CHAPTER 3

CAPACITY CONSIDERATIONS IN RELIABILITY EVALUATION

A general communication system is represented by its probabilistic connected graph \( G = \{N,M\} \), having a set \( N = \{n_1,n_2,\ldots,n_n\} \) of \( n \) nodes (or vertices) and a set \( M = \{m_1,m_2,\ldots,m_m\} \) of \( m \) branches (or links or edges) contained in it. Any branch \( m_t \) is identified with an unordered pair \( (n_1,n_2) \) of nodes (or vertices). Any branch \( m_t \) is normally assumed to be either in the working (or good, or UP or ON) state or failed (or bad or Down or OFF) state. In other words, it is assumed that either the complete information content is fully transmitted or is not transmitted at all. In realistic situation, it may not be justified since most of the practical systems have limited capacity capabilities of their various links and have fixed overall system capacity requirements. Almost every communication system has certain requirements of information contents to be transmitted from the transmitter to the receiver. Also the various communication links of the system have specific channel capacity limitations, which may be chosen so as to minimize the delay and the cost involved in transmitting the message.

Several algorithms for determining the symbolic reliability expression for a general communication system are available in the literature [27-30]. Most of these methods, however, do not consider any limitation on the capacity of
the individual elements or that of the overall system. Recently Lee [93] has tried to solve this problem by considering the minimal paths of a directed graph and by using Lexicographic ordering.

The system in the present context of discussion, may in general be a communication system with specified capacity requirements, a computer communication network allowing a fixed amount of data exchange amongst different terminals of various computer centres; a power distribution system having various power lines of limited power ratings; a pneumatic or hydraulic system for sending some gas or fluid through a pipe line network with limited safe flows through them; a transport system of a big town with limited maximum traffic on the various roads etc.

3.1 LEXICOGRAPHIC METHOD

This method [93] based on the Lexicographic ordering on the oriented probabilistic graph of the system makes use of following assumptions in addition to the general assumptions mentioned in section 1.6:

(i) Each branch is either functioning or failed. The branch flow is bounded by the capacity of the branch i.e., $f_i \leq c_i$; no flow is possible from a failed branch.

(ii) Flow is conserved at each of the nodes.

(iii) Network is good if and only if a specified
amount of flow (assumed known) can be transmitted from the input node to the output node.

The method proceeds with the knowledge of the probabilistic or reliability logic graph (RLG) of the flow network under consideration. If all the branches are functioning i.e., \( x_i = 1 \) for all \( i \), the required flow is assumed to be transmitted from the source node to the terminal node, then the system is said to have its performance function \( \Psi(x) = 1 \). On the other hand if it is not possible for the required flow to be transmitted then the performance function \( \Psi(x) = 0 \) and the network is bad or OFF or failed. The network reliability is the probability when \( \Psi(x) = 1 \). Therefore, the problem of determining the reliability of such a flow network can be reduced to that of finding the set of vectors 
\[ S = \{ x | \Psi(x) = 1 \} \]
from which the reliability expression is calculated.

The algorithm based on this method hinges on the lexicography property. According to which:

(a) A vector is lexicographically positive (or negative) if its first non-zero component is positive (or negative). Thus the vector \( (0, 0, 3, -4, 1) \) is lexicographically positive while a vector \( (0, -2, 1, 5, 3) \) is lexicographically negative. A vector \( x' \) is lexicographically greater (or smaller) than \( x'' \) if \( (x' - x'') \) is lexicographically positive (or negative).
(b) A vector $\mathbf{x}'$ is lexicographically greater than $\mathbf{x}''$ with respect to the ordered index set $A$, if $\mathbf{x}' < A > - \mathbf{x}'' < A >$ is lexicographically positive.

3.2 **Algorithm**

The algorithm begins with any vector, say $\mathbf{x}'$ such that $\Psi(\mathbf{x}') = 1$; let $J = \{i | x'_i = 1\}$ and $\breve{J} = \{i | x'_i = 0\}$.

**Step 0:** (Initialization) Order $J$ to get an ordered set $A$ and put $A' = J$ set

$$h = \prod_{i \in A} p_i, \quad k = 1 \quad \text{and} \quad x'_i = 1 \quad \text{for all} \ i \in A$$

Put $B_0 = \{x | x_1 = 1 \text{ for } i \in A, \ x_1 = 0 \text{ or } 1 \text{ for } i \in A'\}$

Suppose that $r$ is the last element of $A$. Set $x'_r = 0$.

**Step 1:** Determine a binary $n$-dimensional vector $\mathbf{\hat{x}}$ such that $\mathbf{\hat{x}}_i = x'_i$ for all $i \in A$ and $\Psi(\mathbf{\hat{x}}) = 1$. If such an $\mathbf{\hat{x}}$ does not exist go to step 5.

**Step 2:** Put $h = h + \prod \text{Pr}\{X_i = \mathbf{\hat{x}}_i\} \prod \text{Pr}\{X_i > \mathbf{\hat{x}}_i\}$

**Step 3:** If $A' \neq \phi$, go to step 4a, otherwise put

$$E_k = \{x | x_1 = \hat{x}_1, \ i \in A\} \text{ and } k = k + 1.$$ If the last element of $\mathbf{\hat{x}} < A >$ is zero, go to step 5; otherwise go to step 4b.
Step 4a: Delete from $A'$ all elements $i \in A'$ with $\hat{x}_i = 1$ to obtain a reduced set $A'$, add those elements deleted from $A'$, in arbitrary order, to the right of $A$ to obtain an augmented ordered set $A$. Put $B_k \leftarrow \{x|x_1 = \hat{x}_1 \text{ for } i \in A, x_i = 0 \text{ or } 1 \text{ for } i \in A'\}$ and $k \leftarrow k + 1$.

Step 4b: Set $x^*_i = \hat{x}_i$ for all $i \in A$ except the last element of $A$; set the last element of $x^* < A >$ equal to 0. Return to step 1.

Step 5: If $x^*_i = 0$ for all $i \in A$, terminate.

Step 6a: Suppose $x^*_i = 1$. Reduce $A$ by deleting all elements following element $r$ in the ordered set $A$; augment $A'$ by including those elements deleted from $A$.

Step 6b: Set $x^*_i = 0$ and return to step 1. At termination find $h = R = Pr\{\Psi(x) = 1\}$.

Example 3.1: Consider a form of an ARPA network of Fig. 3.1 having directed branches and the other data as follows:

<table>
<thead>
<tr>
<th>Branch</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

The required flow capacity is of 3 units through the system.

Choose the vector $x' = (a \ d \ g)$ as $\Psi(x') = 1$.

Let $J = \{i|x'_i = 1\}$ and $J' = \{i|x'_i = 0\}$. 
Step 0: Order J to get ordered set A and put $A' = J'$

or $A = \{a, d, g\}$; $A' = \{b, c, e, f\}$

and $h \cap A$ and

$A^* = \{a, d\}$; $A'^* = \{b, c, e, f, g\}$

Begin with an initial vector $(x_a, x_d, x_g; x_b', x_c', x_e', x_f')$

Check if 3 units of flow can be transmitted from the input node to the output node if the branch 'g' is removed by procedure [7]. It is not possible for 3 units to flow and hence the network is bad. Now

$A = \{a, d\}; A' = \{b, c, e, f, g\}$

Again check if the network is good, if all branches are working except branch 'd'. A possible flow could be $f_a = 3, f_b = 2, f_c = 1, f_f = 1, f_g = 1$ and $f_d = f_e = 0$; (this in general is not unique). Therefore, $(x_a', x_d', x_g; x_b, x_c, x_f, x_e', x_f')$ is a good state. Now

$A = \{a, d, b, c, f, g\}$ and $A' = \{e\}$

Again check the network performance if all the branches except 'd' and 'g' are working. The network again will be bad.

Similarly we obtain another successful state
The symbolic reliability expression is thus given as:

\[
R = p_a p_d p_g + p_a (1-p_d) p_b p_c p_f p_g + p_a (1-p_d) p_b (1-p_c) p_e p_f p_g
\]

or

\[
R = p_a p_d p_g + p_a q_d p_b p_c p_f p_g + p_a p_b q_c q_d p_e p_f p_g
\]

The methods proposed here in this chapter are based on paths and cutset enumeration technique. Both these methods are relatively simpler and are applicable to the non-directed and directed systems. These methods are easily implemented on a computer and make use of the assumptions already mentioned in section 3.1. In these methods, nodes are also assumed perfect, but the method given in [33] can be applied easily to extend the discussion to the system where nodes also have a finite non-zero probability of failure.

3.3 METHOD BASED ON PATH ENUMERATION

In this method we search a 'group' of branches such that the successful operation of all branches in the group ensures the system success i.e., capability of transmitting the required capacity from the source node to the sink node. It is, however, assumed that this group is minimal in the sense that no subset of this group is a group by itself.

It is further emphasized that this 'group' in general does not correspond to the 'path' in the graph theoretic sense.
For example, Fig. 3.2 shows a path, but if the requirement of the system is to transmit 4 units of information from the source to the sink, then even if all the branches of this path are good, the system is not good. Similarly, for 4 units of information to be transmitted between s and t, four branches shown in Fig. 3.3 represent a minimal set of branches (and thus defined as 'group') required for the system to be good. But this set of branches a,b,c,d of Fig. 3.3 is certainly not a path in the graph theoretic terminology [94].

Such groups are determined from a knowledge of minimal paths of a network. This has been done as there are several efficient methods [81,84,86,89-91] available already for the determination of minimal paths. After all such groups are determined, they are made mutually exclusive with respect to each other by using any of the technique [e.g., 33-37] and then reliability expression immediately follows. This method follows the steps under mentioned:

**Step 1:** Determine all the minimal s-t paths of the RLG.

**Step 2:** Determine the capacities $C_{pi}$ of all minimal paths. The lowest capacity of any branch contained in a minimal path $Pi$ is the capacity of that path. For example, the capacity of the path from s to t in Fig. 3.2 is only 2 units.

**Step 3:** Examine the path capacities $C_{pi}$ for all $Pi$. A path is a valid group if and only if $C_{pi} \geq C_s$, as it can then transmit the required information content $C_s$. We search for

$$2, 3 < C_s$$
the remaining groups by examining the combination of paths. For example, in Fig. 3.3 $P_1 = a,b$ and $P_2 = a,c,d$ are minimal paths with $C_{P_1} = 2$ and $C_{P_2} = 3$. If $C_S = 4$, neither of these paths is a valid group, but a combination of both minimal paths i.e., a set $\{a,b,c,d\}$ can transmit $C_S$ and is hence a valid group. If there is any common branch or branches (in this case branch a) in the combination, the total capacity of such combination is limited to the capacity of that branch which has minimum capacity amongst such branches. The combinations of minimal paths which do not qualify as groups are temporarily listed as invalid groups. It is ensured that no groups are repeated and no subset of a group is already listed as a group.

**Step 4:** Try the other combination of paths and invalid groups with a view to search for the remaining valid groups.

**Step 5:** Make all the valid groups disjoint or mutually exclusive with respect to each other. The symbolic reliability expression is then easily determined.

The existing method proposed by Lee [93] is, of course, a good attempt towards the capacity considerations of the flow systems, but is quite cumbersome and suffers from many drawbacks as it involves the determination of the nature of the vectors. Moreover, it is to be seen at every step whether the vector is lexicographically positive or negative, which is a tedious work. In this method at every iteration, the labelling procedure [7] has to be applied in order to
The methods proposed in this chapter do not experience the difficulties presented in the method of Lee [93]. Since these methods proceed from the cutset or path enumeration, which of course can be determined for the given RLG by [82-92], it is possible to know before hand, as to which one is to be used. When the number of paths are less than the number of cutsets, the method based on the path enumeration is preferred and when the number of cutsets is smaller than the number of paths, the method based on cutsets is used. Further, if there is no severe limitation on the memory size of the computer, even the reasonably large system can be handled making use of the path enumeration based technique of section 3.3. In case the system is very large and the memory size is limited for a computing system, the method based on cutset approach of section 3.5 is preferable, because in this, we are handling only one cutset at a time. This method is very economical and makes use of lesser memory space than the method based on path approach.

3.4 ALGORITHM

This algorithm based on path enumeration technique of section 3.3 is summarised as:

Step 1: Generate a matrix $P_l$ similar to a path matrix. The row denote the minimal paths and the columns correspond to
the branches contained in the paths,

\[ P_{i,j} = \begin{cases} 
    c_j & \text{if branch } j \text{ having capacity } c_j \text{ is present in path } i \\
    0 & \text{otherwise}
\end{cases} \]

In this matrix, paths are ordered according to their cardinality; i.e., a lower row always contains number of non-zero elements at least equal to the higher row.

**Step 2:** Generate a column matrix CPI

\[ CPI_1 = \min_j \{ P_{i,j} \} \]

**Step 3:** If CPI \(_1\) (for any \(i\)) \(\geq c_s\), then the set of branches in the \(i^{\text{th}}\) row is a valid group and it is transferred as one of the rows in the binary matrix P4 by replacing all non-zero entries of the \(i^{\text{th}}\) row of P1 by 1. The element CPI \(_1\) is also transferred as an element of the new column matrix CP4 into the row corresponding to the newly generated row of P4.

**Step 4:** Another binary matrix P2 is formulated with all the non-zero entries of P1 (excluding the rows already transferred to P4) replaced by 1. The column matrix is also correspondingly modified and is renamed as CP2.

**Step 5:** Initially P5 = P2

and CP5 = CP2
Step 6: (a) \( m = 1 \)
   (b) \( n = 1 \)

Step 7: Combine (numerically add) \( m^{th} \) row of \( P2 \) with \( n^{th} \) row of \( P3 \). The capacity of this combination is determined taking into account the common branches between the two rows. If there are no common branches between the rows being combined, then the capacity of the combination is the algebraic sum of the capacities corresponding to the two rows. If there are common branches, then the capacity of the combination is the minimum of the capacities of the common branch(es) and the sum of the two capacities of the rows being combined. If the capacity of the combination is \( \geq C_s \) then it is listed in \( P4 \), provided it is not a subset of an already listed row of \( P4 \). If the capacity of the combination \( < C_s \) then it is placed in \( P3 \). The capacity of the newly added row (in \( P3 \) or \( P4 \)) is the capacity of the combination and is placed in the appropriate list (\( CP3 \) or \( CP4 \)).

Step 8: Increment \( n \) and go to step 7. Continue till all the old rows of \( P3 \) are exhausted.

Step 9: Increment \( m \) and go to step 6b. Continue till all rows of \( P2 \) are exhausted. The process is prematurely terminated if \( P3 \) is found to be empty at any stage. The entries in \( P4 \) are the valid groups. These valid groups are disjointed either by the application of method given in section 3.7 of any other existing algorithm [e.g., 33-37].
Example 3.2: Consider a bridge network of Fig. 3.4 with the data: \( c_a = 2, c_b = 3, c_c = 4, c_d = 2, c_e = 3 \) and \( c_s = 4 \).

Steps 1 and 2:

Path matrix \( P_1 = \)

\[
\begin{bmatrix}
2 & 0 & 4 & 0 & 0 \\
0 & 3 & 0 & 2 & 0 \\
2 & 0 & 0 & 2 & 3 \\
0 & 3 & 4 & 0 & 3 \\
\end{bmatrix}
\]

Step 3: No row is transferred into \( P_4 \) since \( CP_1_i < C_s \) for all \( i \).

Step 4:

\[
P_2 = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
\end{bmatrix}
\]

Continuing from steps 5 to 9 of the algorithm, we find that \( P_4 \) comes out to be:

\[
P_4 = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 \\
\end{bmatrix}
\]

The symbolic expression for the reliability of such
a system after making the two groups mutually disjoint is:

\[ R = P_a P_b P_c P_d + P_a P_b P_c P_e P_d \]

**Example 3.3:** Considering the same example as that of Lee [93]

\[
\begin{bmatrix}
6 & 2 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 3 & 0 & 0 & 3 \\
0 & 0 & 0 & 0 & 2 & 2 & 3 \\
6 & 0 & 1 & 0 & 0 & 2 & 3
\end{bmatrix}
; \quad
\begin{bmatrix}
2 \\
3 \\
2 \\
1
\end{bmatrix}
\]

Path matrix \( P_1 \):

Since \( C_s = 3 \), row 2 gets transferred to \( P_4 \) and remaining non-zero entries are transferred into \( P_2 \) as 1's

\[
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 1
\end{bmatrix}
; \quad
\begin{bmatrix}
2 \\
2 \\
1
\end{bmatrix}
\]

Trying the combinations while taking two at a time and so on through steps 5 to 9, the final \( P_4 \) is

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1
\end{bmatrix}
; \quad
\begin{bmatrix}
3 \\
4 \\
3
\end{bmatrix}
\]

The valid groups are \( a, d, g \); \( a, b, c, f, g \); \( a, b, e, f, g \);
After disjointing these groups, the reliability expression becomes,

\[ R = p_a p_d P_G + p_a p_b p_c q_d P_f P_g + p_a p_b p_c q_d P_e P_f P_g \]

This result is the same as that reported in [93].

**Example 3.4:** An ARPA network with seven branches as shown in Fig. 3.5, the branches have capacities \( c_a = 2, c_b = 2, c_c = 3, c_d = 2, c_e = 1, c_f = 3, c_G = 4 \) and \( c_S = 4 \).

Path matrix \( P_1 = \begin{bmatrix} 0 & 0 & 3 & 0 & 1 & 3 & 0 \\ 2 & 0 & 0 & 2 & 0 & 0 & 4 \\ 2 & 2 & 3 & 0 & 0 & 0 & 0 \end{bmatrix} \) ; \( CP_1 = 1 \)

Finally valid group matrix

\[ P_4 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \] ; \( CP_4 = \begin{bmatrix} 4 \\ 5 \\ 4 \end{bmatrix} \)

There are three valid groups \( a,d,f,g; a,b,c,f,g; a,b,e,f,g \); which can be disjointed and the reliability expression:

\[ R = p_a p_d p_G + p_a p_b p_c q_d P_f P_g + p_a p_b p_c q_d P_e P_f P_g \]

Computer time used for this example was 0.26 seconds.
3.5 METHOD BASED ON CUTSET APPROACH

This method is based on the cutset approach and proceeds with the knowledge of all the minimal cutsets. In any graph $G$ of $m$ edges and $n$ nodes, the order of the number of cutsets is $2^{n-2}$ whereas the order of the number of paths between any pair of nodes is $2^{m-n+2}$. For networks having nodes of average degree greater than four, $m > 2n$ and $2^{m-n+2} > 2^{n-2}$ \[^{[64]}\]. Consequently, such networks have a larger number of paths than cutsets. Computation time would generally be reduced in such cases by calculating network reliability from cutset approach instead of path approach.

In this method, the various cutsets of the RLG are examined one by one. For every cutset it is examined, whether any subset of branches contained in cutset forms a 'valid cut group' so as to defeat the purpose of completely transmitting the required system capacity. The various subsets of every cutset are examined for searching the valid cut groups. It is ensured, that in this process no combination of the subsets are formed which are already considered valid cut groups. These valid cut groups are not generally the cutsets in the graph theoretic terminology.

3.6 ALGORITHM

The algorithm based on the procedure described in section 3.5 has the following steps:
Step 1: Formulate a matrix $K$ similar to the fundamental cutset matrix, in which the rows correspond to the various cutsets, while the columns indicate the branches contained in the particular cutset. All the non-zero entries (corresponding to the presence of the branch in the cutset) are the capacities of the individual branches contained in the cutsets,

$$K_{ij} = \begin{cases} c_j ; & \text{if the } j^{th} \text{ branch having capacity } c_j \text{ is contained in } i^{th} \text{ cutset} \\ 0 ; & \text{otherwise} \end{cases}$$

Although not necessary, it is convenient to have the first cutset such that only source node is one of the subgraphs, and the second cutset such that only the sink node is one of the subgraphs. The remaining rows of the matrix are ordered according to the cardinality of the cutsets.

Step 2: A column matrix $CK$ is also generated which has its $i^{th}$ element as the sum of all the non-zero entries in $i^{th}$ row of matrix $K$, i.e.,

$$CK_i = \sum_j c_{ij}$$

Step 3: Compare the value of the elements $CK_i$ with each of the non-zero capacity value $c_{ij}$ in the $i^{th}$ row; for any particular value of $j$.

If $(CK_i - c_{ij}) \geq C_s$, then the corresponding branch $m_j$ is not a valid cut group. On the other hand, if $(CK_i - c_{ij}) < C_s$
then the corresponding \( m_j \) is a valid cut group.

**Step 4:** The combinations are now formed of the branches which do not form valid cut groups, taking two at a time, three at time and so on. The capacity of any such combination, \( c_x \), is equal to the sum of the capacities of all branches contained in this combination.

If \( (C_{K_i} - c_x) \geq C_g \); then the combination is not a valid cut group.

If \( (C_{K_i} - c_x) < C_g \); then \( c_x \) is a valid cut group.

It is ensured here that the combination of the cut groups already considered as valid cut groups are not formed again and the newly generated combination is not the subset of the cutgroups considered earlier.

**Step 6:** The process is repeated from step 3 to step 5 for all \( i \).

**Example 3.5:** A bridge network, shown in Fig. 3.4, has the same data as that of Example 3.2.

Minimal cutsets are \( a,b; c,d; a,e,d; b,e,c; \)

**Steps 1 and 2:** The cutset matrix \( K \) and the matrix \( C_{K} \) corresponding to the capacities of the cutsets are:
Matrix $K = \begin{bmatrix} 2 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 & 0 \\ 2 & 0 & 0 & 2 & 3 \\ 0 & 3 & 4 & 0 & 3 \end{bmatrix}$; $CK = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 10 \end{bmatrix}$

**Step 3(i):** Compare the first row i.e., cutset $a,b$ with $C_s$,

$5 - c_a = 5 - 2 = 3$, which is $< C_s$.

Branch 'a', therefore, forms a valid 'cut group' and is listed separately,

$5 - c_b = 5 - 3 = 2$, which is $< C_s$; thus branch 'b' is also a valid cut group.

**Step 4:** All the non-zero entries of $K$ corresponding to 'a' and 'b' column are made zero.

The matrix $K$ now takes the new form while $CK$ remains the same,

$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 & 0 \\ 0 & 0 & 0 & 2 & 3 \\ 0 & 0 & 4 & 0 & 3 \end{bmatrix}$; $CK = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 10 \end{bmatrix}$

**Step 3(ii):** Consider the 2nd row i.e., the cutset $c,d$;

$6 - c_c = 6 - 4 = 2$; which is $< C_s$,
which allows 'c' also as a valid cut group.

\[ 6 - c_d = 6 - 2 = 4 \]; which is equal to \( C_s \),
thus 'd' is not a valid cut group.

Since 'd' is the only invalid cut group, combinations can not be formed. Applying step 4, the non-zero entries corresponding to column corresponding to 'c' are again made zero; after which \( K \) becomes:

\[
K = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2 & 0 \\
0 & 0 & 0 & 2 & 3 \\
0 & 0 & 0 & 0 & 3
\end{bmatrix} \quad \text{and} \quad CK = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 10 \end{bmatrix}
\]

**Step 3(iii):** Considering the third row of this \( K \) of step 3(ii):

\[ 7 - c_d = 7 - 2 = 5 \]; which is \( > C_s \),
thus the branch 'd' is not a valid cut group.

Again
\[ 7 - c_e = 7 - 3 = 4 \]; which is equal to \( C_s \),
thus branch 'e' is also not a valid cut group.

Now form the combinations of 'd' and 'e' i.e., only one in this case. Capacity \( c_x = c_d + c_e = 2 + 3 = 5 \). Then
\[ 7 - c_x = 7 - 5 = 2 \]; which is \( < C_s \), thus 'd,e' is a valid cut group.
Step 4 cannot be applied as there is no single element cut group.

Step 3(iv): Considering the fourth row of K of step 3(ii),

\[ 10 - c_e = 10 - 3 = 7 > C_s \] making 'e' as an invalid cut group.

Now the procedure is terminated and the valid cut groups are listed as \{a\}, \{b\}, \{c\} and \{d,e\}. The unreliability expression \( Q \) is determined by disjointing these cut groups successively with the procedure mentioned in section 3.7 and is illustrated below:

Cut group set \( C_e = a + b + c + d + e \)

The 1st term i.e., 'a' is disjointed with all other terms to its right,

\[ a + \bar{a}b + \bar{a}c + \bar{a}d + e \]

disjointing 2nd term \( \bar{a}b \) with terms on its right,

\[ a + \bar{a}b + \bar{a}b \bar{c} + \bar{a}b \bar{d} + e \]

disjointing 3rd term \( \bar{a}b \bar{c} \) with the term to its right,

\[ a + \bar{a}b + \bar{a}b \bar{c} + \bar{a}b \bar{d} \bar{e} \]

Thus unreliability expression follows:

\[ Q = q_a + p_aq_b + p_ap_bq_c + p_ap_bp_cq_dq_e \]

and the reliability \( R = 1 - Q \).
Example 3.7: The ARPA network of Fig. 3.5 is considered and the data is the same as that of Example 3.4.

Minimal cutsets are: \( a, f; c, g; d, b, f; b, e, g; a, d, e, g; c, d, e, f \)

The cutset matrix \( K \) and the matrix \( CK \) corresponding to the capacities of the cutsets are:

\[
K = \begin{bmatrix}
2 & 0 & 0 & 0 & 0 & 3 & 0 & 5 \\
0 & 0 & 3 & 0 & 0 & 0 & 4 & 7 \\
0 & 3 & 0 & 2 & 0 & 3 & 0 & 9 \\
2 & 0 & 0 & 2 & 1 & 0 & 4 & 8 \\
0 & 0 & 3 & 2 & 1 & 3 & 0 & 9
\end{bmatrix}
\]

\[
CK = \begin{bmatrix}
5 \\
7 \\
8 \\
8 \\
9
\end{bmatrix}
\]

By applying the various steps of the algorithm 3.6. The valid cut groups come out to be:

\( \{a\}, \{f\}, \{g\}, \{b,d\}, \{c,d,e\} \)

Cut group set \( C_g = a + f + g + b \text{d} + c \text{d} e \)

These terms are made mutually exclusive or disjointed and the expression for the unreliability comes out to be:

\[
Q = q_a + p_a q_f + p_a p_f q_g + p_a q_b q_d p_f p_g + p_a p_b q_c q_d q_e p_f p_g
\]

From which \( R = 1 - Q \).
3.7 **DISJOINTING PROCEDURE**

The suggested procedure used in the methods 3.5 and 3.6 is briefly reproduced here. It may, however, be noted that any other method could be used as well. The union of all the valid cut groups \( x_1 \) is considered and it is called as cut group set \( C_g = x_1 U x_2 U x_3 U \ldots; x_1, x_2, \ldots \) are the Boolean functions ordered according to their cardinality. First of all, the term \( x_1 \) is considered and all other terms to the right of this are made disjoint with \( x_1 \). A check is then applied to see that no subset of \( x_1 \) is generated in this process and if it does, it is ignored. After that the newly generated second term from left is considered and disjointed with all the other terms to its right.

The process is continued till we reach the last term on the right. At every iteration the check for deleting the subsets of the disjointed terms is applied. The procedure is terminated when all the terms are exhausted and the expression for the symbolic unreliability \( Q \) follows.

This procedure is general and can be applied to the method 3.3 also, where the groups of the valid paths is to be disjointed. In that case, the resulting expression will give us the symbolic reliability \( R \).

Two methods are explained in this chapter for deriving the expression for the symbolic reliability in some practical and realistic systems. These methods are much
simpler as compared to the existing method described in section 3.1. These methods are easily computerizable and make use of much lesser computation. The algorithm at section 3.4 is implemented on DSC-20 system using FORTRAN IV and program listing is given in the Appendix A-I.