NTS-1314

UCRL-53230
Distribution Categories UC-15, 20e, 32

REPRODUCED FROM
BEST AVAILABLE COPY

UCRL--53230

DE82 007920

"Stop": A Fast Procedure for the Exact Computation of the Performance of Complex Probabilistic Systems

G. C. Corynen

Manuscript date: January 1982

DISCLAIMER

NOTICE

PORTIONS OF THIS REPORT ARE ILLEGIBLE. It has been reproduced from the best available copy to permit the broadest possible availability.

LAWRENCE LIVERMORE LABORATORY
University of California • Livermore, California • 94550

Available from National Technical Information Service • U.S. Department of Commerce
5285 Port Royal Road • Springfield, VA 22161 • [illegible] • Microfiche $3.50

HYDROLOGY DOCUMENT NUMBER 105

THIS DOCUMENT HAS BEEN REPRODUCED
FROM NRC MICROFICHE HOLDINGS BY
THE DOCUMENT MANAGEMENT BRANCH,
DIVISION OF TECHNICAL INFORMATION
AND DOCUMENT CONTROL

FOR INFORMATION ABOUT MICROGRAPHIC SERVICES
INCLUDING EQUIPMENT CALL: 28137

FOR DETAILS ON:
MICROFICHE BLOWBACK SERVICES CALL: 28076
APERTURE CARD BLOWBACK SERVICES CALL: 28076
MICROFICHE OR APERTURE CARD DUPLICATION CALL: 28076
FOR EQUIPMENT REPAIRS CALL: 28397

PORTIONS OF THIS DOCUMENT ARE ILLEGIBLE

CONTENTS

Abstract 1
Introduction 1
Notations and Definitions 2
Introduction to the Algorithm 4
The Algorithm 5
An Example 6
Complexity Analysis 8
Comparison with Other Methods 8
Conclusions and Future Work 10
Acknowledgments 10
References 11

"Stop": A Fast Procedure for the Exact Computation of the Performance of Complex Probabilistic Systems

ABSTRACT

A new set-theoretic method for the exact and efficient computation of the probabilistic performance of complex systems has been developed. The core of the method is a fast algorithm for disjointing a collection of product sets which is intended for systems with more than 1,000 components and 100,000 cut sets. The method is based on a "divide-and-conquer" approach, in which a multidimensional problem is progressively decomposed into lower-dimensional subproblems along its dimensions. The method also uses a particular pointer system that eliminates the need to store the subproblems by only requiring the storage of pointers to those problems.

Examples of the algorithm and the divide-and-conquer strategy are provided, and comparisons with other significant methods are made. Statistical complexity studies show that the expected time and space complexity of other methods is $O(me^n)$, but that our method is $O(nm^3 \log(m))$. Problems which would require *days* of Cray-1 computer time with present methods can now be solved in *seconds*. Large-scale systems that can only be approximated with other techniques can now also be evaluated exactly.

INTRODUCTION

The evaluation of the probabilistic performance of systems is an essential step in many systems studies. In risk analyses for instance, the risk distribution function must be computed. In reliability analyses, such as those performed on nuclear reactors, system reliability must be evaluated.

When systems consist of many highly interactive components, the exact evaluation of their performance often exceeds the capability of large computers, and approximation methods are employed. However, in high-risk situations where the costs of system failure are prohibitive, approximation methods are often unsatisfactory, and exact answers are needed. In some cases, of course, the inaccuracies incurred as a result of such approximations are overwhelmed by the inaccuracies in the input data, which are often highly subjective and speculative and exact answers are ruled out. In such cases, however, a sensitivity analysis is often required to determine the sensitivity of performance estimates to changes in input structure, and such an analysis should not depend upon approximations made purely for computational purposes.

The estimation of system performance from the performance of the components and their interconnections often proceeds in two steps. First, a performance rejection region for the system is defined and the combinations of the component states which cause system rejection are determined. These are the "cut sets" of the system. Next, the probability of occurrence of *at least* one of these cut sets is computed from the probabilities associated with the components and the cut sets. Numerous methods have been devised to complete both of these steps,[1,2] but none of these appear satisfactory in handling large-scale problems consisting of more than 1,000 components and involving more than 100,000 cut sets.

An examination of the available methods appears to indicate that the second step deserves particular attention at this time. A recent paper by Abraham[3] significantly improved the efficiency of probability computation methods and may be used

as a standard against which new developments may be contrasted.

A complexity analysis of Abraham's algorithm reveals that, for a system with n components and m cut sets (minimal or not), typically $O(mn^{n-3})$ computer operations and memory locations are required. When a system with 1,000 components and 100,000 cut sets is considered, approximately 5.8×10^{119} operations are required, far beyond any conceivable computer.

The purpose of the present paper is to present a method whose expected time and space requirements are $O(nm^3 \log(m))$. The method is called "STOP" ("*S*uperfast *T*echnique for *O*btaining *P*robabilities"); it represents a set-theoretic viewpoint in which system success or failure is represented as a union of intersections of dependent events, and a disjoint representation for that union is sought. Although this representation is analogous to a sum of product form in a Boolean problem, the set-theoretic view provided considerably more insight into the structure of the computational problem, particularly since a holistic approach is taken where all cut sets are examined and transformed simultaneously via a divide-and-conquer approach to obtain the disjoint form. This is probably the most important feature that distinguishes this parallel procedure from sequential procedures such as Abraham's, in which each cut set is processed individually in some order.

In this paper, we offer a sketch of the complexity analysis and refer the reader to Ref. 10 for a more thorough report of our research. The algorithm itself, however, is presented in considerable detail and with appropriate examples, so that interested readers could implement the algorithm in a manner suitable to their own computational environment, and with minimal interpretation. Our final report will provide a line by line description of our FORTRAN implementation for those workers who are interested in implementing the algorithm without modification.

Although this report deals with coherent[11] systems only, the extension to non-coherent systems is straightforward and does not alter the complexity of the algorithms discussed here.

NOTATION AND DEFINITIONS

Consider a system with n components $C^n = \{C_1, C_2, C_3, \ldots, C_j, \ldots, C_n\}$. With each component C_j associate a binary state variable e_j such that

$e_j = 1$, if C_j is in the *working state* and
$e_j = 0$, if C_j is in the *failed state*.

We consider the *system state variable* as an n-tuple $s = (e_1, e_2, \ldots, e_j, \ldots, e_n)$ of state variables and a *system state* as an n-tuple of *component states* in the *system state space* $Q^n = E_1 \times E_2 \times \ldots \times E_j \times \ldots \times E_n$, denoted by $\mathop{\times}\limits_{j=1}^{n} E_j$, where $E_j = \{0,1\} = U$, the *component state space*, and the coordinates are $j = 1, 2, \ldots, n$. For example, the system state $(0_1, 1_2, 0_3, 1_4, 1_5, \ldots, 0_n)$ represents the condition where C_1 is failed, C_2 is working, C_3 is failed, ..., and C_n is failed. It is straightforward to apply this n-tuple characterization to graphs in order to describe paths from any source node to any sink node, and we shall occasionally use graphs to illustrate our discussion.

Another convenient notion is that of a *simple set*. A simple set S is a product subset of the state space. Thus

$$S = \mathop{\times}\limits_{i=1}^{n} A_i \subset \mathop{\times}\limits_{j=1}^{n} E_j .$$

Simple sets may be used to characterize the causes of system failure (or success), more specifically, failure cuts in graphs or cut sets in fault trees. The simple set $\{1\}_1 \times \{0\}_2 \times U_3 \times \ldots \times U_n$, for instance, represents the condition where component C_1 is working, component C_2 is failed, component C_3 is failed or working, ..., component C_n is failed or working.

The set U in coordinates 3 and n indicates a "don't care" condition in those positions. To simplify notation, and if no danger of confusion exists, we shall represent a simple set more conventionally as an n-tuple $S = (e_1, e_2, \ldots, e_n)$, where e_i is now a ternary variable with possible values 0, 1, or -. The simple set $\{1\}_1 \times \{0\}_2 \times U_3 \times \ldots \times U_n$ above would

then be represented as an n-tuple (1, 0, -, ..., -), where the hyphen represents a don't care condition in the corresponding coordinate.

We shall occasionally represent a collection $\{S_i : i = 1, 2, \ldots, m\}$ of m n-dimensional simple sets (n sets) in the matrix form of Fig. 1. In that representation, $E_{i,j}$ is the jth *coordinate value* of simple set S_i. As a general comment, when the simple sets represent causes of system failure and the value of coordinate j of some simple set S_i equals U, then the corresponding component C_j is *irrelevant* to system failure for that particular cause S_i. If $E_{i,j} = U$ for all indices $i = 1, 2, \ldots, m$, then C_j is irrelevant[11] to the system, not only to system failure. If $E_{i,j} = 0$ for more than one i, then $E_{i,j}$ is a *common-mode event*, and component C_j is *shared* by several simple sets. As the number of indices i for which $E_{i,j} = 0$ increases, for some fixed j, the influence of component C_j upon the system failure increases. In the extreme case when $E_{i,j} = 0$ for all i, the failure of C_j is *necessary* for the failure of the system.

A final but important mathematical comment concerning product sets and their probabilities should be made. Probability distributions of random variables are usually determined by reflecting or mapping back to the sample space via the inverse of the random variables[12-14]. The probabilities of product sets—such as simple sets—are thus

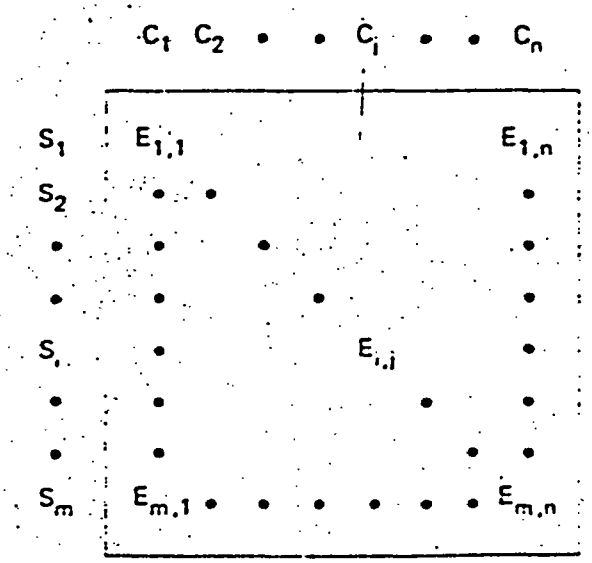

FIG. 1. Matrix representation of a collection of m n-dimensional simple sets $\{S_i : i = 1, 2, \ldots, m\}$.

equal to the probabilities of *intersections* of subsets of the sample space. As an example, consider two random variables $X_1 : \Omega \to R$ and $X_2 : \Omega \to R$. Then $P(X_1 = a, X_2 = b) = P(X_1^{-1}\{a\} \cap X_2^{-1}\{b\})$. In the next section, where we describe the algorithm, we shall use this fact to exploit the set-theoretic *analogue* of the absorption and expansion operations of Boolean algebra.[7] We shall now recall some standard set-theoretic facts.

If U is the universal set, and A and B any two sets, then

$$A \cup U = U \text{ and } A \cap U = A \quad (1)$$

Furthermore,

$$\bigcup_{i=1}^{n} B \cap A_i = B \cap \bigcup_{i=1}^{n} A_i \quad (2)$$

for any collection $\{A_i : i = 1, 2, \ldots, n\}$ of sets. We also have the expansion operation

$$A = A \cap B \dotplus A \cap \bar{B} \quad (3)$$

where $\dotplus$ denotes disjoint union and $\bar{B}$ is the complement of B and the containment (absorption) operation is

$$A = A \cup (A \cap B) \quad (4)$$

Consider some sequence $\{A_i\}$ of simple sets indexed by a set index set $\{i\}$ and a coordinate index set $\{j\}$. If for some $k \in \{j\}$, $A_{ik} = A_k$ for all $i \in \{i\}$, then

$$\bigcup_{i \in \{i\}} \bigcap_{j \in \{j\}} A_{ij} = \bigcup_{i \in \{i\}} A_k \cap \bigcap_{\substack{j \in \{j\} \\ j \neq k}} A_{ij}$$

$$= A_k \cap \bigcup_{i \in \{i\}} \bigcap_{\substack{j \in \{j\} \\ j \neq k}} A_{ij} \quad (5)$$

This factoring, which reduces the n-dimensional problem to an (n−1)-dimensional problem (the kth coordinate removed), will be important in our development. In practice, of course, a weaker condition is usually met in which, for any fixed coordinate k and set index i_0, $A_{i_0 k} = A_{i_0 j} = A_k$ for some subsequence $\{j\}$ of the index sequence $\{j\}$. By combining the factoring and the expansion operations this condition is exploited recursively in our algorithm to obtain the following decomposition:

$$\bigcup_{i\in\{i\}} \bigcap_{j\in\{j\}} A_{ij} = \bigcup_{i\in\{i\}_k} A_k \cap \bigcap_{\substack{j\in\{j\}\\ j\neq k}} A_{ij} \cup \bigcup_{i\in\{i\}-\{i\}_k} A_k \bigcap_{\substack{j\in\{j\}\\ j\neq k}} A_{ij} + \bar{A}_k \bigcap_{\substack{j\in\{j\}\\ j\neq k}} A_{ij}$$

$$= A_k \cap \bigcup_{i\in\{i\}} \bigcap_{\substack{j\in\{j\}\\ j\neq k}} A_{ij} + \bar{A}_k \cap \bigcup_{i\in\{i\}-\{i\}_k} \bigcap_{\substack{j\in\{j\}\\ j\neq k}} A_{ij} \quad . \qquad (6)$$

where $\bar{A}_k$ is the complement of A_k.

INTRODUCTION TO THE ALGORITHM

Consider the failure matrix of Fig. 1 for an n-component system with m causes of failure S_i, where each S_i is a simple set characterizing a system failure condition. The probability of system failure is then

$$\text{Prob(system failure)} = P(\bigcup_{i=1}^{m} S_i) \qquad (7)$$

The simple sets S_i are rarely disjoint. Therefore

$$P(\bigcup_{i=1}^{m} S_i) \neq \sum_{i=1}^{m} P(S_i)$$

usually, and it is natural to seek some transformation of the collection $\{S_i : i = 1, 2, \ldots, m\}$ into another collection $\{S_j : j = 1, 2, \ldots, m'\}$ of simple sets such that the S_j are disjoint, and such that

$$P(\bigcup_{i=1}^{m} S_i) = P(\bigcup_{j=1}^{m'} S_j) = \sum_{j=1}^{m'} P(S_j) \qquad (8)$$

Several methods exist[1,3,5,7] for obtaining this disjoint representation, but a complexity study of Abraham's approach,[1] which appears to dominate the others, shows that they are of exponential order in n, the dimension of the system, and asymptotically linear in m. We present a method which is $O(nm^3 \log(m))$.

Several goals were sought with the new method. In addition to efficient time and space behavior, we needed good asymptotic characteristics. In particular, we needed quick, monotonic and controlled convergence to the correct answer. Specifically, we felt it was important to determine a critical and smallest number k for any problem, such that the divide-and-conquer process could be guaranteed to stop before iteration k. We also endeavored to structure the algorithm in such a way that tight stopping rules could be devised for super-large problems whose answers could not be obtained exactly. All of these goals have been met by the present algorithm.

There are several reasons for the efficiency of our method. First, a "holistic" approach was taken, in which all failure conditions are analyzed as a whole, rather than sequentially as in other methods. Referring to Fig. 1 once again, the entire matrix structure is exploited by operating on both the coordinates and set indices via the containment and expansion operations discussed earlier.

Second, a particular type of divide-and-conquer strategy is employed where the initial problem (the initial failure matrix) is decomposed into two disjoint and simpler problems (failure matrices) along some carefully chosen coordinate. This is done in such a way as to minimize the size of both offspring problems and ensure that these two problems are collectively much easier to solve than the parent problem. This process is continued until a collection of matrices is obtained; each consists of only one simple set that is disjoint from any other, and the entire collection is thus considerably simpler to solve than the original problem.

Third, but not least, a new method for implementing the algorithm on a computer has been devised. Whereas the two previous items dealt with time economy, this third item was an essential step in achieving economy in space. Without it, space requirements might have grown exponentially.

The method employs a "pointer" system which allows us to keep track of all of the offspring problems and their simple sets without having to

store or manipulate the simple sets themselves. Only the pointers to the initial problem sets, and the functional symbols representing how these sets were transformed, have to be retained in memory.

Roughly speaking, there are three major steps to the algorithm.

Step 1—Find the Largest Simple Sets

The purpose of this step is to start the decomposition process by finding the largest contributors first, to accelerate convergence. Presently, largeness is measured by the number of don't cares (-) in the simple sets. (There usually are several such largest sets, but we do not discuss this here.)

Step 2—Find the Coordinate Along Which to Decompose the Problem Into Two Subproblems

This is accomplished by choosing the coordinate with the greatest number of zeros in such a way that the chosen coordinate must have a zero in the position corresponding to the simple set chosen in step 1. This particular selection process is important because it ensures that large sets are not broken up into disjoint pieces along the chosen coordinate. When large sets are retained, the chances for containment are increased, and, as other sets are absorbed, the problems quickly become smaller. Another important peculiarity in this step is that, if there are ties and several coordinates meet the test, the coordinate is selected for which the simple sets having don't cares in that position also have the greatest number of don't cares. This ensures that, during any split of one problem into two subproblems (offspring) in accordance with Eq. (5), each new subproblem inherits at least one large simple set.

Step 3—Eliminate Simple Sets Which are Contained in Other Simple Sets

Referring to Eq. (6) again, some of the sets which constitute the new left-hand problem—the left expression of the right side of Eq. (6)—could be contained in others as a result of the splitting operation and may be eliminated. The purpose of this step is to test for containment and to eliminate such sets.

THE ALGORITHM

In this section we present a detailed description of our algorithm. Our final report will provide a line-by-line description of the flow diagram and its FORTRAN implementation. We assume that the m failure causes (failure paths, cut sets) for a system with n components are available. Further research is under way to merge the path and tree computation with the probability computation.

Algorithm

1. Start with the collection of n-dimensional simple sets $S = \{S_i : i = 1, 2, \ldots, m\}$.
2. Remove all coordinates where *all* sets in S have either a 0 or a don't care (-) and store them for reference at the end of the problem. (These coordinates are irrelevant for the algebraic computations in this algorithm.)
3. Remove all redundant coordinates (those for which at least one other coordinate exists with exactly the same entries). Store these removed coordinates for reference at the end of the problem.
4. Eliminate every set in S that is a subset of some other set in S.
5. Find all simple sets with the maximal number of don't cares.
6. Find the set of coordinates that have a 0 in at least one of the simple sets found in (5).
7. From this set of coordinates, retain only those coordinates in which the greatest number of simple sets in (1) have a 0.
8. For each coordinate retained in (7), determine the maximum number of don't cares that can be found in any simple set with a don't care in that coordinate.
9. Retain only those coordinates whose number found in (8) is not exceeded by any other number found in (8).

10. From the coordinates retained in (9) select the coordinate with the lowest index. This is the *optimal* coordinate j^*, along which to decompose the parent problem into two offspring problems.
11. Replace the parent problem with the two new subproblems in accordance with Eq. (6). The first (left-hand) problem consists of all simple sets of the parent problem with coordinate j^* set equal to 0. The second (right-hand) problem consists of all those simple sets from the parent problem which have a don't care in coordinate j^*, but with all those don't cares set to 1: (analogous to $\bar{A}_{ij}$).
12. For the left-hand subproblem, remove every simple set which is contained in any other.
13. If either of the new problems consists of a single set, store this simple set as a disjoint product of the procedure.
14. If either of the new problems consists of more than one simple set, remove superfluous coordinate as in step (2), and put it at the bottom of the problem list.
15. Take the next problem from the problem list and return to (5). If the problem list is empty we have all the disjoint products and continue with (16).
16. Compute the system probability as the sum of the probabilities of the disjoint simple sets obtained and stored in (13) with restoration of the coordinates removed in steps (2) and (3).

AN EXAMPLE

Consider the directed graph or network of Fig. 2 and its failure matrix in Fig. 3. The problem does not have irrelevant or redundant coordinates and no simple set is a subset of any other. So we may proceed with step 5 of the algorithm, which produces S_1, S_2, and S_3. Step 6 produces coordinates 1, 2, 4, and 5. Step 7 yields coordinates 1, 2, 4, and 5 again. Step 8 yields numbers 3, 3, 3, and 3 respectively, all coordinates are retained in step 9 and $j^* = 1$ is the optimal coordinate found in step 10.

The two new subproblems and all subsequent subproblems are shown in Fig. 4. The first left problem consists of the original four sets with coordinate 1 set to 0. After the containment test only S_1 and S_2 are left, as shown. Indeed, S_1 is contained in S_1, since S_1 has a don't care wherever S_3 has a don't care. Similarly, S_3 is contained in S_2.

On the right-hand side, only the two sets S_3 and S_4 with coordinate 1 set to 1 were produced, since only S_3 and S_4 had a don't care in coordinate position 1.

The left-hand process iterates until it "stops" with simple set S_1, whose first and second coordinates are set to 0, and with S_2, whose first coordinate also equals 0, but whose second equals 1, thus disjointing it from S_1. The final left-hand side products are then (0, 0, -, -, -) and (0, 1, -, -, 0).

The right-hand problem iterates until it "stops" with two simple sets S_3 and S_4. The first has coordinates 1, 4, and 5 set equal to 1, 0, and 0, respectively, and the second has these coordinates set to 1, 0, and 1, respectively. We thus obtain two other disjoint products (1, -, -, 0, 0) and (1, 0, 0, 0, 1). All four final products are shown in the resultant failure matrix; these are obviously disjoint, since any two simple sets have a 0 in at least one coordinate where the other has a 1, or conversely.

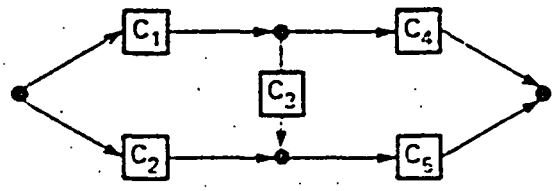

FIG. 2. A simple directed network may be used to illustrate the procedure.

	C_1	C_2	C_3	C_4	C_5
S_1	0	0	-	-	-
S_2	0	-	-	-	0
S_3	-	0	0	0	-
S_4	-	-	-	0	0

FIG. 3. The failure matrix for the problem of Fig. 2.

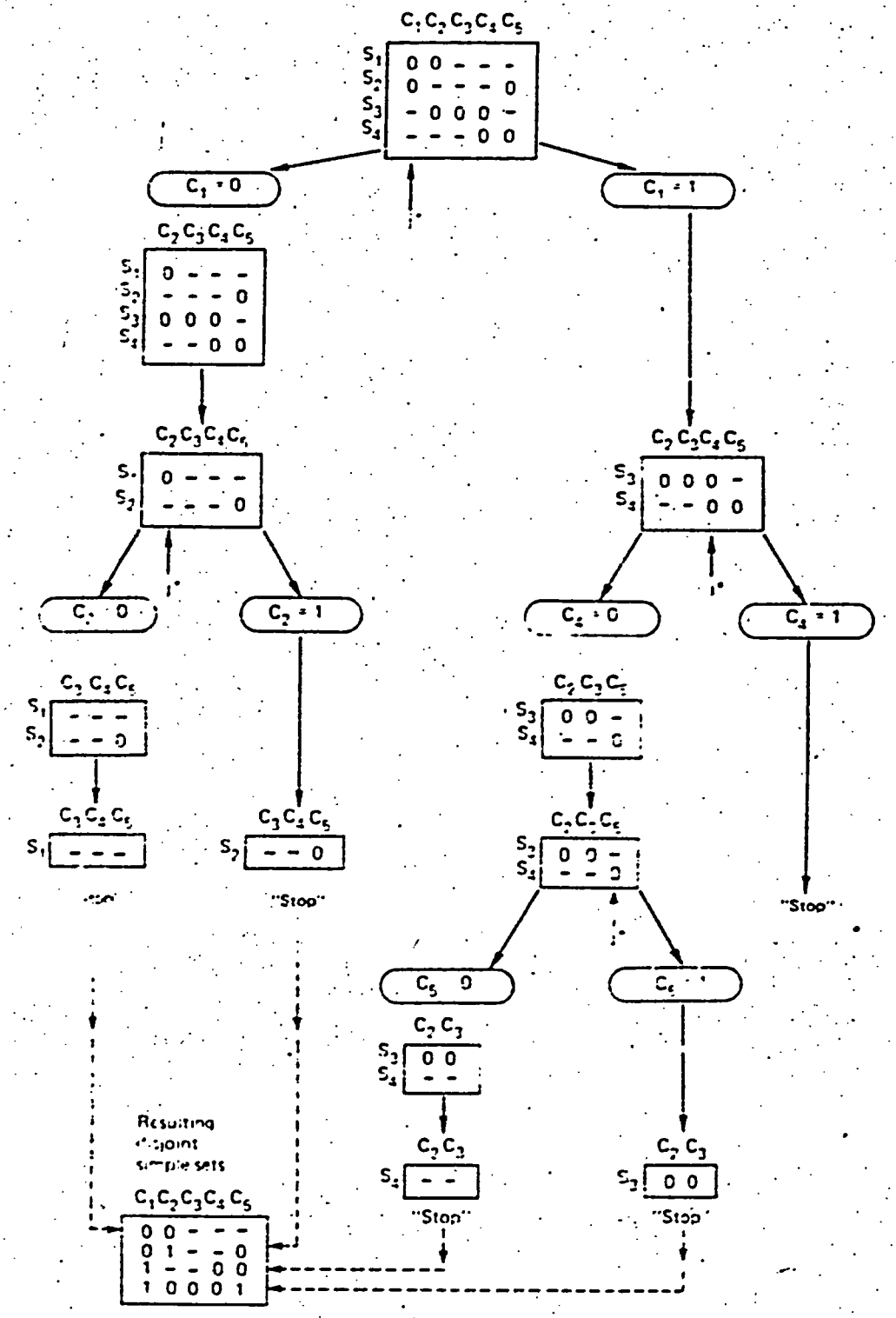

FIG. 4. Application of the procedure to the problem of Fig. 2.

COMPLEXITY ANALYSIS

In this section we summarize the results of two types of complexity studies which were undertaken: a statistical study and a worst case study. Both studies are still underway and will be discussed in our final report.[10] The statistical study is basically a "reasonable worst case" analysis in which, at each step of the algorithm, an "expected worst case" outcome is assumed, consistent with earlier expected outcomes. The results of this analysis have been partially validated by manually working out general medium-sized and randomly chosen problems. Although these results are definitive, no final conclusions will be reached until our ongoing experiments have been completed.

The most important conclusion of these analyses is that, for a system with n components and m failure conditions, the expected *time and space requirements increase as* $O(nm^3 \log(m))$. Although we shall make further comparative comments in the next section, it is important to report that our analyses and experiments have demonstrated that other methods typically require $O(me^{n-3})$ operations and memory locations to compute the same answer.

We conclude this section with an example which is intended to illustrate how the divide-and-conquer approach is used, and how the total work for decomposing a typical problem is computed. Referring to Fig. 5, we start with a problem P for which m = 2000, n = 100, and don't cares are distributed at random, with probability $P_{\cdot} = 0.9$ of finding a don't care in any single position in the failure matrix. This is a typical distribution for complex systems. We thus characterize a problem with a triple (m, n, $P_{\cdot}$).

The work required to break the initial problem into two subproblems is 9000K operations, as shown (each operation is a one-bit comparison). The resultant problems have characteristics (500, 99, 0.9) and (1500, 99, 0.9), respectively. Proceeding to the second-level splits, we obtain a total of 8643K operations to obtain the four new subproblems. Proceeding inductively to the bottom of the hierarchy and adding all the work required to get there, we obtain a cumulative work requirement of 454,500K operations.

COMPARISON WITH OTHER METHODS

Starting with the often-quoted method of Rai,[5] we should point out that finding the "exclusive OR products" when m is large (consider m = 10^5) is difficult because taking complements of branches or paths generates many new paths. In terms of simple sets, the complement of a single set is rarely a single simple set, particularly if the system is large (large value of n). This uncontrolled growth process is the most serious limitation to all algorithms of this type.

The Abraham method,[1] which appears to dominate the business, is an eminent example of this serial or sequential way of thinking. Based on the fact that, for any two sets S_1 and S_2, $S_1 \cup S_2 = S_1 \cup (S_2 - S_1)$, such methods require the computation of set-theoretic differences $P_i - P_j$ between paths P_i and P_j, if these are not disjoint. In the simple example where P_1 = (-, -, 0, 0) and P_2 = (0, -, -, -), two new paths $(P_1 - P_2)_1$ = (0, -, 1, -) and $(P_1 - P_2)_2$ = (0, -, -, 1) are produced as offspring from the difference operation $P_1 - P_2$. These products multiply *exponentially* in the general case as new differences are taken.

We should point out that, until very recently, the first version of our algorithm was very similar to the others discussed here. Its complexity in n was not well understood until extensive experiments were run, and the exponential cost of a sequential, or on-line, approach was discovered. A parallel or holistic approach was then pursued, and the present algorithm is a product of that research. A related approach was taken by Derby,[6] but that approach, although more efficient than the others mentioned here, does not sufficiently exploit the structure of a collection of simple sets, its asymptotic behavior is not controlled, statistical independence between components is assumed, and memory requirements appear severe. The asymptotic behavior is very important for super-large problems in which stopping rules must be employed. In addition to these differences in time behavior, we devoted special attention to space (memory) requirements as well, and a "pointer" approach was developed to keep track of all the products of the computation. This approach allows

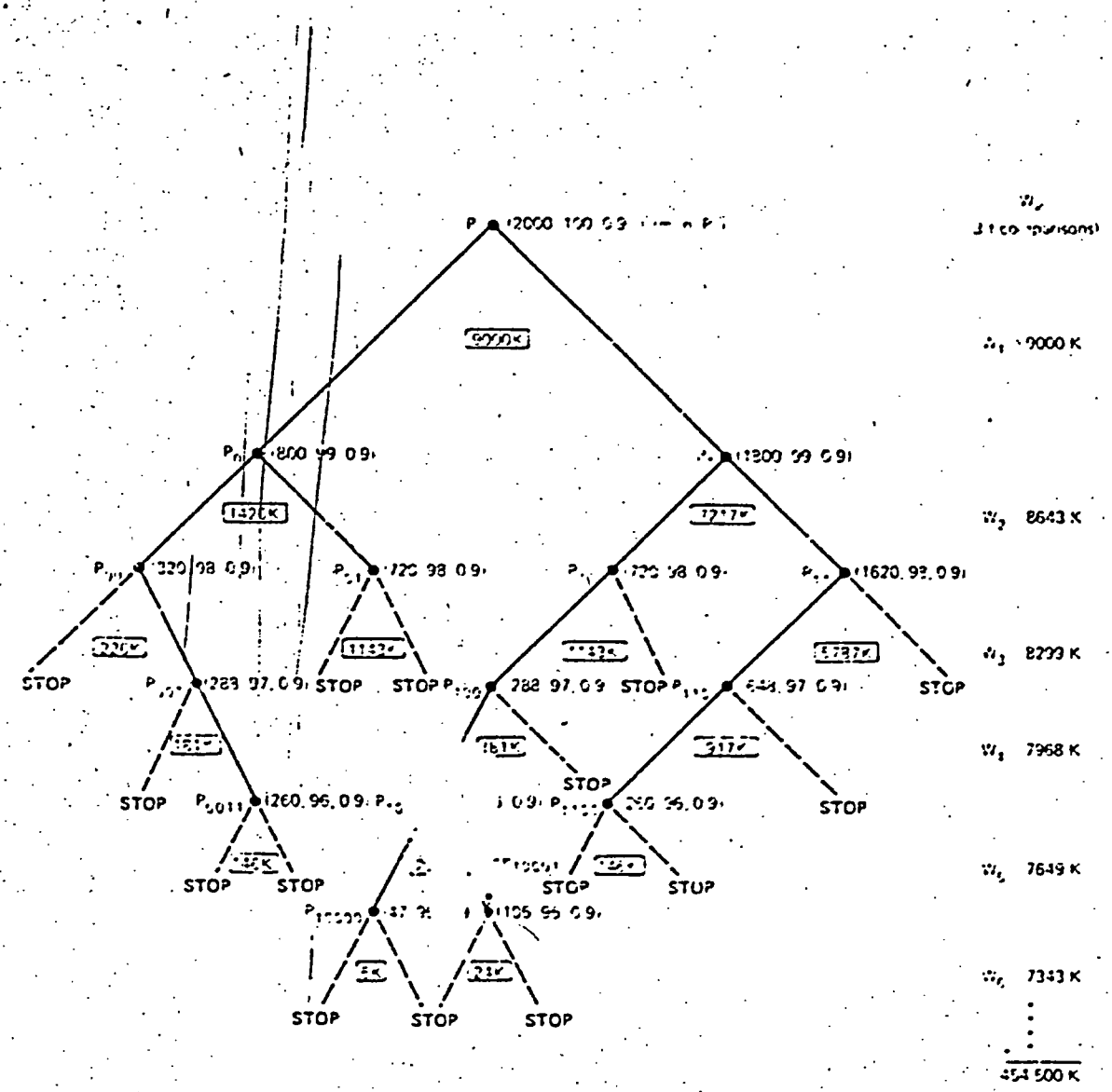

FIG. 5. Illustrating the total work for a medium-sized problem.

us to keep track of the simple set offspring without storing simple sets themselves, but by simply keeping track of the transformation process which gave birth to them. This is analogous to storing a function as a functional symbol rather than storing it as a table of pairs.

To provide a quantitative comparison of the complexity of predominant methods, consider our earlier example where m = 2000, n = 100 and P = 0.9, and a machine is used which takes 1 nanosecond per bit comparison. The inclusion-exclusion essentially varies as 2^n and is not very sensitive to m. For that method, approximately 4×10^{13} years would be required to obtain a disjoint representation. The on-line, or sequential, method of Abraham and others typically would require 27.74 days. Our present method would require approximately 1 second to get the correct answer.

CONCLUSIONS AND FUTURE WORK

A new method for computing the performance of probabilistic systems is described. Although the method is economical for even small systems, it is particularly efficient in time and space when the systems are large. A typical reliability computation for an n-component system with m causes of failure requires approximately $O(nm^2 \log(m))$ computations and memory locations, compared with $O(m^{2^n})$ for other available methods.

ACKNOWLEDGMENTS

This research was supported by the Engineering Research Division of the Lawrence Livermore National Laboratory, and the author appreciates this support. The author also wishes to thank A. Parzialo, J. Reiter, P. Lorton, and C. Patenaude for valuable discussions in general and for the original complexity analysis and implementation of the algorithm, in particular. The excellent secretarial assistance provided by E. Rodenhizer and the technical editing contributions of C. Juad were essential in completing this report.

REFERENCES

1. J.A. Abraham. "An Improved Algorithm for Network Reliability." *IEEE Transactions on Reliability* R-28, No. 1, 58-61 (April 1979).
2. E.V. Krishnamurthy and G. Komissar. "Computer-Aided Reliability Analysis of Complicated Networks." *IEEE Transactions on Reliability* R-21, 86-89 (May 1972).
3. Y.H. Kim, K.E. Case, and P.M. Ghare. "A Method for Computing Complex System Reliability." *IEEE Transactions on Reliability* R-21, 215-219 (November 1972).
4. M.O. Locks. "System Reliability Analysis: A Tutorial." *Microelectron Reliab.* 18, 335-345 (1978).
5. S. Rai and K.K. Aggarwal. "An Efficient Method for Reliability Evaluation of a General Network." *IEEE Transactions on Reliability* R-27, No. 3 (August 1978).
6. Applied Decision Analysis, Inc., *Definition, Development, and Demonstration of Analytical Procedures for the Structured Assessment Approach*, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-15012 (April 1979).
7. R.G. Bennetts. "On the Analysis of Fault Trees." *IEEE Transactions on Reliability* R-24, No. 3 (August 1975).
8. K.K. Aggarwal, K.B. Misra, and J.S. Gupta. "A Fast Algorithm for Reliability Evaluation." *IEEE Transactions on Reliability* R-24, No. 1 (April 1975).
9. D.M. Rasmuson, and N.H. Marshall. "FATRAM-A Core Efficient Cut-Set Algorithm." *IEEE Transactions on Reliability* R-27, No. 4 (October 1978).
10. G.C. Corynen, A. Parziale, and E. Reiter. *A Fast Procedure for the Exact Computation of the Performance of Complex Probabilistic Systems*, Lawrence Livermore National Laboratory, Livermore, CA, UCRL report in progress, 1981.
11. R.E. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing Probability Models* (Holt, Rinehart and Winston, Inc., New York, NY, 1975).
12. L. Breiman. *Probability* (Addison-Wesley, Reading, MA, 1968).
13. K.L. Chung. *Probability Theory* (Academic Press, New York, NY, 1974).

END

DATE FILMED

05/26/82