Markov Algorithms for Computing the Reliability of Staged Networks

R. C. H. Cheng

Technical Report No. UNC/ORSA/TR-86/8

April 1986

UNIVERSITY OF NORTH CAROLINA
AT CHAPEL HILL

Approved for public release; distribution unlimited.
Markov Algorithms for Computing the Reliability of Staged Networks

R. C. H. Cheng

Technical Report No. UNC/ORSA/TR-86/8

April 1986

Curriculum in Operations Research and Systems Analysis
University of North Carolina at Chapel Hill

This research was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0140. Reproduction in whole or part is permitted for any purpose of the United States government.

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
OFFICE OF ADVANCED RESEARCH AND DEVELOPMENT
This technical report has been reviewed and is approved for public release IAW AFR 190-12. Distribution is unlimited.

MATTHEW J. KERGER
Chief, Technical Information Division
ABSTRACT

Certain commonly occurring types of network, whether directed or undirected, exhibit a staged structure. Two algorithms, based on node partitioning, are presented which take advantage of such structure and which use a Markov transition-probability form of recursion. The algorithm for directed networks is related to the Markov chain formulation of Bailey and Kulkarni, but for undirected networks a more detailed form of state definition is used related to one suggested by Rosenthal.

The computational advantages of the algorithms are discussed and some numerical results presented.

Key Words: Network Reliability, Source-to-sink connectedness, Node Partition, Staged Network, Recursive Algorithm.
1. INTRODUCTION

Consider a network $G$ with node set $N$ and arc set $A$. The nodes are perfect but arcs fail randomly and independently with probability $1-p_a$, $a \in A$. Let $s$ and $t$ be two specified nodes. The calculation of

$$P_{st} = \text{prob (there is a path from } s \text{ to } t)$$

is called the $s$-$t$ connectedness problem.

This problem is NP hard [10]. However, many algorithms exist which solve it by enumerating key structures in the network. If the number of these is small then such an algorithm may be computationally efficient. Examples include spanning trees [2], acyclic subgraphs [8], K-graphs [7] and cutsets [4].

A number of methods use node partition [1,6,9]. Though none of these make any special assumption about the form of the network, node partition seems to be a particularly good approach for networks which have a staged structure. In this paper we consider two node partition algorithms which exploit such structure.

Our starting point is a decomposition used by Shogan [9] for directed networks where the nodes are divided into disjoint groups - which we call stages - and 'events' are then defined at each stage in a way which allows their probabilities of occurrence to be calculated from those of the previous stage. Shogan describes an algorithm based on path enumeration and mentions, but does not follow the possibility of one based on state enumeration. We suggest a modification of the definition of event given by Shogan which gives an algorithm of this latter type having a simpler form than that based on path enumeration.
Undirected networks can often also be decomposed into stages. Our main result will be to show how the algorithm can be adapted to deal with this case, too.

Both versions of the algorithm are related to ones already cited. The directed network version is related to that proposed by Bailey and Kulkarni [1], whilst the undirected network algorithm is a recursive version of the general, but not fully specified, framework described by Rosenthal [6].

As well as describing the algorithms, we assess their computational advantages — and drawbacks — including some numerical results.

For clarity, the bulk of our discussion assumes a form of stage decomposition that is simpler than necessary. In section 5 we show how the stage decomposition can be generalized to allow versions of the algorithms that are, in certain situations, much more efficient than the prototype versions.

2. DEFINITIONS AND NOTATION

2.1 Staged Networks

Let \( G \) be a network with node set \( N \) and arc set \( A \). An arc connecting node \( i \) to node \( j \) is denoted by \((i,j)\). We shall say that \( G \) is a **staged network** if \( N \) can be divided into a number, \( Q \) say, of disjoint sets

\[
S_1 = \{s\}, S_2, ..., S_Q = \{t\}
\]  

(2.1)

such that
\( \tilde{S} \): if \((i,j) \in A\) and \(j \in S_q \) then \(i \in S_{q-1} \cup S_q \). \hfill (2.2) 

(Shogan gives a slightly more general version which allows \(i \in S_1 \cup S_{q-1} \cup S_q \) in (2.2). The generalized definition of Section 5 incorporates this case.)

Condition \( \tilde{S} \) ensures that any path from \( s \) to \( t \) can only go from nodes of one stage to those of the next stage. Thus no 'backtracking' from a higher to lower stage is possible.

The node-stages induce a corresponding stage structure on the arcs \( A \), dividing these into disjoint sets:

\[ A_2, A_3, \ldots, A_Q \]

where

\[ A_q = \{ (i,j) \mid i \in S_{q-1} \cup S_q \text{ and } j \in S_q \} \]. \hfill (2.3)

To note which arcs of \( A_q \) are up (operating) and which are down (failed), we define

\[ X_i = 1 \text{ or } 0 \text{ according as arc } i \in A_q \text{ is up or down, and } \]

\[ X = (X_1, X_2, \ldots, X_{|A_q|}). \] \hfill (2.4)

a failure pattern of \( A_q \). It will be clear, from the context, which subset \( A_q \) is being referred to, so the dependence of \( X \) on \( q \) will be suppressed.

Those arcs which, under \( X \), are up, will be called the set of up-arcs of \( A_q \). The probability of obtaining \( X \) is

\[ p(X) = \prod_{a : X_a = 1} p_a \cdot \prod_{a : X_a = 0} (1 - p_a). \] \hfill (2.5)
There are $2^{\left| A_q \right|}$ distinct failure patterns of $A_q$ and we denote the set of all these as $F_q$.

Consider now an undirected network $G$. We shall say that $G$ is \textit{staged} if the nodes can be divided into disjoint sets

$$S_1 = \{s\}, S_2, \ldots, S_Q = \{t\}$$

such that

\begin{equation}
\tilde{T}: \text{ if } (i,j) \in A \text{ and } i \in S_q, \ 1 < q < Q \text{ then } j \in S_q \cup S_q \cup S_{q+1}. \quad (2.6)
\end{equation}

Condition $\tilde{T}$ requires that an arc can only join two nodes of the same or adjacent stages. Decomposition of the arc set into disjoint subsets $A_q$ exactly as defined in (2.3) is possible, and the definition of failure pattern $X$ given in (2.4) also still applies. An example of an undirected staged network is given in Fig. 2.

\subsection*{2.2 Recursive Markov Algorithms}

The algorithms that we investigate operate in the following way:

(i) A set of events, $E$, has to be found where each event $i \in E$, is associated with some subset of nodes.

(ii) The events are \textit{Markov} in the sense that their probabilities of occurrence, $p_i$, can be calculated from the standard Markov one-step transition formula

\begin{equation}
p_j = \sum_{i \in E} p_i q_{ij} \quad J \in E \quad (2.8)
\end{equation}

where

\begin{equation}
q_{ij} = \text{prob} (J \text{ occurs} | I \text{ occurs}) \quad (2.9)
\end{equation}
is the transition probability.

(iii) If the events can be numbered so that transitions go only from a lower to higher numbered event, then \( q_{IJ} = 0 \) if \( J < I \) and the \( p_J \) may be calculated recursively from (2.8) in increasing order of \( J \). A special form of this is when the events can be grouped into disjoint sets \( E_1, E_2, \ldots, E_Q \) with events of one group giving rise to events of the next higher group only. Then (2.8) reduces to

\[
p_J = \sum_{I \in E_q} p_I q_{IJ}, \quad J \in E_{q+1}, \quad q = 1, 2, \ldots, Q. \quad (2.10)
\]

3. THE ALGORITHMS

3.1 A Markov Algorithm for a Directed Staged Network

Consider a directed staged network. The algorithm of Shogan focuses on events associated with a particular stage. Let \( V \) be a subset of \( S_q \). Shogan defines the events

\[
U(V) = \text{"there is a path from } s \text{ to at least one node of } V"
\]

\[
I(V) = \text{"there are paths from } s \text{ to all nodes of } V"
\]

and gives exclusion-inclusion formulas relating the probabilities of occurrence of such events to corresponding events of the previous stage.

It is perhaps simpler to use the following events. Let \( V \subseteq S_{q-1} \) and \( W \subseteq S_q \) and define
[V] = "there is a path from s to all nodes of V, but none from s to \( S_{q-1}\) \( V\)" (3.2)

[V,W] = "Given [V] has occurred, each node of W is reachable from some node of V but there is no path from \( S_{q-1}\) \( V\) to W, or from V to \( S_{q}\) \( W\)" (3.3)

The event (3.2) can be viewed as the union of a number of events, of a more general type given by Bailey and Kulkarni [1], that utilizes the staged structure.

Let \( p[V] \) and \( p[V,W] \) denote the probabilities that [V] and [V,W] occur. Then

\[
p[s] = 1
\]

\[
p[W] = \sum_{V \subseteq S_{q-1}} p[V]p[V,W], \quad W \subseteq S_{q}, \quad q=2, \ldots, Q.
\]

This has form (2.10): the events \( \{[V], V \subseteq S_{q-1}\} \) form \( E_{q-1} \), and the \( p[V,W] \) are the \( q_{ij} \). The last stage calculation (when \( q = Q \) and \( W = \{t\} \) yields \( p_{st} \), the probability of s-t connectedness.

What makes (3.4) simple is that the \( p[V,W] \) are very easily calculated, because \( [V,W] \) depends only on the states of the arcs of \( A_q \). To see this, suppose that \( v \in V \), \( w \in S_{q} \) and there is a path of up-arcs from v to w. From the definition of staged network, this path may initially pass through nodes of \( S_{q-1} \), however, once it reaches a node of \( S_{q} \) it can only continue through nodes of \( S_{q} \) until w is reached. Thus, given [V] has occurred, all the nodes of \( S_{q-1} \) on the path must belong to V. If \( v' \) is the last node of V
on the path, there is, therefore, a path from \( v' \in V \) to \( w \) comprised entirely of up-arcs of \( A_q \). Similarly a path from \( u \in S_{q-1} \setminus V \) to \( W \) would, given that \([V]\) has occurred, imply the existence of some \( u' \in S_{q-1} \setminus V \) and a path of up-arcs of \( A_q \) from \( u' \) to \( w \). It follows, therefore, that \([V,W]\) occurs if and only if, using paths with arcs in \( A_q \) only, each node of \( W \) is reachable from some node of \( V \), no node of \( S_q \setminus W \) is reachable from \( V \), no node of \( W \) is reachable from \( S_{q-1} \setminus V \). This is the required result. It can be viewed in the following way:

**Lemma 1** Let \( V \in S_{q-1} \) and \( X \in F_q \). Then the pair \( V,X \) induces a subset \( W \subseteq S_q \) defined by

\[
W(V,X) = \{ w \mid w \in S_q \text{ and there is a path, comprised only of up-arcs of } A_q \text{ (under } X \text{), from some node of } V \text{ to } w \} \tag{3.5}
\]

Figure 1 illustrates an example of such a \( W(V,X) \).

Clearly

\[
p[V,W] = \sum_{X \in F_q} p(X) \quad \text{if } W(V,X) = W \tag{3.6}
\]

and (3.4) reduces to \( p[s] = 1 \) and

\[
p[W] = \sum_{V \in S_{q-1}} \sum_{X \in F_q} p(V)p(X), \quad q=2, \ldots, Q. \tag{3.7}
\]

In numerical calculations (3.7) is easy to implement as there is no need to compute the \( p[W] \) one at a time. Instead, note that in
calculating all the \( p[W] \), each \( V,X \) combination is used once only. Thus (3.7) can be implemented as

**Algorithm A**

\[
\text{Algorithm A} \\
p[s] \leftarrow 1 \\
\text{for } q=2,\ldots,Q \\
p[W] \leftarrow 0 \text{ all } W \in S_q \\
\text{for each } V \in S_{q-1} \\
\text{for each } X \in F_q \\
\quad W \leftarrow W(V,X) \\
p[W] \leftarrow p[W] + p[V]p(X) \\
\text{next } X \\
\text{next } V \\
\text{next } q
\]

(3.8)

3.2 A Markov Algorithm for Undirected Staged Networks

For an undirected network the event definition (3.2) does not give a useful algorithm because a path may 'backtrack.' Instead, consider all the nodes of a stage \( S_q \) and think of them as partitioned into disjoint groups. Fig. 2 illustrates \( S_2 \) partitioned as \((1)(2)(34)\). We denote a partition by \( \pi \) and the set of all partitions of \( S_q \) by \( T_q \), and define the following event:

\[
[I] = [\pi; v] \\
= "\pi \in T_q \text{ and, using arcs of } \bigcup_{r=2}^q A_r \text{ only:} \\
(i) \text{ each group of nodes in } \pi \text{ is connected} \\
(ii) \text{ nodes between groups are unconnected} \\
(iii) s \text{ is connected to group } v"
\]

(3.9)
The set of all such events associated with $S_q$ will be denoted by Eq. Fig. 2 illustrates the event $[(1)(2)(34); 3] \in E_2$ and the event $[(5)(67); 2] \in E_3$.

The above definition can be viewed as a specialization of a general type of event considered by Rosenthal [6], who outlines an algorithm framework based on combining subnetworks two at a time. The framework does not make specific use of staged structure but clearly has such a form in mind. The algorithm below takes specific note of the Markov nature of the calculation, so that the computational form is somewhat different to that of Rosenthal's even though the event probabilities are essentially the same. We use an analogue of Lemma 1:

**Lemma 2** An event $I = [\pi; x] \in E_q - 1$ and failure pattern $X \in F_q$ induces a unique event $J = [\rho; y] \in E_q$, which will be denoted by $J(I, X)$.

**Proof** We form a subnetwork, $H$, as follows. The node set is $S_{q-1} \cup S_q$. The nodes of $S_q$ are treated normally. However, those of $S_{q-1}$ are divided by the partition $\pi$ into disjoint groups and, in $H$, the nodes of each group are treated as being combined into a single node (there being a separate node for each set). The arc set of $H$ is just the set of up-arcs of $X$.

Now in $H$ the nodes of $S_q$ can be naturally partitioned into disjoint sets; all the nodes within a set are connected (possibly via paths which pass through combined nodes of $S_{q-1}$), but nodes belonging to different sets are disconnected. If we denote this
partition by \( p \) and take \( v \) to be the set (of \( p \)) which is connected
to \( \lambda \), then this uniquely defines \( J = [p; v] \).

Fig. 2 gives an example of such an event \( J(I, X) \).

The same recursive algorithm (3.7) still applies.

\[
p[s; 1] = 1
\]

\[
p[J] = \sum_{I \in E_{q-1}} \sum_{X \in F_q} p[I]p(X), \; J \in E_q, \; q = 2, \ldots, Q
\]

\[
J(I, X) = J
\]  
(3.10)

and again the modified form (3.8) can be used for numerical
calculation:

Algorithm B

\[
p[s; 1] = 1
\]

for \( q = 2, \ldots, Q \)

\[
p[J] + 0 \; \text{all } J \in E_q
\]

for each \( I \in E_{q-1} \)

for each \( X \in F_q \)

\[
J \leftarrow J(I, X)
\]

\[
\]

next \( X \)

next \( I \)

next \( q \)

4. COMPUTATIONAL ASPECTS AND COMPLEXITY

For Algorithm A, the calculation of \( W(V, X) \) and \( p[W] \) is done

\[
2|S_{q-1}| + |A_q|
\]

(4.1)
times at stage $q$. For Algorithm B, finding the partitions $\rho$ is the most time consuming calculation. From the definition of $J(I,X) = [\rho; \nu]$ where $I = \{\pi; \lambda\}$ we see that $\rho$ depends on $\pi$ and $X$ only and not on $\nu$. Thus in (3.11) $\rho$ needs only be determined for each $(\pi,X)$ and not each $(\pi,X,\nu)$, a total of

$$g(|S_{q-1}|) \times 2^{|A_q|}$$

(4.2)
times, where $g(m)$ = number of possible partitions of $m$ objects, is the sum of Stirling numbers of the second kind (see, eg. [5]). As $g$ grows hyperexponentially, see Table 1 for selected values, this is the main factor limiting the size of networks that can be handled.

Table 1

<table>
<thead>
<tr>
<th>$m$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>2m</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>$g(m)$</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>15</td>
<td>52</td>
<td>203</td>
<td>877</td>
<td>4140</td>
</tr>
</tbody>
</table>

Each calculation of $W$ in Algorithm A, and $\rho$ in Algorithm B, requires examination of the connectivity of the subgraph $G_q = \{S_{q-1} \cup S_q, A_q\}$. An algorithm like "breadth-first search" (see [5], for example) can be used for this, and will need $O(|S_{q-1}| + |S_q| + |A_q|)$ elementary operations.

Algorithm B has the additional requirement of a hash function or subroutine which (i) assigns a label to each
partition (of a set of given cardinality) and, inversely, (ii) given a particular label, specifies the precise grouping of the nodes in that partition. Part (i) is needed to identify the component of the array \( P[J] \) to be incremented in (3.11) and (ii) is needed to allow partitions to be stepped through systematically in the outer loop of (3.11). The Appendix gives such a subroutine, whose complexity is \( O(m) \) where \( m \) is the cardinality of the set being partitioned.

To test its effectiveness, a Fortran version of Algorithm B was used, on an IBM Personal Computer AT, to calculated \( P_{st} \) for the dodecahedron network of Fig. 3 and the grid network of Fig. 4. The computation time was 4½ minutes and 52 minutes respectively. Though we have not made a direct comparison, this compares well with other algorithms. For example, for the test network used in [1], which is in effect a dodecahedron reduced by 3 nodes and 5 arcs, Bailey and Kulkarni report timings of 54 minutes, 8 minutes and 1 minute 26 seconds using respectively, the algorithms of Buzacott [3], Provan and Ball [4] and their own [1], on an IBM 4381-k which is approximately 10 times faster than the IBM PC-AT, (all algorithms computed the reliability as .99806 when \( P_a = 0.9 \), all a).

Both Algorithms A and B resemble dynamic programming recursions whose calculations of one stage depend only on the results of the immediately preceding stage. For networks like the grid of Fig. 4, this means that a substantial computational
saving is possible by avoiding the repetition of identical calculations made at each stage. More precisely, suppose the subnetworks \( G_q = \{S_{q-1} u S_q, A_q \} \) and \( G_r = \{S_{r-1} u S_r, A_r \} \) (with \( q < r \), say) have exactly the same structure. Then, during the calculations at stage \( q \), the computed values of \( W(V, X) \) (respectively \( J(I, X) \)) should be saved for all \( V \in S_{q-1} \) (respectively \( I \in E_{q-1} \)) and \( X \in F_q \). This time consuming computation need not then be repeated during the \( r \)th stage calculation, as, with suitable labelling of the nodes and arcs of \( G_r \), so that they correspond to those of \( G_q \), the same values of \( W(V, X) \) (respectively \( J(I, X) \)) can be used for \( V \in S_{r-1} \) (respectively \( I \in E_{r-1} \)) and \( X \in F_r \).

For example in the grid network of Fig. 4, the CPU time of 52 minutes is mostly taken up by the calculations of stage 3 (12 min 5 secs) and of stages 4, 5 and 6 (13 min 10 secs each). As \( G_q \) (\( q = 3, 4, 5, 6 \)) are identical, we would estimate an algorithm that avoids the repeated calculation of the \( J(I, X) \) would reduce the computation by more than half.

Finally we note that, for clarity of exposition, an unnecessarily rigid definition of staging has been given. In the next section we show how the definition can be generalized. It should be emphasized that though the timings above already compare favorably with published work, the use of generalized stages can lead to further substantial savings in computing time.
5. OVERLAPPING STAGES

The definitions of staging, $\tilde{S}$ and $\tilde{T}$, can be relaxed to give more flexibility to the two algorithms. The following generalization can be used for both directed and undirected networks.

$\tilde{U}$: There are three sequences of subsets of $N$:

$$S_1 = \{s\}, S_2, \ldots, S_q = \{t\}$$

$$M_1, M_2, \ldots, M_{Q-1}$$

$$N_1 = \{s\}, N_2, \ldots, N_q = \{t\}$$

such that

$$\bigcup_{q=1}^{Q} S_q = N$$

$$M_{q-1} \subseteq S_{q-1}$$

$$N_q \cap \left( \bigcup_{r=1}^{q-1} S_r \right) = \emptyset.$$

And for any $(i,j) \in A$:

if $i \in N_q$, $j \in N_r$ and $q > r$ then $j \in S_{q-1}$  \hspace{1cm} (5.4)

The condition (5.3) allows the sets $S_q$ to overlap. Conditions $\tilde{S}$ and $\tilde{T}$ comprise the special case $M_q = S_q$, all $q$, when $N_q = S_q$; the $S_q$ are then disjoint and (5.4) reduces to (2.2) or (2.6).

It is readily verified that the $N_q$ are disjoint and that

$$\bigcup_{q=1}^{Q} N_q = N.$$  \hspace{1cm} (5.5)
The node stages induce a staged structure on the arcs which still decompose into disjoint subsets

\[ A_2, A_3, \ldots, A_Q \]  

where

\[ \bigcup_{2}^{Q} A_q = A \quad (5.6) \]

\[ A_1 = \{(i,j) | i \in S_{q-1} \cup N_q \text{ and } j \in N_q \}. \quad (5.7) \]

With this definition of \( S_q \) and \( A_q \), Algorithms A and B still apply completely unaltered.

The stage \( q \) calculations are still of the order of (4.1) or (4.2) under condition \( \tilde{Q} \). The calculations of stage \( q \) will thus be efficient if \( S_{q-1} \) and \( A_q \) are such that the following two conditions are met.

(i) The cardinality of \( S_{q-1} \) is kept small to keep small the factor involving \( |S_{q-1}| \) in (4.1) or (4.2). This requires making \( |M_{q-1}| \) as large, and \( |N_q| \) as small as possible.

(ii) The cardinality of \( A_q \) is kept small to keep small the factor \( |A_q| \) in (4.1) or (4.2). This can be done, indirectly, by making \( |N_q| \) as small as possible.

There is a trade-off here in that a small \( |N_q| \) at each stage will make for a larger total number of stages \( Q \). As computational complexity depends only linearly on \( Q \), but exponentially on \( |A_q| \), the trade in the direction of small \( |N_q| \) will almost always be worth making.

As an example consider the grid of Fig. 4. If, instead of the staging given in the figure, we use
\[ S_2 = (1,2,\ldots,6), \]
\[ S_3 = (S_2 \setminus (1))u7 = (2,3,\ldots,7) \]
\[ S_4 = (S_3 \setminus (2))u8 = (3,4,\ldots,8) \]
\[ S_5 = (S_4 \setminus (3))u9 = (4,5,\ldots,9) \]
\[ S_{30} = (25,26,\ldots,30) \]
\[ S_{31} = (t) \]

then \(|S_q| = 6, |M_{q-1}| = |N_q| = 1\) for \(q = 2,3,\ldots,30\), and \(|A_q| \leq 2\) for \(q = 3,4,\ldots,30\). The four main stages of the original decomposition, each with \(|A_q| = 11\), is thus replaced by 24 stages each with \(|A_q| \leq 2\). The computational complexity is thus reduced by a factor \(- (4 \times 2^{11})/(24 \times 2^2) = 85.3\). Because of incidental computational overheads which we have not fully accounted for (such as initialization of arrays), the actual improvement in speed will be by a factor rather less 85.3; however the improvement is still very substantial. A Fortran implementation of this version of staging reduced the computing time, from the 52 minutes quoted previously, to 1 minute 58 seconds.

A similar use of overlapping stages for the dodecahedron example of Fig. 3 reduced computation time from 4½ minutes to 34 seconds.
REFERENCES


APPENDIX

We describe here a method of labelling partitions and its inverse which enables a partition to be recovered from its label.

The partitions of \( M \) objects can be divided into \( M \) sets, the \( k \)th set comprising all those partitions where the objects are divided into \( k \) groups. For instance \((13)(2)(4)\) represents the partition of 4 objects into 3 groups: (13), (2) and (4); this partition thus belongs to the 3rd set. Let

\[
g(k,M) = \text{number of partitions (of } M \text{ objects)}
\]

in the \( k \)th set

These are the Stirling numbers of the second kind (see, for example, [5]).

We shall label partitions in the order of these sets (i.e. those of the first set first, then those of the second set, and so on). The position of a partition is thus completely defined once the position within the set to which it belongs is fixed. Table 1A illustrates one method by which the partitions of \( m \) objects may be built up by systematically listing all the ways that the \( m \)th object can be adjoined to each of the partitions of the previous \( m-1 \) objects. The method used is as follows.

For a partition of \( m \) objects, let \( k_m \) be the set to which it belongs and \( j_m \) its position in the set. Now, a partition is defined by specifying, for each object \( m \), the group \( n_m \) to which it belongs. In Table 1A, \( k_M \) and \( j_M \) are determined from \( n_m \) \((m-1,2,\ldots,M)\) using:
\[ k_1 = 1, \quad j_1 = 1, \]
\[ k_m = k_{m-1}, \quad j_m = (j_{m-1} - 1) k_{m-1} + n_m \quad \text{if} \quad n_m \leq k_{m-1} \]
\[ k_m = n_m, \quad j_m = j_{m-1} + g(k_m, m) - g(k_{m-1}, m-1) \quad \text{otherwise} \]

which recursively gives the position \( j_m, k_m \) of the partition of the first \( m \) objects, for \( m = 1, 2, \ldots, M \).

Table 1A

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(1)</td>
<td>(12)</td>
<td>(123)</td>
<td>(1234)</td>
</tr>
<tr>
<td>2</td>
<td>(1)(2)</td>
<td>(13)(2)</td>
<td>(14)(23)</td>
<td>(1)(234)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1)(23)</td>
<td>(12)(3)</td>
<td>(12)(34)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(12)(3)</td>
<td>(12)(4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1)(24)(3)</td>
<td>(1)(2)(34)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1)(2)(34)</td>
<td>(13)(2)(4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1)(23)(4)</td>
<td>(1)(2)(4)</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>(1)(2)(3)(4)</td>
</tr>
</tbody>
</table>

Partition of \( m \) objects into \( k \) groups
The overall position of a partition is

\[ r = \sum_{k=1}^{k-1} g(k,m) + j_m. \]

The inverse calculation allows the group, \( n_m \), to which each object \( m \) belongs, to be calculated from \( r \). It is as follows.

Set \( m = M \) and determine \( k_m, j_m \) \((= k_M, j_M)\) from the conditions

\[ r - M \sum_{k=1}^{k-1} g(k,m) = j_M > 0, \quad r - \sum_{k=1}^{k-1} g(k,M) \leq 0. \]

Moreover \( g(k,m) = kg(k,m) + g(k-1,m-1) \), so that

A. Either \( j_m - k_m g(k_m,m-1) = j > 0 \), in which case the partition (of the first \( m \) objects) is the \( j \)th amongst the last \( g(k_m-1,m-1) \) partitions of the \( k_m \)th set. This means the object is in the last group (on its own!) and so

\[ n_m = k_m, \quad k_m-1 = k_m, \quad j_m - 1 = j. \]

B. Otherwise we can set \( k_m-1 = k_m \) and find \( n_m, j_m-1 \) from the conditions

\[ n_m = j_m - (j_m-1-1) k_m > 0, \quad j_m - j_m-1 k_m \leq 0. \]

This fixes object \( m \) as being in group \( n_m \); and the partition of the first \( m-1 \) objects as being the \( j_m-1 \)th in the \( k_m-1 \)th set.

Thus setting \( m = m-1 \), we can repeat steps A or B until \( m=1 \), when we set \( n_1 = 1 \); at which stage all the \( n_m \) \((m=1,2,\ldots,M)\) have been found.
Figure 1  A Directed Staged Network

\[ S_1 = (s), \quad S_2 = (1,2,3), \quad S_3 = (4,5,6), \quad S_4 = (t) \]

\[ A_2 = (a,b,...,g), \quad A_3 = (h,i,...,n), \quad A_4 = (o,p,q) \]

Event \([V] = [(2,3)]\) has occurred at stage 2.

\( A_2 \) has failure pattern \( X = (0101000) \).

\( W(V,X) = (4,5) \).
Figure 2: An Undirected Staged Network

\[ S_1 = (s) \quad S_2 = (1,2,3,4) \quad S_3 = (5,6,7) \quad S_4 = (t) \]
\[ A_2 = (a,b,\ldots,g) \quad A_3 = (h,i,\ldots,o) \quad A_4 = (p,q,r) \]

Event \( I = [(1)(2)(34);3] \) has occurred at stage 2.

\( A_2 \) has failure pattern \( X = (11010100) \)

\( J(I,X) = [(5)(67);2] \)
Figure 3  Dodecahedron

$S_1 = (s)$  $S_2 = (2, 3, 4)$  $S_3 = (5, 6, \ldots, 10)$  $S_4 = (11, 12, \ldots, 16)$

$S_5 = (17, 18, 19)$  $S_6 = (t)$

$A_2 = (1, 2, 3)$  $A_3 = (4, 5, \ldots, 12)$  $A_4 = (13, 14, \ldots, 21)$

$A_5 = (22, 23, \ldots, 27)$  $A_6 = (28, 29, 30)$

$\Pr($arc operational$)$  

<table>
<thead>
<tr>
<th>$\Pr$</th>
<th>reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>.5</td>
<td>.290155</td>
</tr>
<tr>
<td>.9</td>
<td>.997120</td>
</tr>
<tr>
<td>.95</td>
<td>.999705</td>
</tr>
</tbody>
</table>
Figure 4  A Grid Network

Reliability = 0.5 if p(arc operating) = 0.5

$S_1 = \{s\}$  $S_2 = \{1, 2, \ldots, 6\}$  $S_3 = \{7, 8, \ldots, 12\}$
$S_4 = \{13, 14, \ldots, 18\}$  $S_5 = \{19, 20, \ldots, 24\}$  $S_6 = \{25, 26, \ldots, 30\}$  $S_7 = \{t\}$
**Markov algorithms for computing the reliability of staged networks**

Certain commonly occurring types of network, whether directed or undirected, exhibit a staged structure. Two algorithms, based on node partitioning, are presented which take advantage of such structure and which use a Markov transition probability form of recursion. The algorithm for directed networks is related to the Markov chain formulation of Bailey and Kulkarni, but for undirected networks a more detailed form of state definition is used related to one suggested by Rosenthal.

The computational advantages of the algorithms are discussed and some numerical results presented.
END

DTIC

8-86