An Efficient Cutset Approach for Evaluating Communication-Network Reliability With Heterogeneous Link-Capacities

Sieteng Soh, *Member, IEEE,* and Suresh Rai, *Senior Member, IEEE*

***Abstract*—This paper presents an efficient cutset approach to compute the reliability of a large communication network having heterogeneous link capacities. The reliability measure has been defined as capacity related reliability (CRR). The proposed method, subset cut technique (SCT), requires the cutset information of the network. For each minimal cut C_i, and a given minimum bandwidth requirement $W_{\min}$, the method enumerates all nonredundant subset cut (SC), where each SC relates link capacities & minimum bandwidth requirements. Note that if all links in an SC fail, the capacity of the induced cut C_i will be less than $W_{\min}$. Given the failure probability of each link, and the nonredundant SC, any Boolean technique for generating mutually disjoint terms can be utilized to obtain a capacity related unreliability (CRU) of the network. Thus, the CRU for an (s,t) node pair is the probability that the network has a capacity of less than $W_{\min}$ for the given node pair. Note that $\mathrm{CRR} = 1 - \mathrm{CRU}$. Two SCT algorithms are proposed: Algorithm-1, and Algorithm-2; and the suitability of using either algorithm is also discussed. Examples are given to illustrate the techniques. The time complexity, and the proof of correctness for the proposed algorithms are also included. It is shown empirically that the time complexity of generating the nonredundant SC of a network is polynomial in the order of the number of cuts of the network. The proposed SCT algorithms have been implemented in C. We have utilized the SCT to generate the CRR of some large communication networks with various $W_{\min}$ values.**

***Index Terms*—Capacity related reliability, communication network, cutset, network flow, network reliability, sum of subsets, terminal reliability.**

I. Introduction

VARIOUS measures for the reliability index of a communication network are proposed in the literature [1]–[19], [24], [25]. In one measure, the reliability index of the network is represented by its terminal reliability, which is defined as the probability that there exists at least one path for a given source-destination (s,t) node pair of the network [1]–[5]. Here, we assume that the network has equal link capacity, or the link capacities are large enough to sustain the transmission messages (packets) of any bandwidth (size). However, this assumption is unrealistic. The link capacity is a function of cost, and is limited; and each link in a network may have different capacity. Moreover, almost every communication system has a certain required capacity of information contents to be transmitted, and hence there can be a minimum bandwidth requirement, $W_{\min}$, from the source node to the terminal node of the network in order for the network to be considered to operate successfully. In setting up a video conferencing, as an example, the two connecting parties must agree on the minimum bandwidth requirement. Similarly, establishing a virtual private network between two remote sites of a company also requires such minimum bandwidth [26]. With these additional constraints, we consider that the two connecting nodes can communicate successfully only if the network has at least the required minimum capacity of $W_{\min}$. The network reliability is, thus, measured as the probability that the network has, at least, a minimum bandwidth of $W_{\min}$ between the (s,t) node pair. Reference [15] calls such performance index as capacity related reliability (CRR). Note that this reliability measure can also be used as the performance index of other networks, such as a power distribution network, a transportation network, or a water distribution network.

Several researchers [6]–[19] have proposed techniques to compute the CRR. Some of the techniques require only the pathset [8]–[12], or the cutset information [17], [18]; some others [14]–[16] in addition to the pathset, require the cutset to help compute the capacity of the generated composite paths [15], while methods in [13], [19] build a branching-tree (with disjoint tree nodes) directly without using minimal paths and/or cuts. Method [6] requires large memory sizes for solving large networks, and hence, is not efficient. On the other hand, Lee's technique [7] is quite involved & cumbersome, and also the method is restricted to directed graphs [9]. Misra & Prasad [8], and Aggarwal, *et al.* [9] propose two-step approaches to obtain CRR. In the first step, their algorithms [8], [9] start with the pathset information of the evaluated network to generate composite paths. In the second step, any Boolean algebra to obtain reliability expression [4], [5] is used to generate the CRR from the composite paths. However, these algorithms fail, in general, to generate correct results [11], [15]. The method in [8] fails to generate all success composite paths, while the basic problem with the method in [9] lies in the procedure used to compute composite path capacity. Aggarwal [10], [11] has introduced the concept of "weighted reliability index" to benchmark the performance of a communication network with heterogeneous link capacities. In [11], he also proposes a new method to calculate the capacity of a subnetwork formed by any subset of paths (i.e., the composite paths). Varshney, *et al.* [12] generalize the method in [11] to a multistate graph. However, the procedures used to calculate the maximum flow in [11] & [12], in general, do not

Manuscript received July 26, 2002; revised May 23, 2003. The work of S. Rai was supported in part by the NSF grant CCR 0073429. Associate Editor: W. H. Sanders.

S. Soh is with the Department of Computing, Curtin University of Technology, Perth, WA, Australia (e-mail: soh@cs.curtin.edu.au).

S. Rai is with the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA, USA (e-mail: suresh@ece.lsu.edu).

Digital Object Identifier 10.1109/TR.2004.842530

produce correct results [15], [20], [21]. Schanzer [20] suggests two ways to correct the problems, while Kyandoghere [21] provides a correction of the procedure. Rueger [13], and Rai, *et al.* [14] have proposed algorithms which obtain a symbolic CRR expression under a capacity constraint. However, their algorithms suffer from the drawback of generating a huge number of redundant paths/cuts. The algorithms, thus, are impractical even for moderate size graphs where the number of paths is more than ten. Recently, Jane, & Yuan [19] proposed an algorithm to improve the method in [13]. Their improved algorithm [19] is shown to run faster, and to be more robust than Rueger's method, with a tradeoff for producing an average of four times larger number of disjoint terms than those generated in [13]. Furthermore, their labeling scheme approach is more suitable for networks with directed links, and the steps 3–5 of their algorithm may be repeated exponentially [19].

Rai & Soh [15] proposed a technique to enumerate all success states of the network for the CRR problem. In addition, they provide a new method to compute the capacity of a composite path by utilizing the max-flow min-cut theorem [22]. However, each capacity computation needs all the cuts of the network, and the computation is done for each generated composite path; hence, the overall compensation is expensive. Recently, Lee & Park [16] proposed a new method which reduces the number of composite paths generated from minpaths, and thus reduce the number of capacity computations. However, the time complexity (to generate the composite paths) of the methods in [15], [16] is exponential in the order of the number of simple paths of the network.

In most practical systems, the number of cuts is much smaller than the number of paths [17], and hence computing CRR from cutset information is expected to be more efficient. The method in [17] requires only cutset information to determine the CRR. The method computes the CRR in two main steps. In the first step, the method enumerates all possible nonredundant subsets of a cut C_i, for all i, each of which is capable of blocking the transmission of messages of size $W_{\min}$ from s to t in the given network. Note that such subset is called a *valid cut* in [17]. In the second step, the method uses any existing sum-of-disjoint products algorithm [4], [5] to compute the CRR from the *valid cut* groups. To generate the *valid cut* group, the method [17] uses a cutset matrix, A, to represent the cutset of the network. Rows of the matrix correspond to the various cutset, and columns indicate the links contained in the particular cutset. The nonzero entries in the matrix show the link capacities. Utilizing the matrix, the method generates a column matrix CA in which its ith entry is computed as $CA_i = \sum_j C_{ij}$. Note that CA_i is the cut capacity for C_i. The method enumerates the valid cut from matrices A & CA. Starting from the first row in A, the algorithm computes $CA_i - C_{ij}$. If the result is less than $W_{\min}$, a 1-link *valid cut* is found, and all nonzero entries of A in column j are made zero. Then, from the remaining nonzero entries in that row, the algorithm enumerates all possible two combinations, three combinations, and so on; and each of them is checked for being *valid-cut*, i.e., $CA_i - C < W_{\min}$, where C is the sum of all link capacities in the combination. Note that while generating the two or more links forming a *valid-cut*, the method also deletes any redundant *valid cuts*. This process is repeated for each row of the matrix. However, the method [17] is efficient only for computing CRR of small networks where some 1-link *valid cuts* exist. For large networks without or with a small number of 1-link *valid cuts*, the method has to generate all possible combinations of subsets for all the cuts, and hence, is not efficient. As an example, with $W_{\min} = 4$, the network shown in Fig. 2 contains no 1-link valid cut, and thus for a cut {1,3,4,8,13,14}, the method has to generate $\binom{6}{2}$ possible 2-link, $\binom{6}{3}$ possible 3-link, $\binom{6}{4}$ possible 4-link, and $\binom{6}{5}$ possible 5-link to generate three nonredundant *valid cuts*: {3,4,13,8,14}, {1,3,4,13,14}, and {1,3,4,8,13). Thus, for each cut C_i with cardinality L_i, 2^{L_i} combinations of links are generated; and hence this step is very time consuming. Furthermore, the method fails to detect for system failure when we set $W_{\min} > W_{\max}$, where $W_{\max}$ is the maximum capacity flow through the network [22]. Thus, a more efficient algorithm is needed to compute the CRR of large networks. Soh & Rai [18] have proposed a cutset-based algorithm to solve the CRR. Note that the difference between the method in [17] and that in [18] lies in the *valid cut* enumerations. In [18], the *valid cuts*, referred as modified power set (MPS), are generated more efficiently; and hence CRR can be computed faster. The method in [18] generates the MPS by utilizing only links with capacity less than $W_{\min}$ in each cut. Because the method in [18] generates subsets from fewer links than those required in [17], the method in [18] is more efficient compared to that in [17]. The method presented in this paper is an improvement over the algorithm in [18].

The layout of the paper is as follows. Section II describes the CRR concepts, and other related issues. In Section III, the subset cut technique (SCT) is introduced, and two algorithms, Algorithm-1, and Algorithm-2, to enumerate the SC are presented. The section provides some definitions to help reduce the time complexity of the algorithms. In addition, implementation issues for both algorithms are discussed. Several examples illustrating the concepts and correctness proof of the methods are also given in the section. In Section IV, the time complexities of the proposed methods are derived. Section V discusses the CRR results for some communication networks, and concludes the paper.

II. Terminology

In the graph model $G(V, E)$ of a communication network, where vertices (V) represent communication centers, and links (E) denote connection services, consider each link j has a finite capacity w_j which is known a priori. A flow in a network is a function assigning a nonnegative number f_j to each link j so that $f_j \leq w_j$, and for a vertex (that is neither source nor terminal) the in- & out-flow are the same (flow conservation). Thus, w_j provides a bound on a flow passing through link j. The network is good if and only if a specified amount of signal capacity $W_{\min}$ can be transmitted from the input to the output node, or (s, t) node pair. This condition implies that the network must have at least $W_{\min}$ bandwidth/capacity for the given (s, t) node pair.

A link j is said to be UP (DOWN) if it is functioning (failed). An UP (DOWN) link is denoted by j $(\bar{j})$. Note, the nodes (V) are assumed to be always good. An (s, t) cut is a disconnecting set. All communications between a prescribed (s, t) node pair is disrupted once all the links in (s, t) are not operational. An (s, t) cut i, C_i, is minimal if no proper subset of it represents

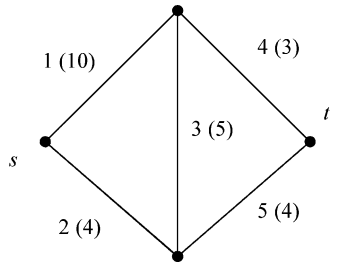


Fig. 1. Bridge network.

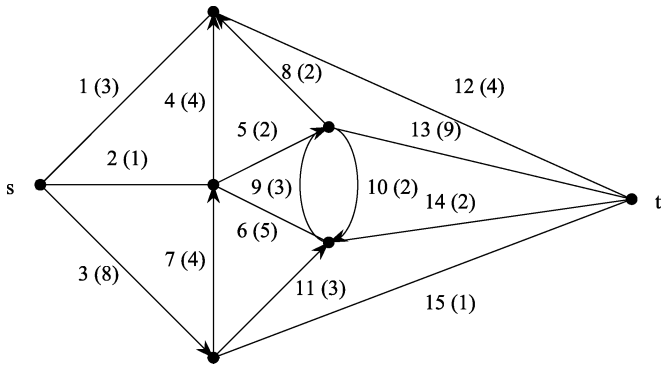


Fig. 2. 7 node, 15 link network.

a 'cut.' The cutset $C_{s,t}$ is the set of all minimal cuts for the graph $G(V,E)$. The capacity of a cut C_i, $W(C_i)$, is the sum of link-capacities in C_i. From max-flow min-cut theorem [22], the maximum capacity flow, $W_{\max}$, through the network is given as the minimum of $W(C_i)$. As an example, consider a bridge network as shown in Fig. 1. The figure shows the link capacities in the brackets. The cutset $C_{s,t}$ is given as: $C_1 = (1,2)$, $C_2 = (1,3,5)$, $C_3 = (2,3,4)$, and $C_4 = (4,5)$. The capacities $W(C_i)$ are $W(C_1) = 14$, $W(C_2) = 19$, $W(C_3) = 12$, and $W(C_4) = 7$; and hence, $W_{\max} = \min\{14,19,12,7\} = 7$.

A simple path i, P_i, for an (s,t) node-pair is formed by the set of UP links such that no nodes are traversed more than once. Any proper subset of simple paths does not result in a path between the node pair. The pathset $P_{s,t}$ is a set whose elements are simple paths. The capacity of a simple path P_i, $W(P_i)$, is obtained from the capacities of UP links contained in P_i, and is given as the minimum capacity of the links in P_i. Thus, the capacity $W(P_1)$ for a $P_1 = (1,4)$ in Fig. 1 is computed as $W(P_1) = \min\{w_1, w_4\} = 3$. If $W(P_i) \geq W_{\min}$, a simple path P_i, in addition to satisfying the connectivity requirement, fulfils the capacity constraint too. The path P_i is then called a success state of $G(V,E)$. Otherwise, the P_i represents a failed state. In the event that link capacities are infinitely large, all simple paths form success states because they do provide (s,t) connectivity, and their successes ensure the system or network success. However, for a finite capacity situation, all simple paths may or may not lead to the success states of $G(V,E)$. Depending on $W_{\min}$, some or all simple-paths may fail to satisfy the capacity constraint. Thus, simple path, and cut, two important concepts in terminal reliability, has to be revisited while considering the CRR measure. The concept of composite path & its capacity determination [15], and subset cut (introduced in Section III) are the steps in this direction.

III. The Subset-Cut Technique (SCT)

This section introduces the subset cut (SC) concept, and some definitions which lead us in developing the SCT to compute the CRR.

Definition: A subset-cut, $SC_{i,j}$, for $j > 0$, is a subset of a cut C_i with capacity $W(SC_{i,j}) > W(C_i) - W_{\min}$.

The capacity $W(SC_{i,j})$ is the sum of link-capacities in $SC_{i,j}$. If all links in $SC_{i,j}$ fail, the capacity of the induced cut C_i between the given (s,t) node pair of a network will be less than $W_{\min}$. Note that when $W_{\min} = 1$, and all links in the network have equal link capacity $w_k = 1$, each $SC_{i,j}$ reduces to the cut of the graph theoretic terminology (i.e., equal to C_i). Also note that a cut C_i is always an SC.

For any i, j, p, and q, an $SC_{i,j}$ is called a redundant SC if there exists an $SC_{p,q}$ as its subset. This redundancy can occur among the SC generated from a cut C_i (called *internal redundant*), and may also occur among the SC generated from all cuts in the network (called *external redundant*). We define a *minimal-subset-cut*, $MSC_{i,j}$, as a non internal-redundant SC generated from a cut C_i; and a *network-minimal-subset-cut*, $NMSC_{i,j}$, as a non external-redundant $SC_{i,j}$. A C_i is an internal redundant SC if the cut contains at least one $SC_{i,j}$. Given the cutsets, link capacities, and a required $W_{\min}$ of a network, the proposed SCT algorithms generate the NMSC for the network.

It is obvious that the SC concept is the reverse of the composite path [15]. An SC is a subset of any cut C_i of a network whose link failures (DOWN) are able to block the required system capacity, $W_{\min}$, between a given (s,t) node pair in the network, whereas a k-composite path [15] is the composition of k paths whose link successes (UP) are able to meet the required bandwidth for the given (s,t) node pair. The capacity related unreliability (the probability that the network has a capacity of less than $W_{\min}$ for the given node pair), CRU, is computed from the enumerated NMSC. Any Boolean techniques [1], [2], [4], [5] can be used to generate the mutually disjoint expression for the NMSC. An expression (value) for the CRU, and hence the CRR $(= 1 - \mathrm{CRU})$, is obtained by substituting each DOWN (UP) link i $(\bar{i})$ in the mutually disjoint forms with its probability of failure (success), $q_i(1 - q_i)$.

A. The SC Enumeration Technique

In general, computational complexity of the SC enumeration depends on:

(i) the total number of subsets of links generated for each C_i,
(ii) the total number of internal (external) redundant SC generated/removed, and
(iii) the total number of cuts in the network that are used to generate SC.

Therefore, to design an efficient SC enumeration algorithm, we need to reduce the values of these three parameters.

The problem to generate the $SC_{i,j}$ from a C_i is similar to the well-known *sum-of-subsets* problem [23]. In the sum-of-subsets problem with n *distinct* positive numbers or weights, we generate all combinations of these numbers whose sum is *equal* to a given integer M. On the other hand, in our SC enumeration problem, there are L_i *nondistinct* positive numbers or

link capacities, and we enumerate all *minimal* combinations of these capacities (and hence links because each link has a one-to-one correspondence with its capacity) whose sum is *larger* than $W(C_i) - W_{\min}$. Note, L_i is the total number of links in C_i. Obviously we can use any sum-of-subsets algorithm to generate all the SC by running the algorithm for each M larger than $W(C_i) - W_{\min}$. Then, taking care of redundant SC, we obtain all the MSC. However, for a general network with $W_{\min} \ll W(C_i)$, this approach will generate an excessive number of subsets, most of which are redundant SC, and hence the approach is not efficient. Furthermore, the sum-of-subsets algorithms [23] do not take advantage of *non distinct* link capacities, and other network properties which can be used to help reduce the time complexity of enumerating the SC.

The SC for a given cut C_i can also be enumerated by, first, exhaustively taking all possible combinations (subsets) of links in C_i, where each of the generated subsets is tested for being an SC. Second, we remove the internal redundant $SC_{i,j}$ to generate the set of $MSC_{i,k}$. We repeat these two steps for all cuts of the network. Finally, we remove the external redundant SC to obtain the set of NMSC for the network. However, this simple (exhaustive) method is not efficient. For each cut C_i containing L_i links, this method has to generate 2^{L_i} subsets, most of which are not SC. Furthermore, we also have to remove many internal & external redundant SC to generate the NMSC. Therefore, we need to devise an efficient algorithm to avoid generating the exponential number of subsets, and to reduce the number of internal & external redundant SC.

In the following subsections, we propose two SCT algorithms to generate SC: Algorithm-1, and Algorithm-2. To reduce the number of subsets generated for each C_i, these two algorithms generate subsets only from links each with capacity less than $W_{\min}$. Furthermore, the proposed algorithms reduce the computational complexity of generating the NMSC by removing some cuts a priori, should the cuts be detected to eventually generate redundant SC. The proposed Algorithm-1 is efficient for enumerating the NMSC for small to moderate sized networks, while the SCT Algorithm-2 is suitable for obtaining the NMSC of large networks, where there are some sets of links each with the same link capacity.

B. SCT Algorithm-1

This proposed SCT Algorithm-1 reduces the computational complexity of SC generation by reducing the total number of subsets enumerated for each cut C_i, and by removing some C_i, which will eventually produce redundant SC. Furthermore, this Algorithm-1 employs a greedy approach. The method will stop generating additional subsets once it detects that larger subsets will generate only redundant SC. In this greedy approach, the total number of subsets generated depends on the link capacities, and the required $W_{\min}$. In the following, we define several concepts which are needed in the SCT Algorithm-1.

Definition: A sum-of-subsets-less, $SSL_{i,j}$ (for $j = 1, 2, \ldots$), is a subset of a cut C_i which has a sum of link capacities less than $W_{\min}$.

Utilizing the SSL, we propose the following theorem to generate the $SC_{i,j}$ for a cut C_i.

Theorem 1: $SC_{i,j} = C_i - SSL_{i,j}$, where '$-$' is a set *difference* operation.

Proof: Because each link has a nonnegative capacity, from the theorem we get the following equality: $W(SC_{i,j}) = W(C_i) - W(SSL_{i,j})$. To prove the correctness of this theorem, we need to show that each $SC_{i,j}$ generated using Theorem 1 meets its definition, i.e., $W(SC_{i,j}) > W(C_i) - W_{\min}$. To show this, we consider two cases.

Case 1: $|SSL_{i,j}| = 0$. Substituting $|SSL_{i,j}| = 0$ to the theorem, we obtain $W(SC_{i,j}) = W(C_i)$. For $W_{\min} > 0$, it is obvious that $W(SC_{i,j}) > W(C_i) - W_{\min}$; and hence the theorem is proved for this case.

Case 2: $|SSL_{i,j}| > 0$. Because, by definition, $W(SSL_{i,j}) < W_{\min}$, substituting the $SSL_{i,j}$ in the theorem with $W_{\min}$, we obtain $W(SC_{i,j}) > W(C_i) - W_{\min}$, and hence the theorem is also proved for this second case. □

Following Theorem 1, the $SC_{i,j}$ for a cut C_i can be easily generated once we obtain all their corresponding $SSL_{i,j}$. The $SSL_{i,j}$, in turn, are generated from the links in a cut C_i. However, because $W(SSL_{i,j}) < W_{\min}$ (by definition), it is obvious that each link k in a $SSL_{i,j}$ has a link capacity $w_k < W_{\min}$; and hence the SSL can be more efficiently enumerated by considering only the links in C_i, each with link capacity less than $W_{\min}$. Here, we define a $small\_link_i$ as a subset of a cut C_i, which contains links each of which has link capacity less than $W_{\min}$. Therefore, the $SSL_{i,j}$ can be more efficiently generated from a $small\_link_i$.

The advantages of generating the SC following Theorem 1 are two-fold. First, we notice that, in general, and for various values of $W_{\min}$, the number of links in a $small\_link_i$ $(= \tau_i)$, is less than the number of links in C_i $(= L_i)$; and hence, enumerating SSL from a $small\_link_i$ is more efficient than obtaining SC directly from the corresponding cut C_i. Second, this approach does not necessarily generate all 2^{τ_i} subsets from each $small\_link_i$. For $k < \tau_i$, if each subset of size k has capacity greater than or equal to $W_{\min}$, then subsets of size $k+1$ should not be generated because they will not produce any more SSL. This *greedy* approach avoids the unnecessary generation of subsets.

The problem of generating all $SSL_{i,j}$ from a $small\_link_i$ is also similar to the sum-of-subsets problem [23]. Here, there are τ_i *nondistinct* positive numbers or link capacities, and we generate all combinations of these capacities (and hence links because each link has one-to-one correspondence with its capacity) whose sum is *less* than $W_{\min}$. Thus, the SSL can be generated by exhaustively taking all possible 2-subsets, 3-subsets, all the way up to τ_i-subset from the $small\_link_i$, where the capacity of each subset should be less than $W_{\min}$.

However, some redundant SSL may still be generated. Note, these redundant SSL, in turn, will result in internal redundant SC, and hence should be deleted. We consider a $SSL_{i,j}$ redundant if there exists a $SSL_{i,k}$ as its superset. Here, we define a *maximum-sum-of-subsets-less*, $MSSL_{i,j}$, as a non redundant SSL in a cut C_i. The $MSC_{i,j}$ for a cut C_i can be generated either by removing all redundant SC, or by enumerating them from the $MSSL_{i,j}$. The correctness of this second approach is validated by the following lemma.

TABLE I
CUTSET FOR THE NETWORK IN FIG. 2

C_1={1 2 3}	C_{10}={1 2 7 11 15}
C_2={2 3 12}	C_{11}={1 3 4 9 14}
C_3={3 6 12}	C_{12}={1 4 6 11 15}
C_4={1 3 4 6}	C_{13}={1 4 9 14 15}
C_5={3 9 12 14}	C_{14}={2 7 11 12 15}
C_6={3 12 13 14}	C_{15}={1 2 7 9 14 15}
C_7={6 11 12 15}	C_{16}={1 3 4 8 13 14}
C_8={9 12 14 15}	C_{17}={1 4 8 13 14 15}
C_9={12 13 14 15}	C_{18}={1 2 5 7 8 13 14 15}

Lemma 1: $MSC_{i,j} = C_i - MSSL_{i,j}$, where '$-$' is a set *difference* operation.

Two theorems are now presented. Theorem 2 can be used to detect a priori network failures for a required $W_{\min}$, while Theorem 3 is used to delete some of the cuts which eventually produce redundant SC, and hence this action reduces the complexity of enumerating the SC.

Theorem 2: For any i & j, and a given $W_{\min}$, if there exists an empty set $SC_{i,j}$, the network is always failed.

Proof: From Theorem 1, an empty set $SC_{i,j}$ implies $SSL_{ij} = C_i$. When $C_i = SSL_{ij}$, $W(C_i) < W_{\min}$ because, by definition, $W(SSL_{ij}) < W_{\min}$. From max-flow min-cut theorem, the capacity of a network $(W_{\max})$ is given as the minimum of all $W(C_i)$ for all i, and thus, $W_{\max}$ can not be larger than $W(C_i)$. If $W(C_i) < W_{\min}$, then it follows that $W_{\min} > W_{\max}$; hence, the network always fails. □

Example: Consider $W_{\min} = 8$, and a bridge network shown in Fig. 1. Given $C_2 = \{4, 5\}$, we obtain $SSL_{2,1} = \{4, 5\} = C_4$. Following Theorem 2, there is no success state in the network for $W_{\min} = 8$. Note, $W_{\min}(= 8) > W_{\max}(= 7)$.

Theorem 3: Consider a subset cut $SC_{a,j}$ which contains links each of which has capacity $w_k \geq W_{\min}$. If there is any cut C_b such that $SC_{a,j} \subseteq C_b$, then any subset cut $SC_{b,i}$ generated from C_b will always be a redundant SC.

Proof: From Theorem 1, each $SC_{b,i}$ generated from C_b contains at least all links in $SC_{a,j}$. Thus, by definition, each $SC_{b,i}$ is redundant. □

To take the benefit of the property described in Theorem 3, we suggest all the cuts in $C_{s,t}$ be ordered following the increasing cardinality of their number of links, and we generate the SC starting from the cut with the smallest cardinality. Then, each SC which meets the given criteria in Theorem 3 is kept in a separate list. Should there be a cut that is a superset of any SC in the list, the detected cut can be ignored. Thus, this step reduces unnecessary generations of subsets, which in turn, also reduces the number of (external) redundant SC.

Example: Table I shows all cuts, C_1 through C_{18} (sorted based on their increasing cardinality), of the network in Fig. 2 for the given (s, t) node pair. For $W_{\min} = 4$, the $small\_link_{18}$ for the cut C_{18} is obtained as:{1 2 5 8 14 15}. Because each link in a $small\_link$ is always less than $W_{\min}$, each 1-subset of the small-link is a SSL. Taking all 2-subsets of the $small\_link_{18}$, we obtain $SSL_{18,j}$ as: {2 5}, {2 8}, {2 14}, {2 15}, {5 15}, {8 15}, {14 15}. Reducing redundant SSL, we find that all, but {1}, are redundant 1-link SSL. The algorithm, then, generates all 3-subsets from the $small\_link_{18}$, and finds that none of them is an SSL. Thus, the algorithm stops generating larger subsets, and all the obtained SSL are $MSSL$. Using Lemma 1, we obtain the MSC for the cut C_{18}: $MSC_{18,1} = \{1\,2\,5\,7\,8\,13\,14\,15\} - \{1\} = \{2\,5\,7\,8\,13\,14\,15\}$, $MSC_{18,2} = \{1\,2\,5\,7\,8\,13\,14\,15\} - 2\,5 = \{1\,7\,8\,13\,14\,15\}$, $MSC_{18,3} = \{1\,5\,7\,13\,14\,15\}$, $MSC_{18,4} = \{1\,5\,7\,8\,13\,15\}$, $MSC_{18,5} = \{1\,5\,7\,8\,13\,14\}$, $MSC_{18,6} = \{1\,2\,7\,8\,13\,14\}$, $MSC_{18,7} = \{1\,2\,5\,7\,13\,14\}$, and $MSC_{18,8} = \{1\,2\,5\,7\,8\,13\}$. Similarly, from C_2, we generate $MSC_{2,1} = \{3, 12\}$. Note that $w_3 \geq W_{\min}$, and $w_{12} \geq W_{\min}$; also $MSC_{2,1} \subseteq C_3\ (= \{3, 6, 12\})$, $MSC_{2,1} \subseteq C_5\ (= \{3, 9, 12, 14\})$, and $MSC_{2,1} \subseteq C_6\ (= \{3, 12, 13, 14\})$. Thus, by Theorem 3, we can a priori delete cuts C_3, C_5, and C_6. Should we generate the SC for the deleted cuts, we obtain the following redundant MSC: {3, 6, 12}, {3, 9, 12}, {3, 12, 14}, and {3, 12, 13}. Using this method for the remaining cuts, we generate the MSC of the network:{1 3}, {2 3}, {3 12}, {3 4 6}, {6 11 12}, {6 12 15}, {12 13}, {9 12}, {12 14 15}, $\{1, 3, 4, 9\}^{\mathrm{R}}$, $\{1, 3, 4, 14\}^{\mathrm{R}}$, {3 4 9 14}, {1 7 11}, {1 2 7 15}, {2 7 11 15}, {1 4 6 11}, {1 4 6 15}, {4 6 11 15}, {1 4 9}, {1 4 14 15}, {4 9 14 15}, {7 11 12}, {2 7 12 15}, {3 4 8 13 14}, $\{1, 3, 4, 13, 14\}^{\mathrm{R}}$, $\{1, 3, 4, 8, 13\}^{\mathrm{R}}$, {1 4 8 13}, {1 4 13 14}, {4 8 13 14 15}, {1 2 7 9}, {1 7 9 14}, {1 7 9 15}, {2 7 9 14 15}, {1 2 7 8 13 14}, {1 2 5 7 8 13}, {1 2 5 7 13 14}, {1 5 7 8 13 15}, {1 5 7 8 13 14}, {1 5 7 13 14 15}, {1 7 8 13 14 15}, {2 5 7 8 13 14 15}.

Removing the external redundancies (the ones in the list with a superscript 'R'), we obtain the NMSC for the network. Finally, using CAREL [1], we get the expression for CRU. Assuming that the network has equal link unreliability of 0.1, CRU is given as 0.050 156, and, hence, $\mathrm{CRR} = 1 - 0.050\,156 = 0.949\,844$.

The proposed SCT Algorithm-1 produces the SSL (with total link capacities less than $W_{\min}$) to help enumerating the SC, and this approach is different than the steps taken in [17]. Because SSL are generated from $small_link$s, it is obvious that this approach is more efficient compared to that in [17] because, in general, $\tau_i < L_i$. However, if we generate the subsets exhaustively, we may have to generate up to 2^{τ_i} subsets for each C_i. We observe that the worst case scenario for Algorithm-1 is when it eventually generates only one SSL, i.e., $SSL_{i,j} = small\_link_i$. For this case, Algorithm-l generates all $2^{\tau_i} - 1$ possible subsets, and there are $2^{\tau_i} - 2$ redundant SSL; Algorithm-1 is recommended for use only when τ_i, on average, is small (≤ 10) and $W(small\_link_i) \gg W_{\min}$. Assuming that, in general, each link of a large network has nonunique capacity, we propose SCT Algorithm-2.

C. SCT Algorithm-2

Given a $small\_link_i$, we group all of the links based on their link capacities. Let G_{i,β_k} be a group of links each with a link capacity of β_k units. Thus, for g different link capacities, we obtain a set of groups, G_i, comprising $G_{i,\beta_1}, G_{i,\beta_2}, \ldots, G_{i,\beta_g}$. Let φ_{β_k} denote the number of links in a group G_{i,β_k}, and thus $\sum_k \varphi_{\beta_k} = \tau_i$. For a given $W_{\min}$, there are no more than $(W_{\min} - 1)$ groups in the G_i, and hence $g < W_{\min}$.

Example: Consider the following set of links, each with its corresponding link capacity.

Link	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
Capacity	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6

For $W_{\min} = 8$, we find $small\_link_i = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r\}$. Dividing the links in the $small\_link$ into groups, we obtain the set of groups G_i as $G_{i,1} = \{a, b, c\}$, $G_{i,2} = \{d, e, f\}$, $G_{i,3} = \{g, h, i\}$, $G_{i,4} = \{j, k, l\}$, $G_{i,5} = \{m, n, o\}$, and $G_{i,6} = \{p, q, r\}$.

The proposed SCT Algorithm-2 generates all MSSL of a $small\_link_i$ in two steps:

Step 1: Generate all combinations of sum-of-subsets notations, $SSLN_{i,j} = \{\alpha_1 : \beta_1\} \times \{\alpha_2 : \beta_2\} \times \{\alpha_3 : \beta_3\} \times \ldots \times \{\alpha_w : \beta_w\}$, for $w \leq g$, and $j = 1, 2, \ldots$

Step 2: Generate $MSSL$ from each $SSLN_{i,j}$.

Each $\{\alpha_k : \beta_k\}$ denotes a set of all α_k-subsets of G_{i,β_k}, for $0 \leq \alpha_k \leq \varphi_{\beta_k}$, and $1 \leq k \leq g$. The 'x' is a set multiply operator. For each $SSLN_{i,j}$, the α_k and the β_k are computed such that $\sum_{i=1}^{w} \alpha_i \beta_i < W_{\min}$.

An $SSLN_{i,j}$ represents one or more SSL for $small\_link_i$. This proposed SCT Algorithm-2 utilizes the fact that for large networks, there are some sets of links, each of which has the same link capacities. By grouping the links by link capacity, and using a more concise notation to represent a set of SSL, the number of subsets generated will be less, and hence the number of redundancy tests will also be reduced. The problem to enumerate all $SSLN_{i,j}$ is also similar to the sum of subsets problem [22]. Here, given *distinct* positive numbers β_k, each with φ_{β_k} duplicates, we generate all combinations of the numbers with the requirement that their sum should be less than $W_{\min}$.

To avoid generating redundant SSL, and hence internal redundant SC, we remove any redundant $SSLN_{i,j}$. An $SSLN_{i,j} = \{\alpha_1 : \beta_1\} \times \{\alpha_2 : \beta_2\} \times \{\alpha_3 : \beta_3\} \times \ldots \times \{\alpha_w : \beta_w\}$, is redundant if there is an $SSLN_{i,k} = \{a_1.b_1\} \times \{a_2.b_2\} \times \{a_3.b_3\} \times \ldots \times \{a_w.b_w\}$ such that $\alpha_1 \leq a_1$, $\alpha_2 \leq a_2$, $\alpha_3 \leq a_3$, and $\alpha_w \leq a_w$. It is obvious that using these concise notations, there will be fewer redundancy tests/deletions to produce SC. However, each redundancy test needs more computation time than that needed in Algorithm-1.

Example: Using the set of group G_i, obtained in the previous example, and given $W_{\min} = 8$, we enumerate the fourteen SSLN: $SSLN_{i,1} = \{1:6\} \times \{1:1\}$, $SSLN_{i,2} = \{1:5\} \times \{1:2\}$, $SSLN_{i,3} = \{1:5\} \times \{2:1\}$, $SSLN_{i,4} = \{1:4\} \times \{1:3\}$, $SSLN_{i,5} = \{1:4\} \times \{1:2\} \times \{1:1\}$, $SSLN_{i,6} = \{1:4\} \times \{3:1\}$, $SSLN_{i,7} = \{1:3\} \times \{1:2\} \times \{2:1\}$, $SSLN_{i,8} = \{1:3\} \times \{2:2\}$, $SSLN_{i,9} = \{1:3\} \times \{3:1\}$, $SSLN_{i,10} = \{2:3\} \times \{1:1\}$, $SSLN_{i,11} = \{1:2\} \times \{3:1\}$, $SSLN_{i,12} = \{2:2\} \times \{3:1\}$, $SSLN_{i,13} = \{3:2\} \times \{1:1\}$, and $SSLN_{i,14} = \{3:1\}$. Each SSLN meets the inequality $\sum_{i=1}^{w} \alpha_i \beta_i < W_{\min}$. As an example, for $SSLN_{i,7} = \{1:3\} \times \{1:2\} \times \{2:1\}$, the sum of link capacities of one link in $G_{i,3}$, of one link in $G_{i,2}$, and of any two links in $G_{i,1}$ $(= 2)$, is equal to 7 (less than $W_{\min}$). The $SSLN_{i,11}$ & $SSLN_{i,14}$ are redundant with respect to the $SSLN_{i,12}$, and hence are deleted.

Once we generate all non redundant SSLN, we generate all $MSSL$, which in turn are used to enumerate the $MSC_{i,j}$. The $MSSL$ are generated from an $SSLN_{i,j}$ as follows.

(i) From each $\{\alpha_k : \beta_k\}$, we generate a $G\_sub_{\beta_k}$ which comprises all α_k-subsets of G_{i,β_k}. Let $G\_sub_{\beta_k,j}$ denote a subset in the $G\_sub_{\beta_k}$, for $j = 1, 2, \ldots, \binom{\varphi_{\beta_k}}{\alpha_k}$.

(ii) We obtain all $MSSL_{k,j}$ by taking the set multiplications of each $G\_sub_{\beta_{k,l}}$ in $G\_sub_{\beta_k}$ with each $G\_sub_{\beta_{h,j}}$ in $G\_sub_{\beta_h}$, for all, h, j, k, and l.

Example: Given $SSLN_{i,12} = \{2:2\} \times \{3:1\}$, obtained in the previous example, we generate its $G\_sub_2 = \{\{d, e\}, \{d, f\}, \{e, f\}\}$, and $G\_sub_1 = \{a, b, c\}$. Taking the set multiplications of G_sub_2 with G_sub_1, we obtain three MSSL: {d, e, a, b, c}, {d, f, a, b, c}, and {e, f, a, b, c}. Similarly, from $SSLN_{i,7} = \{1:3\} \times \{1:2\} \times \{2:1\}$, we obtain the following 27 MSSL $\{g, h, i\} \times \{d, e, f\} \times \{(a, b), \{a, c), (b, c)\} = \{g, d, a, b\}$, {g, d, a, c}, {g, d, b, c}, {g, e, a, b}, {g, e, a, c}, {g, e, b, c}, {g, f, a, b}, {g, f, a, c}, {g, f, b, c}, {h, d, a, b}, {h, d, a, c}, {h, d, b, c}, {h, e, a, b}, {h, e, a, c}, {h, e, b, c}, {h, f, a, b}, {h, f, a, c}, {h, f, b, c}, {i, d, a, b}, {i, d, a, c}, {i, d, b, c}, {i, e, a, b}, {i, e, a, c}, {i, e, b, c}, {i, f, a, b}, {i, f, a, c}, and {i, f, b, c}. Note, should we not delete the redundant $SSLN_{i,11} = \{1:2\} \times \{3:1\}$, as an example, the following three redundant SSL are obtained: {d, a, b, c}, {e, a, b, c}, and {f, a, b, c}. Continuing this process on the remaining SSLN, we obtain 120 MSSL.

Even though our greedy approach in Algorithm-1 avoids generating 2^{18} possible subsets from the $small\_link_i$ (in the previous example) to obtain the 120 $MSSL$, the method still needs to generate $\binom{18}{1} + \binom{18}{2} + \binom{18}{3} + \binom{18}{4} + \binom{18}{5} + \binom{18}{6} = 31\,179$ subsets, in addition to removing 31 059 redundant SSL. On the other hand, Algorithm-2 needs to remove only two redundant SSLN; and from the other twelve (nonredundant) SSLN, the method produces 120 MSSL. Algorithm-2, in the example, generates only 70 subsets. Thus, the Algorithm-2 is expected to be more efficient compared to Algorithm-1 for enumerating the SC of large networks.

IV. Implementation, and Time Complexity of Algorithms

1) SCT Algorithm-1: The implementation of the proposed SCT Algorithm-1 (to generate the NMSC set) is shown in Fig. 3.

2) Implementation and Time Complexity: The time complexity of Algorithm-1 depends on the total number of cuts $(= m)$ in the network, the number of links in each cut C_i $(= L_i)$, the number of links in each $small\_link_i$ $(= \tau_i)$, the total number of subsets generated for each C_i $(= S_i)$, the total number of MSC in each C_i $(= M_i)$, and the total number of NMSC in the network $(= A)$. The last four parameters, in turn, heavily depend on the network topology, the assigned link capacities, and the required $W_{\min}$.

The function `is_red_by_theorem3 ( )` of the Algorithm-1 in Fig. 3 is implemented as follows.

```
is_red_by_theorem3 (C_i):
for (all MSC_{j,k} in MSC_i3) do
  if (MSC_{j,k} ⊆ C_i) then
    return (1)
return (0)
```

Referring to Theorem 3, this function is used to delete, a priori, the cuts which will always produce redundant SC. Assuming linear searching, the time complexity of this function is $O(\eta)$,

SCT Algorithm-1:

Input: $C_{s,t}$, link capacities, and W_{min};

Output: *NMSC_set*

```
Sort all cuts Ci in ascending cardinality; // i = 1, ..., m
for (all Ci in Cs,t) do
begin
    if (is_red_by_theorem3(Ci)) then
       delete Ci;
    else
    begin
       generate_MSC_1 (MSCi, Ci);
       generate_NMSC (NMSC_set, MSCi);
    end;
end;
```

Fig. 3. The SCT Algorithm <1.

where η is the total number of MSC in the MSC_t3. Each subset testing can be done in $O(1)$ because each cut C_i, MSC, and NMSC is bit-implementable.

Fig. 4 shows the implementation of function generate_MSC_1 ( ) of the Algorithm-1. The function gen_small_link ( ) in %% of Fig. 4 generates a $small\_link_i$ for a cut C_i. In the following we show the implementation for the function.

```
/* Generate small_linki from a cut Ci */
gen_small_link (Ci, found_MSC):
for (each link k in Ci) do
    if (wk < Wmin) then
      put link k in small_linki;
/* There is no small_link, each wk ≥ Wmin */
if (small_linki == ∅) then
begin
  put Ci in MSCi; // The Ci is an MSC
  found_MSC = TRUE;
end;
```

The function requires $O(L_i)$ time, where L_i is the total number of links in a cut C_i. If each link k has capacity $w_k \geq W_{\min}$, the function makes C_i as the MSC, and returns found_MSC = TRUE. This flag will make function generate_MSC_1 ( ) returns with the obtained MSC (see Fig. 3).

The gen_all_subset (small_link$_i$, j) in %%% of Fig. 4 enumerates all sized j subsets of the small_link$_i$, and therefore its time complexity is $O(\binom{\tau_i}{j})$, where τ_i is the total number of links in small_link$_i$. In %%% of Fig. 4, each of the subset is tested for being an SSL, and if it is, then an SC is obtained. The is_SSL ( ) function requires only $O(\tau_i)$ time.

```
generate_MSC_1 (MSCi, Ci):
/* Step 2.1*/
small_linki = gen_small_link(Ci, found_MSC);
if (found_MSC) then
   return;
/* Step 2.2 */
for (j=1; j ≤ τi; j++) do
begin
    stop_searching = TRUE;
    /* Step 2.2.1 */
    subset_listi,j = gen_all_subset(small_linki, j);
    /* Step 2.2.2 */
    for (each subsetk in subset_listi,j) do
    begin
       if (is_SSL(subsetk) then
       begin
           stop_searching = FALSE;
           SCi,k = Ci - subsetk;
           put_new_SC (SCi,k, MSCi, MSC_t3);
       end;
    end;
    if (stop_searching) then
       break;
end;
```

Fig. 4. Function generate_MSC_1 ( ).

Then, function put_new_SC ( ) in Fig. 4 obtains nonredundant MSC$_i$ or deletes redundant MSC from the existing list. The implementation of the function put_new_SC ( ) is shown as follows.

```
put_new_SC (SCi,k, MSCi, MSC_t3):
new_msc = TRUE;
/* Check for redundancy to produce the
set MSCi */
for (each SCi,j in MSCi) do
begin
  /* The generated SC is redundant */
  if (SCi,k ⊆ SCi,j) then
  begin
    new_msc = FALSE;
    break;
  end;
  else if (SCi,j ⊆ SCi,k) then
     delete SCi,j from MSCi;
```

```
end;
if (new_msc) then
begin
  put SC_{i,k} into MSC_i;
  /* Used for theorem 3; Complexity is
O(L_i) */
  if (each link capacity in the SC_{i,k},
w_l ≥ W_min) then
    put the SC_{i,k} in MSC_t3;
end;
```

As shown in the function, each nonredundant SC is stored in $\mathtt{MSC_i}$, and is also tested for meeting the requirement of Theorem 3. The test requires $O(L_i)$ time, and hence the complexity of this function is $O(M_i + M_i * L_i)$. With $\binom{\tau_i}{j}$ iterations, the time complexity of **%%%** in Fig. 3 is $O\left(\binom{\tau_i}{j} * (M_i + \tau_i) + M_i * L_i\right)$.

The steps **%%$** & **%%%** in general, are used for $j < \tau_i$ (due to its greedy approach). The worst-case time complexity of the **%%** (for $j = 1, 2, \ldots, \tau_i$) is $O\left(2^{\tau_i} * (M_i + \tau_i + 1) + M_i * L_i\right)$. Finally, the worst-case complexity of function generate_MSC_1 ( ) can be computed directly from the sum of the results for **%$** & **%%** which gives $O\left(2^{\tau_i} * (M_i + \tau_i + 1) + M_i * L_i\right)$.

The algorithm for function generate_NMSC ( ) is shown in Fig. 5. There are two different cases for removing external redundant SC. In CASE 1, the new generated MSC is redundant; while in CASE 2, one or more MSC in the NMSC_set are redundant. Removing redundancy in CASE 1 is more efficient compared to that in CASE 2 because once the new generated MSC is detected to be redundant, the process is stopped. One of the advantages of ordering the cuts based on their increasing cardinality is to make the occurrences of redundancies in CASE 1 more likely, as observed in our experimental results (about 90%). From the implementation of generate_NMSC ( ), we compute the worst-case time complexity of the function to be $O(A * M_i)$.

3) Time Complexity of the SCT Algorithm-1: The is_red_by_theorem3 ( ), generate_MSC_1 ( ), and generate_NMSC ( ) functions are called for each cut C_i, and hence the time complexity of the proposed Algorithm-1, on average, is $O\left(m * (2^{\tau} * (M + \tau + 1) + M * (L + A))\right)$. Note that the number of minimal paths, and minimal cuts of a general network with n nodes & b links is $O(2^{b-n+2})$, and $O(2^{n-2})$, respectively. Because $\tau < L$, $L \ll M$, $M \ll A$, and $A \approx m$, and assuming in general $b \approx n \approx L$, the worst case time complexity of the Algorithm-1 can be approximated as $O((2M + L + 1) * m^2) = O(M * m^2)$.

1) SCT Algorithm-2: The only difference between Algorithm-1 & Algorithm-2 is on the implementation of the generate_MSC_1 ( ) function. Thus, in the following, we show only the implementation, and the time complexity of the modified function, generate_MSC_2 ( ), for Algorithm-2.

2) Implementation and Time Complexity: Fig. 6 shows the implementation of function generate_MSC_2 ( ). Function gen_group_$\mathtt{G_i}$ ( ) in **%$** of the figure generates a set of groups $G_i = \{G_{i,\beta_k}\}$ from a cut $\mathtt{C_i}$, for $k \in \{1, 2, 3, \ldots\}$, where each β_k is a unique link capacity. The function requires

```
generate_NMSC (NMSC_set, MSC_i):

for (all MSC_{i,j} in MSC_i) do

begin

    is_NMSC = TRUE;

    for (each NMSC_k in NMSC_set) do

    begin

        /* CASE 1: MSC_{i,j} is an external redundant MSC*/

        if (NMSC_k ⊆ MSC_{i,j}) then

        begin

            is_NMSC = FALSE;

            break;

        end;

        /*CASE 2: NMSC_k is external redundant MSC */

        if (MSC_{i,j} ⊆ NMSC_k) then

           remove NMSC_k from the NMSC_set;

    end;


    if is_NMSC then

        put MSC_{i,j} into NMSC_set;

end;
```

Fig. 5. Function generate_NMSC ( ).

$O(L_i)$ time, where L_i is the number of links in a cut C_i. If each link l has capacity $w_l \geq W_{\min}$, the function makes C_i as the MSC, and returns found_MSC = TRUE. This flag will make function generate_MSC_2 ( ) returns with the obtained MSC (see Fig. 6). In the following we show the implementation of the function.

```
gen_group_G_i (C_i, found_MSC):
for (each link l in C_i) do
  if ((w_l < W_min) and (w_l = β_k)) then
    put link l in G_{i,β_k};
/* There is no link l with w_l < W_min */
if (G_i = ∅) then
begin
  /* The C_i is an MSC */
  put C_i in MSC_i;
  found_MSC = TRUE;
end;
```

The generate_SSLN ($\mathtt{G_i}, \mathtt{W_{min}}$), in **%%** of Fig. 6, enumerates all possible nonredundant $\mathtt{SSLN_{i,j}} = \{\alpha_1 : \beta_1\} \times \{\alpha_2 : \beta_2\} \times \{\alpha_3 : \beta_3\} \times \ldots$ from all G_{i,β_k} in G_i. The best scenario

```
generate_MSC_2 (MSC_i, C_i):
/* Step 2.1 */
G_i = gen_group_ G_i (C_i, found_MSC);
if (found_MSC) then
   return;
/* Step 2.2 */
SSLN_i = generate_SSLN(G_i, W_min);
/* Step 2.3 */
generate_MSSL();
for (each SSLN_{i,k} in SSLN_i) do
begin
    /* Step 2.3.1 */
    for (each {α_j:β_j} in SSLN_{i,k}) do
        G_sub_{β_j} = gen_all_subset(G_{i,β_j}, α_j);
    /* Step 2.3.2 */
    MSSL_i = G_sub_{β_{k,l}} x G_sub_{β_{h,j}} x ... ;
    put_MSC(MSSL_i);
end;
```

Fig. 6. Function generate_MSC_2 ( ).

for the function is when each link in G_i has the same link capacity w_k, for any k (i.e., there is only one group in G_i). For this case, the complexity of the function is $O(\binom{\tau_i}{j})$, where $j \leq \tau_i$ is an integer such that each j-combination of the links qualifies as a SSL, i.e., $(j * w_k) < W_{\min}$. On the other hand, the worst-case scenario for the function is when each link has a different link capacity. For this case, there are τ_i groups in G_i, and therefore, 2^{τ_i} SSLN are generated with each SSLN representing only one SSL. In this worst case, Algorithm-2's complexity is the same as that of Algorithm-1. Thus, the complexity of this worst case is $O(2^{\tau_i})$, and the average complexity of the function is $O((\binom{\tau_i}{j} + 2^{\tau_i})/2)$.

%& of Fig. 6 obtains MSSL from $SSLN_{i,k}$, for $k = 1, 2, \ldots \mu_i$. The gen_all_subset $(G_{i,\beta_j}, \alpha_j)$ in **%&!$** of the figure generates all sized α_j subsets of the links in G_{i,β_j}. Let φ_{β_k} be the number of links in group G_{i,β_j}. The complexity of the function is thus $O(\binom{\varphi_{\beta_k}}{\alpha_j})$.

In **%&!%** of Fig. 6, we start with the set multiplications of the $G\_sub_{\beta_{k,l}}$ to generate all $MSSL$, and hence the MSSL for the C_i. The complexity of this function depends on the number of MSSL generated. Thus, the complexity of this step, on average, should be $O(M_i/\mu_i)$. Function put_MSC ( ) in Fig. 6 obtains all the MSC, and put them in a list. In the function, each generated MSC is tested for meeting the requirement of Theorem 3. The test requires $O(L_i)$ time, and hence the complexity

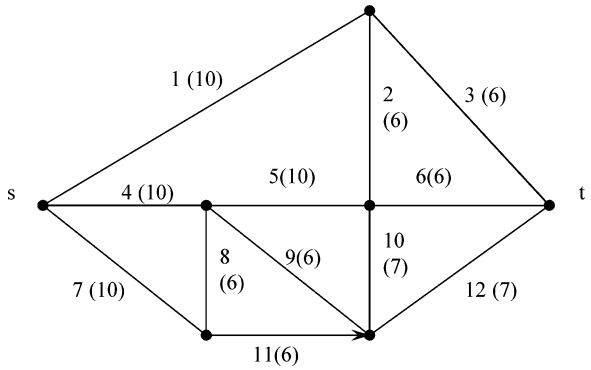


Fig. 7. 7 node, 12 link network.

of the function is $O(M_i * L_i)$. The implementation of function put_MSC ( ) is shown as follows.

```
put_MSC (MSSL_i):
for (each MSSL_{i,k} in MSSL_i) do
begin
  MSC_{i,k} = C_i − MSSL_{i,k};
  put MSC_{i,k} into MSC_i;
  /* Used for theorem 3; Complexity:
O(L_i) */
  if (each link capacity in the MSC_k,
w_l ≥ W_min) then
    put MSC_{i,k} to MSC_t3;
end;
```

Because **%&!$** & **%&!%** are iterated μ_i times, the complexity of **%&** is $O(\mu_i * \binom{\varphi_{\beta_k}}{\alpha_j} + M_i + \mu_i * M_i * L_i)$. Summing up the complexities of **%!$**, **%!%**, and **%&** we obtain the time complexity of the generate_MSC_2 ( ) function as $O(L_i + (\binom{\tau_i}{j} + 2^{\tau_i})/2 + \mu_i * \binom{\varphi_{\beta_k}}{\alpha_j} + M_i + \mu_i * M_i * L_i)$ $= O((\binom{\tau_i}{j} + 2^{\tau_i})/2 + \mu_i * \binom{\varphi_{\beta_k}}{\alpha_j} + \mu_i * M_i * L_i)$. In general, $\binom{\varphi_{\beta_k}}{\alpha_j} \ll M_i * L_i$, and hence the complexity expression can be reduced to $O((\binom{\tau_i}{j} + 2^{\tau_i})/2 + \mu_i * M_i * L_i)$. Comparing this time complexity for Algorithm-2 with that for Algorithm-1, we may conclude that for a large τ, the complexity of the function generate_MSC_2 ( ) used in Algorithm-2 is better than that of function generate_MSC_1 ( ) used in Algorithm-1. Therefore, the Algorithm-2 is suitable for use in large networks with large τ.

V. Experimental Results and Discussion

The proposed SCT Algorithm-1 & Algorithm-2 are simple, and have been implemented in C on a Pentium III computer. Both Algorithm-1 & Algorithm-2 are used to generate the NMSC for the networks shown in Figs. 2 & 7 for various values of $W_{\min} \leq W_{\max}$. The two methods generate exactly the same sets of NMSC for the network. Then, assuming an equal link reliability of 0.9 for all links, and utilizing CAREL [1] on the NMSC, we compute the CRU, and hence the CRR of the networks. The resulting CRR for the networks match the

TABLE II
RESULTS FOR THE NETWORK IN FIG. 2

W_{min}	NMSC	MSC	SC	IRED	ERED	subset	T3
1	18	18	18	0	0	0	0
2	16	16	16	0	0	20	6
3	21	21	29	8	0	53	6
4	37	42	63	21	5	151	3
5	32	46	94	48	14	299	7
6	24	56	135	79	32	332	7
7	18	58	174	116	40	359	7
8	17	41	134	93	24	221	9
9	14	64	198	134	50	305	3
10	11	88	365	277	77	688	0

TABLE III
RESULTS FOR THE NETWORK IN FIG. 7

W_{min}	NMSC	MSC	SC	IRED	ERED	subset	T3
1 to 6	20	20	20	0	0	0	0
7	41	56	56	0	15	117	0
8 to10	38	71	71	0	33	179	0
11, 12	31	102	102	0	71	326	0
13	38	111	165	54	73	598	0
14	21	139	207	68	118	598	0
15, 16	21	141	211	70	120	602	0
17	23	190	287	97	167	602	0
18	24	210	309	99	186	602	0
19	26	192	345	153	166	767	0

TABLE IV
RESULTS FOR THE NETWORK IN FIG. 8

W_{min}	NMSC	MSC	SC	IRED	ERED	subset	T3	CRR
1 to 3	214	214	214	0	0	0	0	0.987390
4	230	230	230	0	0	213	22	0.976531
5	218	275	275	0	57	388	55	0.967208
6	294	405	405	0	111	731	35	0.948631
7	298	428	496	68	130	1085	40	0.947716
8	287	504	689	185	217	1842	40	0.945882
9	190	611	927	316	421	2986	50	0.775060
10	246	968	1429	461	722	5971	15	0.767096
11	171	1065	1710	645	894	7880	15	0.690289
12	143	1183	2052	869	1040	10060	15	0.687399
13	173	1465	2679	1214	1292	17354	11	0.682157

results reported in [15], [16]. These results, empirically, show the correctness of our methods.

In Tables II & III, we show the total number of subset cuts (SC), minimal-SC (MSC), and network-minimal-SC (NMSC) generated using Algorithm-1 for the networks in Figs. 2 & 7, respectively. When $W_{\min} = 1$, the total number of NMSC is equal to the number of cuts in the network. As shown in the tables, the total number of NMSC for each $W_{\min}$ is of the same order as the number of cuts of the networks. The NMSC are generated by removing all the external-redundant SC (ERED, from all cuts) from the MSC, which in turn are generated by removing all the internal-redundant SC (IRED, within a cut) from the SC. For each $small\_link$ of a cut, we generate its subsets, and each subset is checked for being an SC. The tables show the total number of subsets generated to produce all the SC. The total number of IRED, ERED, and subsets generated determine the computational complexity of the method. The results in the tables, empirically, show that the complexity of the algorithm is polynomial in the number of cuts. Columns T3 in Tables II & III show the number of cuts that can be removed a priori following Theorem 3. However, as shown in Table III, such cuts may not exist at all.

Table IV shows the CRR and the other parameters for the network in Fig. 8, with various $W_{\min}$ values. Notice that each of

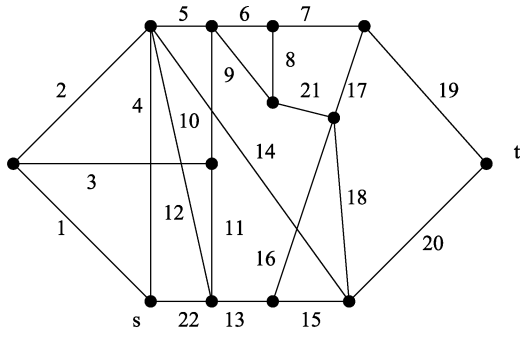


Fig. 8. 13 node, 22 link network.

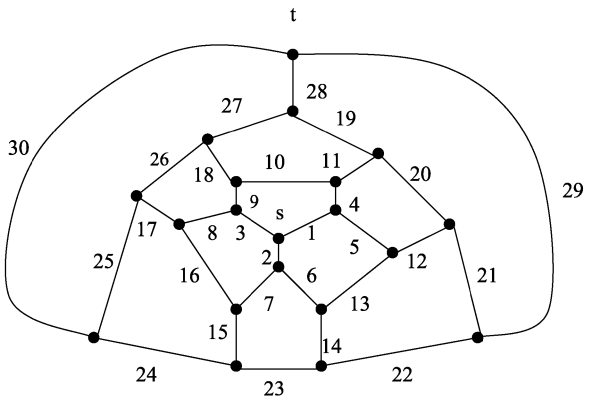
TABLE V(a)
RESULTS FOR THE NETWORK IN FIG. 9 USING ALGORITHM-1

W_{min}	NMSC	MSC	SC	IRED	ERED	subset	T3
1<6	7376	7376	7376	0	0	0	0
7	7644	10413	10413	0	2769	16925	2055
8	8855	25397	25397	0	16542	68978	1032
9	7067	48906	48906	0	41839	198032	113
10<12	4962	70374	70374	0	65412	378642	0
13	4675	68062	77499	9437	63387	864170	0
14	5278	77052	103045	25993	71774	1114879	0
15	4794	104056	150553	46497	99262	1185575	0
16	2184	159705	226982	67277	157521	1197592	0
17	1199	226301	296654	70293	225162	1198806	0
18	782	283016	353390	70374	282234	1198816	0
19	624	305636	380477	74841	305012	1513832	0
20	647	297916	395063	97147	297269	2075477	0
21	773	295201	435765	140564	294428	2466397	0
22	479	314580	526904	212324	314101	2599331	0

TABLE V(b)
RESULTS FOR THE NETWORK IN FIG. 9 USING ALGORITHM-2

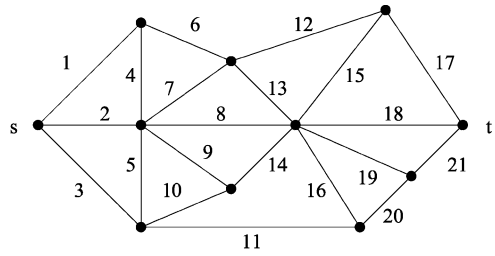
W_{min}	NMSC	MSC	SSLN	IREDN	ERED	subset	CRR
1<6	7376	7376	7376	0	0	0	0.997120
7	7644	10413	5104	0	2769	10196	0.983694
8	8855	25397	11687	0	16542	25395	0.960965
9	7067	48906	20341	0	41839	48906	0.948304
10<12	4962	70374	27840	0	65412	70374	0.928967
13	4675	68062	27840	0	63387	68062	0.915370
14	5278	77052	27840	2375	71774	76769	0.834765
15	4794	104056	32499	2476	99262	100833	0.809854
16	2184	159705	38421	2657	157521	145250	0.645474
17	1199	226361	50708	2660	226162	203261	0.533820
18	782	283016	57735	2661	282234	244015	0.481695
19	624	305626	64024	2661	305012	266635	0.475291
20	647	297916	64024	5159	297269	261933	0.465750
21	773	295201	64024	7192	294428	260788	0.451991
22	479	314580	66574	10029	314101	263664	0.367090

the number of subsets generated is also in polynomial order of the number of cuts of the network. This shows, empirically, the efficiency of the proposed Algorithm-1. In Table V(a) & V(b), we show the results generated from 7376 cuts of the network in Fig. 9 for various values of $W_{\min}$. We used Algorithm-1 to produce Table V(a), and Algorithm-2 to generate Table V(b). As shown in Table V(a), even though the total number of subsets, and the internal redundant SC, are still in polynomial order of the number of cuts of the network, the numbers are significantly increasing. In Table V(b), we show the number of subsets generated by the SCT Algorithm-2. Comparing the results with those in Table V(a), the total number of subsets generated has been reduced significantly. The SSLN column in the table shows the number of sum-of-subsets notations generated by Algorithm-2,



Links:	1	2	...																										30
Cap:	6	7	9	8	8	6	7	8	9	6	8	7	6	9	8	7	6	8	9	7	9	7	8	9	8	9	9	8	9	7

Fig. 9. 20 node, 30 link network.



Links:	1	2	...	21
Cap:	50	30	15	50	15	25	45	15	10	10	20	90	10	60	10	20	10	80	10	10	10

Fig. 10. 11 node, 21 link network.

and the IREDN is the number of redundant SSLN. The nonredundant SSLN, then, are translated to obtain the MSSL, which, in turn, are used to generate the MSC. Because we use bit notations for SC & MSC, only two *set-operations* (in Algorithm-1) are needed to determine if an $SC_{i,j}$ $(MSC_{a,b})$ is redundant with respect to an $SC_{i,k}$ $(MSC_{p,q})$. On the other hand, determining an SSLN redundant (with respect to another SSLN) in Algorithm-2 is more time consuming. However, for larger networks, the total number of SSLN generated in Algorithm-2 would be far less than the total number of SC in Algorithm-1. Furthermore, once we generate a set of nonredundant SSLN, we can directly generate non-(internal) redundant SC. Therefore, Algorithm-2 is more suitable for use to generate the NMSC for large networks ($small_link$s of size larger than 10). The average $small\_link$ of the network in Fig. 9 is 9.5. However, Algorithm-2 does not reduce the number of external redundant SC (see Column ERED in Tables V(a) and V(b)). It is obvious that an improved algorithm is needed to further reduce the complexity of generating the NMSC.

We used Algorithm-1 to compute the CRR of the 5-node 7-link ARPA network used in [7], [17]. The network contains six cuts, and for $W_{\min} = 3$ the average $small\_link$ is 1.75; and thus Algorithm-1 is more suitable to be used than Algorithm-2. Our method generates exactly the same result as obtained in [17]. For $W_{\min} = 3$, the network contains only one 1-link *valid cut* [17], and therefore the method in [17] has to generate 31 subsets to obtain five *valid cuts* of the network. On the other hand, Algorithm-1 generates only 7 subsets, and removes two external-redundant SC, and hence our method is more efficient than the method in [17]. Furthermore, because the method in [17] uses a 2-dimension cutset-matrix to represent the cuts of a network, the method [17] requires large memory spaces to solve the CRR of large networks. The algorithms in [15] & [16] generate a set of composite paths (cp) from the pathset information of a network. The two algorithms also require a max-flow-min-cut algorithm [21] to compute the capacity flow for each generated cp [15], [16] to generate a set of minimal success composite paths. In general, the algorithms [15], [16] generate $O(2^{m_f})$ cp (m_f is the total number of failure simple paths of the network), and require all cuts information of the network to compute the capacity flow of each generated cp [15]. Therefore, the pathset-based algorithms are not suitable for computing the CRR of large networks with hundreds or thousands of paths & cuts. Neither [15] nor [16] provides experimental results for solving the CRR of large networks, and hence we are unable to compare the efficiency of our algorithms with those in [15], [16].

TABLE VI
PERFORMANCE COMPARISONS WITH METHOD IN [19]

W_{min}	NMSC	#Disjoint products		CRR	CPU time (milliseconds)			
		SCT	JY		SCT			JY
					Algorithm-1	CAREL	Total	
10	94	1004	9860	0.997673	< 1	10	10	8123
20	159	964	10487	0.968098	< 1	20	20	10216
30	204	791	3759	0.885689	10	30	40	5251
40	233	1054	3049	0.863109	10	50	60	5086
50	163	375	1233	0.737366	20	20	40	1988
60	101	221	492	0.629520	10	10	20	802
70	49	75	190	0.401041	20	< 1	20	247
80	17	17	31	0.183449	20	10	30	27
85	17	17	*	0.166772	70	< 1	70	*

* Not reported

To compare the performance of our approach with that recently reported in [19], we used Algorithm-1 to compute the CRR, for various $W_{\min}$, of the network shown in Fig. 10 (Case 1 in [19]; average $small\_link = 3.25$). Table VI shows, for each $W_{\min}$, the number of NMSC produced, the number of disjoint products generated (using CAREL [1] from each of the NMSC set), and the running time of our SCT method. The table also provides the required CPU time on a Pentium III, and the number of disjoint products generated using the Jane & Yuan's (JY) technique as reported in [19]. We first generated the cutset of the network, which required 110 milliseconds of CPU time. From the cutset, we then used Algorithm-1 to generate the NMSC for various values of $W_{\min}$. Finally, we used CAREL to obtain the CRR for each set of NMSC, assuming each link has a reliability of 0.9. The CPU time required for our SCT approach, as shown in Table VI, includes only the CPU time needed to generate the NMSC using Algorithm-1, and the time for running CAREL. The cutset enumeration was done only once (for the network), and hence we have excluded its CPU time from our calculation. As shown in Table VI, our method outperforms the JY [19] in both the required CPU time, and the number of disjoint products generated. Based on the results reported in [19], our method is also better than Rueger's technique [13].

From Table V(b) we also notice that the total number of NMSC generated are decreasing with increasing $W_{\min}$ values

(from 7376 to 476). We observe that the time required to generate the CRR of the network, when the number of NMSC is far less than the number of cuts of the network, is less than that needed to compute the network's terminal reliability. As an example, CAREL needed 156.27 seconds to obtain the terminal reliability of the network (from 7376 cuts), while the Algorithm-2 & CAREL took only 37 seconds to compute the CRR for the network with $W_{\min} = 22$. Generating NMSC takes polynomial time in the number of cuts, while generating CRR (terminal reliability) from NMSC (cuts) using CAREL is exponential in the number of NMSC (cuts). Thus, when the total number of NMSC is far less than the number of cuts, computing CRR is more efficient compared to obtaining the terminal reliability of the network.

Tables IV and V(b) show the CRR for the networks shown in Figs. 8, and 9, respectively, for various values of $W_{\min}$, and assuming an equal link reliability of 0.9 for all links. From the CRR values, we observe that there are tradeoffs between allowing a larger bandwidth requirement in the networks (increasing the $W_{\min}$), and the reliability of the network to meet such a requirement. However, for some values of $W_{\min}$, we get good tradeoffs. As an example, increasing the $W_{\min}$ from 10 to 12 units for the network in Fig. 9 does not reduce the reliability of the network. Similarly, increasing the $W_{\min}$ from 3 to 8 units (167% increase) reduces the reliability of the network in Fig. 8 only by 3%. On the other hand, increasing $W_{\min}$ by only 1 unit (from 8 to 9) for the network in Fig. 8 reduces the reliability of the network appreciably (by 18%). Our method can be used to determine the optimal $W_{\min}$ for certain network configurations, and hence can be used to compare the reliability of various candidate topologies having heterogeneous link-capacities. We are exploring to extend our method so that it can also be used to find the bottleneck of the network, and add minimal extra bandwidth as necessary to improve the network's CRR.

ACKNOWLEDGMENT

The authors are grateful to the anonymous referees for their valuable suggestions and comments.

REFERENCES

[1] S. Soh and S. Rai, "CAREL: Computer Aided Reliability evaluator for distributed computer networks," *IEEE Trans. Parallel and Distributed Syst.*, vol. 2, no. 2, pp. 199–213, Apr. 1991.
[2] T. Luo and K. S. Trivedi, "An improved algorithm for coherent-system reliability," *IEEE Trans. Reliab.*, vol. 47, no. 1, pp. 73–78, Mar. 1998.
[3] S.-Y. Kuo, S.-K. Lu, and F.-M. Yeh, "Determining terminal-pair reliability based on edge expansion diagrams using OBDD," *IEEE Trans. Reliab.*, vol. 48, no. 3, pp. 234–246, Sep. 1999.
[4] C. J. Colbourn, *The Combinatorics of Network Reliability*. New York: Oxford University Press, 1987.
[5] S. Rai and D. P. Agrawal, *Distributed Computing Network Reliability*: IEEE Computer Society Press, 1990.
[6] S. Rai and Y. C. Oh, "Tighter bounds on full access probability in fault-tolerant multistage interconnection networks," *IEEE Trans. Parallel and Distributed Syst.*, vol. 10, pp. 327–335, Mar. 1999.
[7] S. H. Lee, "Reliability evaluation of a flow network," *IEEE Trans. Reliab.*, vol. R-29, no. 1, pp. 24–26, Apr. 1980.
[8] K. B. Misra and P. Prasad, "Comments on: reliability evaluation of flow networks," *IEEE Trans. Reliab.*, vol. R-31, pp. 174–176, Jun. 1982.
[9] K. K. Aggarwal, Y. C. Chopra, and J. S. Bajwa, "Capacity consideration in reliability analysis of communication systems," *IEEE Trans. Reliab.*, vol. R-31, pp. 177–181, Jun. 1982.
[10] K. K. Aggarwal, "Integration of reliability and capacity in performance measure of a telecommunication network," *IEEE Trans. Reliab.*, vol. R-34, no. 2, pp. 184–186, Jun. 1985.
[11] ——, "A fast algorithm for the performance index of a telecommunication network," *IEEE Trans. Reliab.*, vol. R-37, pp. 65–69, Apr. 1988.
[12] P. K. Varshney, A. R. Joshi, and P.-L. Chang, "Reliability modeling and performance evaluation of variable link-capacity networks," *IEEE Trans. Reliab.*, vol. 43, no. 3, pp. 378–382, Sep. 1994.
[13] W. J. Rueger, "Reliability analysis of networks with capacity constraints and failures at branches and nodes," *IEEE Trans. Reliab.*, vol. R-35, pp. 523–528, Dec. 1986.
[14] S. Rai, A. Kumar, and E. V. Prasad, "Computing the performance index of a computer network," *Reliab. Eng.*, vol. 16, pp. 153–161, 1986.
[15] S. Rai and S. Soh, "A computer approach for reliability analysis of large telecommunication-network with heterogeneous link-capacities," *IEEE Trans. Reliab.*, vol. 40, no. 4, pp. 441–451, Oct. 1991.
[16] S. M. Lee and D. H. Park, "An efficient method for evaluating network reliability with variable link-capacities," *IEEE Trans. Reliab.*, vol. 50, no. 4, pp. 374–379, Dec 2001.
[17] K. K. Aggarwal, Y. C. Chopra, and J. S. Bajwa, "Modification of cutsets for reliability evaluation of communication systems," *Microelectron. Reliab.*, vol. 22, no. 3, pp. 337–340, 1982.
[18] S. Soh and S. Rai, "A cutset approach to survivability evaluation of large telecommunication networks with heterogeneous link-capacities," in *Proc. Int. Symp. Systems and Circuits*, Jun. 1991, pp. 896–899.
[19] C.-C. Jane and J. Yuan, "A sum of disjoint products algorithm for reliability evaluation of flow networks," *Eur. J. Oper. Res.*, vol. 131, pp. 664–675, 2001.
[20] R. Schanzer, "Comments on: reliability modeling and performance of variable link-capacity networks," *IEEE Trans. Reliab.*, vol. 44, no. 4, pp. 620–621, Dec. 1995.
[21] K. Kyandoghere, "A note on: reliability modeling and performance of variable link-capacity networks," *IEEE Trans. Reliab.*, vol. 47, no. 1, pp. 44–45, Dec. 1998.
[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed: MIT Press, 2001.
[23] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms, Pseudocode*: Computer Science Press, 1998.
[24] S. Liu, K.-H. Cheng, and X. Liu, "Network reliability with node failures," *Networks*, vol. 35, no. 2, pp. 109–117, 2000.
[25] D.-R. Liang, R.-H. Jan, and S. K. Tripathi, "Reliability analysis of replicated and-or graph," *Networks*, vol. 29, pp. 195–203, 1997.
[26] A. Kumar, R. Rastogi, and A. Silberschatz, "Algorithms for provisioning virtual private networks in the hose model," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 565–578, Aug. 2002.

received a B.S. degree in electrical engineering from University of Wisconsin Madison, and M.S. and Ph.D. in electrical engineering from Louisiana State University, Baton Rouge. He was a faculty member (1993–2000), and the director of the Research Institute (1998–2000) at Tarumanagara University-Indonesia. He is currently a Lecturer with the department of Computing at Curtin University of Technology, Perth, W.A., Australia. Dr. Soh has published several papers in some refereed international journals, and conference proceedings in the area of computer network, network reliability, and parallel and distributed processing. He is a member of the IEEE.

is a Professor with the Department of Electrical and Computer Engineering at Louisiana State University, Baton Rouge, Louisiana. Dr. Rai has taught and researched in the area of network traffic engineering, ATM, reliability engineering, fault diagnosis, neural netbased logic testing, and parallel and distributed processing. He is a co-author of the book Wave Shaping and Digital Circuit; and tutorial texts Distributed Computing Network Reliability, and Advance in Distributed System Reliability. He has guest edited a special issue of IEEE TRANSACTIONS ON RELIABILITY on the topic Reliability of Parallel and Distributed Computing Network. He was an Associate Editor for IEEE TRANSACTIONS ON RELIABILITY from 1990 to 2004. Dr. Rai has published about 100 technical papers in the refereed journals and conference proceedings. He received the best paper award at the 1998 IEEE International Performance, Computing, & Communication Conference (Feb. 16-18, Tempe, Arizona; paper title: S. Rai and Y. C. Oh, Analyzing packetized voice and video traffic in an ATM multiplexer). Dr. Rai is a senior member of the IEEE, and a member of the ACM.