Discrete Applied Mathematics 37/38 (1992) 489-500
North-Holland

Generating the states of a binary stochastic system

D.R. Shier

Department of Mathematical Science, Clemson University, Clemson, SC 29634, USA

E.J. Valvo and R.E. Jamison

Department of Mathematical Science, Clemson University, Clemson, SC 29634, USA

Received 9 September 1989
Revised 22 November 1990

Abstract

Shier, D.R., E.J. Valvo and R.E. Jamison, Generating the states of a binary stochastic system, Discrete Applied Mathematics 37/38 (1992) 489-500.

An important aspect of planning a communication or distribution system is assessing its performance when the components are subject to random failure. Since exact calculation of stochastic performance measures is usually difficult, the behavior of the system can instead be approximated by generating a subset of all system states. Specifically, we consider here the generation of states of a binary stochastic system in order of nonincreasing probability. Such an ordering ensures that maximum coverage of the state space (in terms of probability) will be obtained for a specified number of generated states. We identify a particular discrete structure, a distributive lattice, underlying this generation problem, and use this structure to guide an algorithm for generating in order the states of the given system. Computational results suggest that the proposed method improves on existing algorithms for this generation problem.

Keywords. Algorithm, lattice, network, partial order, performance measures, reliability.

1. Introduction

It is frequently of interest to evaluate the performance of a system, such as a communication or distribution system, which is composed of failure-prone components. Since the exact calculation of most realistic performance measures (throughput, delay, reliability) is in general NP-hard [2], there is reason to investigate methods for approximating such measures. One approximation strategy involves generating the states of a given binary stochastic system in order of nonincreasing probability. As shown by Li and Silvester [5], this can in fact be accomplished without examining

the entire state space. Once this generation procedure has been carried out, it is not difficult to obtain bounds on various performance measures for the system [5]. An attractive feature of this approach is that lower and upper bounds can be generated at each step, and the whole process of generating states in nonincreasing order can be continued until the bounds become sufficiently close.

Li and Silvester discussed the application of this approach to several performance measures arising in the analysis of computer networks. They provided an $O(n^2k+nk\log k)$ algorithm for generating the k most probable states of a system with n components, where k must be specified in advance. An improved algorithm, that does not require k to be specified in advance, was subsequently given by Lam and Li [4]. This algorithm, with an $O(nk+k\log k)$ time complexity, provides an order of magnitude improvement over the previous procedure.

Rather surprisingly, there is an elegant algebraic structure (a distributive lattice) underlying the state space, as discussed in Section 2. This structure can be exploited to produce an algorithm for generating the most probable states of the system in order, without the need to examine the entire state space. Section 3 discusses the details of this algorithm and compares it to the method given in [4]. In addition, the worst-case computational complexity of the algorithm is shown to be related to a certain algebraic invariant of the lattice. Computational results with implementations of the various algorithms are described in Section 4.

2. Structure of the state space

Consider a system with n failure-prone components $1, 2, \ldots, n$, assumed to fail independently of one another. Suppose that component i is found in the operational mode with probability p_i, and in the failed mode with probability $q_i = 1-p_i$. Without loss of generality it can be assumed that each $p_i \geq \frac{1}{2}$, since otherwise the analysis can proceed using $1-p_i$ in place of p_i. Also we will number the components from least reliable to most reliable, giving

$$\tfrac{1}{2} \leq p_1 \leq p_2 \leq \cdots \leq p_n \leq 1. \tag{1}$$

Consequently, the ratios $R_i = q_i/p_i$ are placed in nonincreasing order:

$$1 \geq R_1 \geq R_2 \geq \cdots \geq R_n \geq 0. \tag{2}$$

Each *state* of the system corresponds to a subset X of the set of components $\{1, 2, \ldots, n\}$, where the elements of X represent the *failed* components occurring in that state. The *state space*, containing all states for n components, is denoted by $S = S_n$, where $|S| = 2^n$. Assuming independent failures, each state $X \in S$ has probability $p(X)$ given by

$$p(X) = \prod_{i \in \bar{X}} p_i \prod_{j \in X} q_j = \left(\prod_{i=1}^{n} p_i \right) \prod_{j \in X} R_j = \left(\prod_{i=1}^{n} p_i \right) R(X), \tag{3}$$

where $R(X)$ is the *R-value* of state X. The special case when $X = \emptyset$ (that is, when all n components are operational) is assigned the R-value 1, consistent with equation (3).

The objective here is to generate the states X in order of nonincreasing probability $p(X)$. It will be seen that simply knowing the ordinal information conveyed by (1) provides considerable information about the relative probabilities of the various states. To clarify this connection, we define, for any two states $X_i, X_j \in S$, the order relation $X_i \geqq X_j$ if $p(X_i) \geq p(X_j)$ holds for *all* values p_m satisfying (1). It is then straightforward to show the following.

Property 2.1. *The set S of all states forms a partially ordered set* $(S, \geqq)$.

We can represent this partially ordered set $(S, \geqq)$ as a (directed) graph in which each state $X \in S$ corresponds to a node of the graph. If state $X = \{i_1, i_2, \ldots, i_k\} \in S$, where $i_1 < i_2 < \cdots < i_k$, then the associated node is labeled $i_1 i_2 \cdots i_k$. The distinguished state $X = \emptyset$ corresponds to node 0. If $X_i \geqq X_j$ holds, then an arc is drawn between the corresponding two nodes in the graph.

Note that by equation (3) comparing $p(X_i)$ with $p(X_j)$ for all p_m satisfying (1) is equivalent to comparing $R(X_i)$ with $R(X_j)$ for all R_m satisfying (2), so that the ordering of R-values in (2) completely determines the arcs of this graph. For example, consider the partially ordered set $(S_3, \geqq)$. An arc extends from node 2 to node 13 in this graph, since $R_2 \geq R_3 \geq R_1 R_3$. The graph of the partially ordered set $(S_3, \geqq)$ can be displayed more clearly by removing all arcs that are implied by transitivity, yielding its Hasse diagram, shown in Fig. 1.

In general, the Hasse diagram for n components is comprised of 2^n nodes, one for each state of the system. The 2^n nodes are arranged into $n+1$ levels such that level j, for $j = 0, \ldots, n$, contains $\binom{n}{j}$ nodes. Each node on level j corresponds to a state with exactly j failed components. The arcs of the Hasse diagram can be separated into two different categories: arcs within a given level and arcs between two consecutive levels. Within each level j $(j = 1, \ldots, n-1)$, an arc extends from node $k_1 \cdots k_{i-1} k_i \cdots k_j$ to node $k_1 \cdots k_{i-1} k_i + 1 \cdots k_j$ provided $k_i + 1$ and k_{i+1} are distinct and $k_i + 1 \leq n$. Between two consecutive levels j and $j+1$ $(j = 0, \ldots, n-1)$,

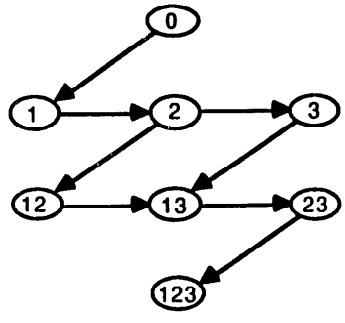

Fig. 1.

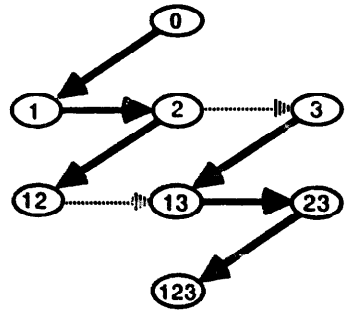

Fig. 2.

an arc extends from each node $k_1k_2\cdots k_j$ on level j, with $k_1 \neq 1$, to the node $1k_1k_2\cdots k_j$ on level $j+1$.

An interesting feature of the Hasse diagram based on $n \geq 2$ components is that it contains two copies of the Hasse diagram on $n-1$ components. One of the copies is comprised of all nodes in which component n is operating: this copy is an exact duplicate of the Hasse diagram for the $n-1$ component system. The second copy contains all nodes in which component n has failed. The labels in this copy simply have the integer n adjoined to the end of each label in the first copy. It is straightforward to verify that there are 2^{n-2} arcs joining these two copies. The Hasse diagram of the 3 component system shown in Fig. 2 illustrates the above duplication feature. The two copies of the Hasse diagram on 2 components are shown with heavy lines, while the $2^{3-2}=2$ arcs joining the copies are shown as dotted lines.

If f_n denotes the number of arcs in the Hasse diagram for n components, then this duplication property shows that f_n satisfies $f_n = 2f_{n-1} + 2^{n-2}$, $n \geq 2$. Solving this recursion with $f_1 = 1$ then yields the following.

Property 2.2. *The total number of arcs in the Hasse diagram for n components is* $(n+1)2^{n-2}$.

This same partially ordered set $(S_n, \geqq)$ arises in other contexts [6,7,9], and known properties of this structure are now briefly mentioned. Several of these will be useful in the subsequent discussions of Section 3. First, the partially ordered set forms a special type of lattice, a distributive lattice [9]. Moreover, the lattice can be *ranked*: namely, its nodes can be decomposed into subsets $P_0, P_1, \ldots, P_h$, such that arcs of the Hasse diagram only join nodes in consecutive sets P_k. The rank of any node is simply the sum of the integers comprising its label. Thus, node 0 has rank 0, node $12\cdots n$ has rank $\frac{1}{2}n(n+1)$, and so the lattice has a height (maximum rank) $h = \frac{1}{2}n(n+1)$.

In an n component system with height h, let $r = (r_0, r_1, \ldots, r_h)$ be its rank vector, with $r_i = |P_i|$ signifying the number of nodes having rank i. As shown in [6,9], the rank vector is symmetric and unimodal. For example, the Hasse diagram for a 3 component system has $h = 6$ and the rank vector $r = (1, 1, 1, 1, 1, 1, 1)$.

3. An algorithm for state generation

Once the state space has been identified as a partially ordered set, it is not difficult to formulate an algorithm for generating, in order, the most probable states of the system. This section describes an algorithm, which conceptually works on the Hasse diagram, and then compares it to existing procedures for solving this problem. We also examine a worst-case upper bound on the complexity of our algorithm, which is related to a certain invariant of the underlying partial order.

In the Hasse diagram for a partially ordered set, let the *indegree* of a node be the number of nodes (or *predecessors*) that cover the given node: i.e., the number of arcs entering that node in the Hasse diagram. Any (finite) partially ordered set contains at least one node with indegree 0. In our particular case, $(S_n, \geqq)$ contains a single such node (namely, node 0), and it corresponds to the most probable state of the system. When this node is removed from the Hasse diagram, there will be at least one node in the reduced Hasse diagram with indegree 0. More generally, when the first k most probable nodes have been removed from the Hasse diagram, there will remain some set C of nodes having indegree 0 in the reduced Hasse diagram. Any two such nodes (states), $X, Y \in C$ must be incomparable in the original partial order. All such nodes are thus candidates for selection as the next most probable state. The partial information embodied in (1) is not sufficient to completely order the probabilities of the states represented in C. However, by comparing R-values of the candidates, the next most probable state (selected from C) can be determined.

The general idea of the algorithm is to at each step remove from C a node X with largest R-value and then update the candidate set C, since the removal of X (including its arcs) may create new nodes with indegree 0. Namely, any successors Y of X (having X as a predecessor) will have their indegree reduced by 1. In order to avoid scanning the entire partial order, it is useful to maintain an "active set" A, which contains those nodes in the Hasse diagram having a predecessor that has already been removed from the Hasse diagram. A node is transferred from the set A to the candidate set C whenever its indegree becomes 0. This process of successively removing nodes from C and updating the relevant sets can be continued until all states in S_n have been generated in order. More typically, however, the process is continued until some stopping criterion is satisfied. For example, termination may be governed by achieving some desired *coverage* π of the state space; the coverage is defined as the sum of the probabilities of the most probable states generated so far [5].

The procedure described above yields the following algorithm GENERATE for identifying the most probable states in order of (nonincreasing) probability.

Algorithm GENERATE

Input: Number of components (n), coverage desired (π), and component probabilities $(p_1, \ldots, p_n)$ ordered as in (1).

Output: States in order of nonincreasing probability until the specified coverage is obtained.

Step 1. [Initialization]
$C := \{0\}$; $A := \emptyset$; sum: = 0;
Step 2. [Iterative loop]
while (sum $< \pi$) **do**
Step 2.1. delete the node $X \in C$ having the largest R-value;
Step 2.2. sum: = sum $+ p(X)$; output X;
Step 2.3. find successors of X, place them on A (if not already present) and update their indegrees;
Step 2.4. remove all nodes with indegree 0 from A and place them on C;

We now establish some properties of the sets used in the algorithm above. Let D denote the set of elements deleted from the partially ordered set at some step of the algorithm. Thus the states in D have already been correctly generated in order. The "neighborhood" of D in $(S_n, \geqq)$ is defined as $\Gamma(D) = \{X \notin D: X \text{ has a predecessor } Y \in D\}$ so $\Gamma(D) = C \cup A$. A set $K \subseteq S_n$ is termed *convex* [3] if for any $X, Z \in K$ and $Y \in S_n$ with $X \geqq Y \geqq Z$ we have $Y \in K$. The following two results are proved in the Appendix.

Lemma 3.1. *The set D is a convex subset of S_n.*

Theorem 3.2. *At any step of the algorithm, $|\Gamma(D)| \leq |D|$.*

In our implementation of algorithm GENERATE, the candidate set C is maintained as a heap [8]; this allows selection and deletion of the most probable node from C (Step 2.1) to be done in time logarithmic in the size $|C|$ of the current candidate set. Suppose a total of k nodes (states) have been output when the algorithm terminates. The total number of insertions into C (Step 2.4) minus the total number of deletions from C equals the final size of C, which by Theorem 3.2 is at most k. Since there are exactly k deletions from C, at most $2k$ insertions are required into the heap, and thus Step 2.4 takes at most $O(k \log |C|)$ operations. In addition, we use a hash table to maintain the active set A. While the insertion and/or location of an element in a hash table can be expensive in the worst case, on average these operations can be carried out in constant time [1]. Since there will be at most $O(n)$ successors of a given node, Step 2.3 requires $O(nk)$ work on average. The average computational complexity of algorithm GENERATE can then be expressed as $O(nk + k \log m)$, where m is the maximum size of the heap occurring during execution of the algorithm.

By Theorem 3.2, we are assured that $|C|$ is in general less than k, and so a (weak) upper bound on m is k. As a matter of fact, the algorithm ORDER of Lam and Li uses a heap whose typical size is k, yielding the $k \log k$ term in their complexity

estimate $O(nk + k \log k)$. Thus, the proposed algorithm might provide some advantages over that given in [4] with respect to both computational effort and storage, since our candidate set is (even in the worst case) smaller than the corresponding candidate set used in the algorithm of Lam and Li. In order to get a sense of the difference between the sizes of these sets in practice, we show the results of running both algorithms on a small example, having $p_1 = 0.55$, $p_2 = 0.6$, $p_3 = 0.7$, $p_4 = 0.8$, $p_5 = 0.9$. Table 1 shows for this example (run until a coverage of $\pi = 0.90$ is achieved) the sizes of the respective candidate sets along with the most probable state output at each iteration. The candidate set for GENERATE here stays much smaller than the corresponding set used by ORDER. Moreover, the active set required for GENERATE in this example attained a maximum size of 4, still quite small. This finding, that the candidate set in GENERATE stays quite modest in size, is confirmed by the empirical results presented in Section 4.

The computational complexity of algorithm GENERATE, both theoretically and empirically, is determined in large part by the effort needed to process the candidate set C. While Theorem 3.2 provides the upper bound $k = |D|$ on the maximum size of the set C, we now explore an alternative upper bound related to the structure of the partially ordered set $(S_n, \leqq)$.

An important observation, made earlier, is that all elements in C must be incomparable in the partial order, since they all have indegree 0 in the reduced Hasse diagram. In other words, the elements of C form an *antichain* in the partially ordered set: a set of mutually incomparable elements in the partial order. Thus, an upper bound on the maximum size m of C is the size μ of the largest antichain in $(S_n, \leqq)$.

Table 1

Iteration	Size of candidate sets		Most probable state
	ORDER	GENERATE	
–	–	–	0
0	1	1	1
1	2	1	2
2	3	2	12
3	4	1	3
4	5	2	13
5	6	2	23
6	7	2	4
7	8	3	123
8	9	2	14
9	10	2	24
10	11	3	124
11	12	2	5
12	11	2	34
13	12	2	15
14	11	2	134

Table 2. % of states needed for $\pi = 0.90$ coverage

n	$p = 0.85$	$p = 0.9$	$p = 0.95$
10	12.7	4.88	1.07
15	4.54	1.28	0.22
20	1.60	0.30	0.018
25	0.56	0.045	0.003
30	0.19	0.013	0.0003

Stanley [9] and Proctor [6,7] have studied in other contexts this same partially ordered set, and have demonstrated some rather deep results concerning its structure. Specifically, the elements of the partially ordered set can be partitioned into sets P_i of rank i. Moreover, the partially ordered set is *Sperner*, meaning that the size of the largest antichain is the same as the maximum size of the sets P_i. Since the partially ordered set is rank unimodal and rank symmetric, a maximum-sized antichain occurs for a set P_i at the half-height δ of the lattice: namely, at rank $\delta = \lfloor \frac{1}{2}h \rfloor = \lfloor \frac{1}{4}n(n+1) \rfloor$.

As a result, we obtain an upper bound $\mu = |P_\delta|$ on the maximum possible size of the candidate set. Since the rank of a node $i_1 i_2 \cdots i_t$ is simply the sum of its constituent integers i_j, another way of expressing this upper bound μ is as the number of partitions of the integer δ into distinct parts, none of which exceed n. Equivalently μ is the coefficient of x^δ in the generating function $(1+x)(1+x^2)(1+x^3)\cdots(1+x^n)$.

4. Computational results

In this section, we assess the empirical computational complexity of state generation algorithms. First it should be pointed out that if all $p_i = \frac{1}{2}$, then every state has equal probability and in order to generate coverage proportion π that same proportion of the system states needs to be generated. In this case there is little point in employing state generation techniques. However, if the system components are reasonably reliable, then a relatively small proportion of the system states accounts for a large proportion of the total probability. To illustrate this fact suppose for simplicity that all components have the common probability $p_i = p$. Table 2 displays the percent of states that need to be generated (in most probable order) to achieve $\pi = 0.90$. For example, in a 25 component system with $p = 0.9$, only 0.045% of the system states need to be generated to achieve the specified coverage. As seen in the table, this proportion dramatically decreases as n and p increase. Thus there is hope for being able to approximate the performance of binary systems if the individual components are sufficiently reliable. Fortunately this is frequently the case in practice and the empirical results reported here will study such systems.

In our computational study, the number of components $n = 5, 6, \ldots, 20$ and the component probabilities (not necessarily identical) are randomly selected from the interval $[0.85, 0.995]$. The set of p_i's defining the $n-1$ component system is a

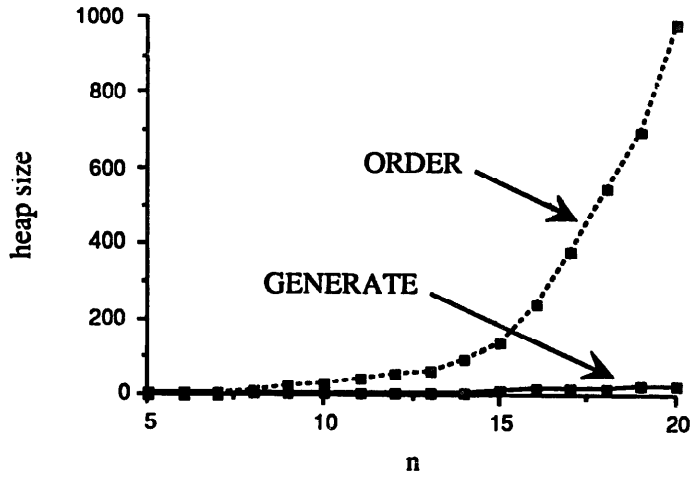

Fig. 3. Comparison of maximum heap sizes.

subset of the set of p_i's for the n component system. All of the experimental data were obtained using implementations of the algorithms in Pascal, run on a Macintosh Plus microcomputer.

Figure 3 compares the maximum size of the candidate sets required by the two generation algorithms GENERATE and ORDER for $\pi = 0.90$ coverage. It is seen that the candidate set for GENERATE remains quite manageable in size, whereas that for ORDER grows rapidly with n. As a result, the heap operations are more expensive with the latter algorithm. Figure 4 compares the total storage requirements of both algorithms; in the case of GENERATE, this total includes the (maximum) storage for both C and A. In terms of overall space efficiency, then, GENERATE is more economical than ORDER. This can be an important practical consideration, because of potential limitations on available storage imposed by the

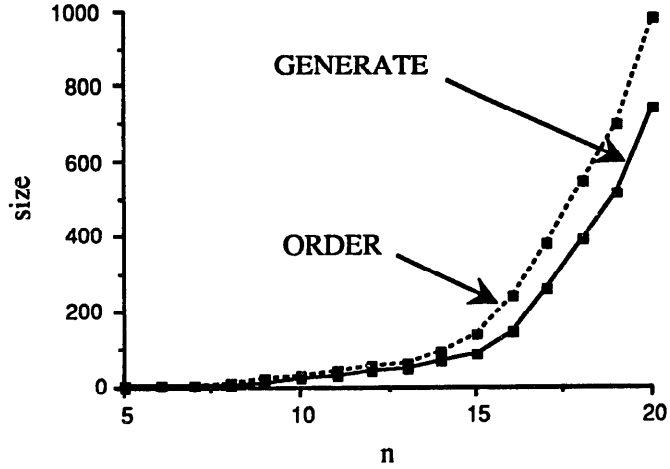

Fig. 4. Comparison of total storage.

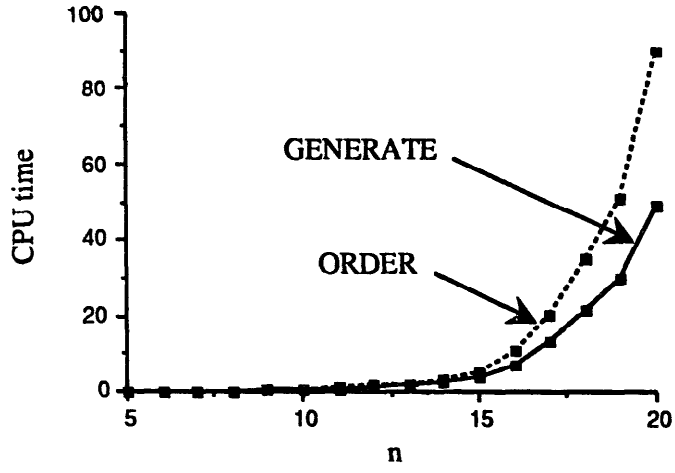

Fig. 5. Comparison of execution times.

growth of the state space with n. The empirical time complexity of the algorithms is shown in Fig. 5, which displays the CPU time (in seconds) required for the test problems of different sizes n. The superiority of GENERATE over ORDER is clearly evident in these test problems.

It is important to stress that all algorithms for generating most probable states face an explosive growth of the state space with n. As a result, the applicability of such algorithms is probably limited to about 30–35 components, still an advance over existing approaches for calculating exact performance measures in general stochastic networks. In fact, there are no general methods known for approximating such performance measures that operate in polynomial time [2]. Therefore, an empirical assessment of algorithms, as that carried out here, is more appropriate than comparison based on asymptotic worst-case behavior.

5. Conclusions

The objective of this paper has been to describe how the identification of an underlying discrete structure aids considerably in understanding a certain problem arising in reliability analysis. Specifically, being able to place the components in order of nondecreasing reliability provides a good deal of information about the relative probability of states in the state space. In addition, this algebraic viewpoint leads quite naturally to an algorithm based on the Hasse diagram for generating the states in order. It is of interest that the analysis of this derived algorithm itself is aided by studying the maximum-sized antichain in the lattice. Computational results are presented to complement the theoretical findings, and they indicate that the algorithm described here is reasonably effective in practice, offering storage and speed improvements over existing approaches.

Acknowledgement

The authors are indebted to Jim Lawrence for his helpful and insightful comments. The work of the first author was supported by the United States Air Force Office of Scientific Research (AFSC) under Grants AFOSR-84-0154 and AFOSR-89-0071.

Appendix

We present here proofs of the results stated in Section 3.

Proof of Lemma 3.1. Suppose $X, Z \in D$ with $X \geqq Y \geqq Z$. Since Y is more probable than Z it must be removed before Z. Because $Z \in D$, it follows that $Y \in D$. □

Proof of Theorem 3.2. We establish a one-to-one mapping from $\Gamma(D)$ to D. Suppose that node X is in $\Gamma(D)$. Then $X = i_1 i_2 i_3 \cdots i_m \cdots i_t$ has some predecessor $Y \in D$, say $Y = i_1 i_2 i_3 \cdots i_m - 1 \cdots i_t$. If there are several predecessors, the choice can be made arbitrarily. Also if $m = 1$ and $i_m = 1$, then we interpret this prescription as defining $Y = i_{m+1} \cdots i_t$. Now define the map $\psi : \Gamma(D) \to D$ using $\psi(X) = i_1 i_2 i_3 \cdots i_{m-1} i_{m+1} \cdots i_t$. Note that $0 \geqq \psi(X) \geqq Y$, where node 0 is the most probable state; since $0, Y \in D$ then the convexity of D yields $\psi(X) \in D$.

Now suppose that there is some $X' \in \Gamma(D)$, $X' \neq X$, with $\psi(X') = \psi(X)$. Then X' must have a predecessor $Y' \in D$, which we assume has the form

$$Y' = i_1 i_2 i_3 \cdots i_{m-1} i_{m+1} \cdots i_{r-1} k i_r \cdots i_t.$$

(A similar argument governs other placements of the index k.) Since the indices satisfy $i_m < i_{m+1} < \cdots < i_{r-1} < k < i_r < \cdots < i_t$, it can be verified that $Y \geqq X \geqq Y'$ holds. Again by the convexity of D and the fact that $Y, Y' \in D$ it follows that $X \in D$, a contradiction. Thus, the map is one-to-one and the stated result follows. □

References

[1] A. Aho, J. Hopcroft and J. Ullman, Data Structures and Algorithms (Addison-Wesley, Reading, MA, 1983).

[2] M. Ball, Computational complexity of network reliability analysis: an overview, IEEE Trans. Reliability 35 (1986) 230-239.

[3] G. Grätzer, Lattice Theory (Freeman, San Francisco, CA, 1971).

[4] Y. Lam and V. Li, An improved algorithm for performance analysis of networks with unreliable components, IEEE Trans. Comm. 34 (1986) 496-497.

[5] V. Li and J. Silvester, Performance analysis of networks with unreliable components, IEEE Trans. Comm. 32 (1984) 1105-1110.

[6] R. Proctor, Representations of sl(2,C) on posets and the Sperner property, SIAM J. Algebraic Discrete Methods 3 (1982) 275–280.

[7] R. Proctor, Solution of two difficult combinatorial problems with linear algebra, Amer. Math. Monthly 89 (1982) 721–734.

[8] R. Sedgewick, Algorithms (Addison-Wesley, Reading MA, 1983).

[9] R. Stanley, Weyl groups, the hard Lefschetz theorem, and the Sperner property, SIAM J. Algebraic Discrete Methods 1 (1980) 168–184.