been that about half of the probabilities are data dependent and readily obtained, 20 percent are non-critical, 25 percent are readily obtained from the structure of the program. The remaining 5 percent are sweaty and can require much analysis to obtain. However, since the analytical technique is fast, we can by parametrically examining values of these difficult probabilities, find out if they are indeed critical.
ANALYSIS

The analytical technique is a step-by-step node elimination based on what is sometimes called the "star-mesh transformation." We shall eliminate a node, and its associated incident and exident links and replace every combination of incident and exident link with equivalent links that bypass that node. There are three cases of importance—links in series, links in parallel, and loops. The situations for the series case is shown in Figure 5. The transformation equations for the series case are:

$$P_{ik} = P_{ik}' + P_{ik}''$$
$$\mu_{ik} = (P_{ik}' \mu_{ik}' + P_{ik}'' \mu_{ik}'')/(P_{ik}' + P_{ik}'')$$
$$\lambda_{ik} = (P_{ik}' \lambda_{ik}' + P_{ik}'' \lambda_{ik}'')/(P_{ik}' + P_{ik}'') + (\mu_{ik}' P_{ik}'' + \mu_{ik}'' P_{ik}')/(P_{ik}' + P_{ik}'') - \mu_{ik}$$

The transformations for the loop is shown in Figure 7. The equations are:

$$P_{ik} = P_{ik}/(1 - P_{ii})$$
$$\mu_{ik} = \mu_{ik}' + P_{ii}(1 - P_{ii})$$
$$\lambda_{ik} = \lambda_{ik}' + \lambda_{ii} P_{ii} (1 - P_{ii}) + \mu_{ii}^2 P_{ii}/(1 - P_{ii})^2$$

The algorithm proceeds as follows:

1. Select a node for removal—other than an entrance or an exit.
2. Apply the series equations to eliminate the node. This creates new links.
3. Combine parallel links into a single equivalent link by using the parallel equations.

Figure 6. The equations are:

$$P_{ik} = P_{ik}' + P_{ik}''$$
$$\mu_{ik} = (P_{ik}' \mu_{ik}' + P_{ik}'' \mu_{ik}'')/(P_{ik}' + P_{ik}'')$$
$$\lambda_{ik} = (P_{ik}' \lambda_{ik}' + P_{ik}'' \lambda_{ik}'')/(P_{ik}' + P_{ik}'') + (\mu_{ik}' P_{ik}'' + \mu_{ik}'' P_{ik}')/(P_{ik}' + P_{ik}'') - \mu_{ik}$$

The transformations for the loop is shown in Figure 7. The equations are:

$$P_{ik} = P_{ik}/(1 - P_{ii})$$
$$\mu_{ik} = \mu_{ik}' + P_{ii}(1 - P_{ii})$$
$$\lambda_{ik} = \lambda_{ik}' + \lambda_{ii} P_{ii} (1 - P_{ii}) + \mu_{ii}^2 P_{ii}/(1 - P_{ii})^2$$

The algorithm proceeds as follows:

1. Select a node for removal—other than an entrance or an exit.
2. Apply the series equations to eliminate the node. This creates new links.
3. Combine parallel links into a single equivalent link by using the parallel equations.
4. Eliminate loops.
5. Repeat until only entrances and exits remain.

For manual calculations it is best to represent the flow chart by a matrix. The outmost column and row of the matrix is removed, reducing its rank. We have written a program that performs these calculations, a sample of which is shown in Figure 8. We have here a rather complicated model if we take into account all the possible loops and such. The links are described by the names of the nodes they span. The node names can correspond to the labels in the original flow chart. The special nodes “GILA” and “ZEND” are included as programming conveniences. The output shown has the expected probability of 1 and a mean value of 2263 microseconds. Another example, involving a subroutine with three entrances and two exits yields a somewhat more complicated result and is shown in Figure 9.

The transformation equations can be derived on the basis of very weak assumptions. We assume a Markov model. We assume further that the running time for a link does not depend upon the running time of other links. The only thing that has to be assumed about the distributions representing a link is that they exist and have a first and second moment. Other than that, the equations are valid for any distribution. There are additional refinements to the process to distinguish between deterministic and probabilistic loops that will not be discussed here. Furthermore, the analytical method is also applicable to the determination of memory utilization and channel utilization.

To gauge the efficiency of the procedure; a 100 link program requires less than two seconds of 360/50
time. The running time of a 1000 link analysis is under 10 seconds.

REFERENCES

The algorithms described here were first developed by the author in 1964. The assumptions, however, were overly strong—i.e., required that all link distributions be Gaussian. A more formal derivation based on weak assumptions (i.e., the distribution exists and have first and second moments) is to be found in References 1, 2, as well as a more detailed discussion of non-Markov models in which the node probability depends on the previous history of the program. It is shown there that while the mean value is not affected by these assumptions, the standard deviation is. The algorithm as programmed, is accordingly modified. We have only presented the variance equations for the Markovian case. These equations can be readily shown to yield an upper bound for the variance.

1 P W ARMS JR
Derivation and verification of system 6403 mathematical formulas
Data Systems Analysts Inc 503-TR-3 December 15 1969
2 P W ARMS JR
Forthcoming MS thesis in computer and information science
University of Pennsylvania
3 W FELLER
An introduction to probability theory and its applications
John Wiley & Sons Inc Volume II Chapter XIV 1966
4 B BEIZER
Application manual, system 6403
Data Systems Analysts Inc 503-TR-2 August 22 1969
5 P W ARMS JR
Instruction manual, system 6403
Data Systems Analysts Inc 503-TR-1 August 22 1969
6 S E ELMAGHRABY
An algebra for the analysis of generalized activity networks
Management Science Volume 10 Number 3 April 1964

This paper represents a parallel development in a more general area. Elmaghraby treats the generalized network. It will be seen that the network treated here is his EXCLUSIVE-OR case.