

An Enhanced Approach To Network Reliability
Using Boolean Algebra

by Alexandru Octavian Balan

An Honors Thesis presented to the Departments of Computer Science
and Mathematics of Lafayette College on May 16, 2003

Thesis advisors:

Prof. Lorenzo Traldi – Mathematics Department

Prof. Chun Wai Liew – Computer Science Department

Abstract

The network reliability analysis problem consists of evaluating a measure of the reliability of a network. A more general context than the network reliability analysis is the stochastic coherent binary system (*SCBS*) analysis. One approach used in determining the reliability of a system involves Boolean algebra, and in particular a sum of disjoint products (*SDP*) technique. This paper introduces generalized sum of disjoint products (*GSDP*), a generalized version of *SDP*, and evaluates the benefits of such a technique. A *GSDP*-based algorithm is presented and a comparison is provided between an existing *SDP* technique and *GSDP*.

1. Introduction

In today's world computers are very rarely stand-alone. They are often connected to networks that can vary in size from small to gigantic. Local Area Networks (LAN) connect a relatively small number of network devices over relatively short distances. In contrast, Wide Area Networks (WAN) span large physical distances and typically include extremely large numbers of network devices. The Internet is an example of a WAN.

Initially high reliability systems were required in cases where failure of such systems could have caused massive damage or loss of human life. Examples include aircraft systems, nuclear reactor control systems, and defense command and control systems. However, it has been recognized that very high reliability systems make economic sense in a wide range of industries such as telecommunications, banking and credit verification systems [3, 18]. In addition, the need for reliable inter-computer communication has increased dramatically lately with the introduction of Distributed Computer Systems (*DCS*) [20].

One way the performance of such a network can be measured is by determining the reliability parameter, which is the probability that the network operates. The reliability parameter can be used as part of network design procedures. If the design of a network does not yield a satisfactory reliability value, then the design needs to be adjusted by either changing the topology of the network, or by adding, removing or replacing unreliable network components. When a new design is generated, the reliability of the network is

recomputed to determine if it is satisfactory. Several such iterations may be needed before a satisfactory network design is achieved [3].

For certain methods, the evaluation of the network reliability parameter can be divided into two stages. In the first stage a reliability formula is obtained based on the topology of the network, while in the second stage a numerical value is computed from the reliability formula. Existing techniques for determining the reliability formula require quite a bit of computational time, while the opposite is true for the second stage. These methods are preferable in cases where the network design is complete, but the reliability parameter changes and needs to be re-evaluated multiple times. It is therefore desirable to obtain effective reliability formulas that would allow quick evaluation of the reliability parameter of the network.

2. Layout of the Paper

The motivation for the present paper is presented in section 3 and concentrates on the reliability of network systems. Section 4 describes the more general problem that is considered in this paper, namely the analysis of a stochastic coherent binary system (SCBS). Existing approaches towards system reliability are included in section 5, followed by a presentation of the sum of disjoint products (SDP) technique in section 6, and by a literature review on this specific technique in section 7. Section 7 also introduces the generalized sum of disjoint products (GSDP) method, a completely new approach that generalizes SDP. Section 8 shows how a numerical value for the reliability of a system can be obtained from a GSDP reliability expression. A short overview of the BT-03 algorithm that uses GSDP to analyze an SCBS system appears in section 9, followed in section 10 by a detailed presentation of the GSDP transformations and reductions used by BT-03. Experimental results and a discussion of the results can be found in sections 11 and 12, ideas for suggested future studies are presented in section 13, and the conclusion appears in section 14.

3. Motivation

There is an inherent difficulty in modeling the failure of a network. This is because the causes of failures are frequently random and they may vary over time. If however the causes of failures are well understood, then historical data can be used to estimate the rates of component failures [3]. Discrete stochastic models¹ are typically used in network reliability analyses. The network devices and the connections between them form the components of the network. In general, both the devices and the connections are prone to failure. The components of a stochastic binary network can take only one of two states: operative or failed. The state of a network at a given time is given by the states of all its components at that time. The component probability of operation can be interpreted as the component's reliability [3]. The state of a component is a probabilistic random event. In order to simplify the problem, this event is assumed to be independent of the states of the other components in the network.

To summarize, the following two assumptions have been made in the literature:

1. Each component of the network is either operational or failing (a two-state component).
2. Component states are mutually statistically independent, which means the operation of one component is not affected by another [3, 11, 21, 24].

A network system can be described by a network graph $G = (V, E)$, where V is a set of nodes or vertices, and E is a set of undirected edges and / or directed arcs. The vertices of the graph correspond to network devices, while the arcs and edges correspond to the connections between them. Each component, vertex or arc / edge, has an associated probability of operating p_v or p_e . The vector of component reliabilities in a network is denoted by \mathbf{p} .

¹ A discrete stochastic model is a mathematical model consisting of probabilistic random variables that have countable range.

Example 1

Suppose the problem is to determine the reliability of sending a message from a computer terminal s to a computer terminal t through a network that has the topology shown in Figure 1. The network system consists of three routers, R_1 , R_2 , and R_3 , the two terminals, s and t , and the existing connections between them.

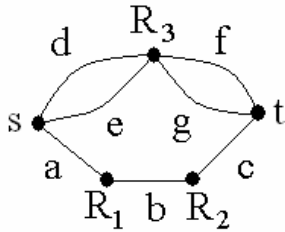


Figure 1

The graph $G = (V, E)$ that corresponds to the network system just described has $V = \{s, t, R_1, R_2, R_3\}$ and $E = \{a, b, c, d, e, f, g\}$, and has the associated reliability vector $\mathbf{p} = (p_x)_{x \in E \cup V}$.

The network reliability analysis problem consists of evaluating a measure of the reliability of a network. The inputs to the problem are the network and a failure probability for each component of the network. The type of reliability that appears most extensively in practice is the *k-terminal measure* with its two special cases, the *two-terminal measure* and the *all-terminal measure*. The two-terminal measure is the probability that there is an operating path between two particular nodes (also called terminals) of an undirected network. The problem in Example 1 is a 2-terminal reliability problem. The all-terminal measure is the probability that an undirected network is connected. The k-terminal measure is the probability that a specified node subset K of an undirected network is connected [3]. Another type of reliability is the *source-to-K-terminal measure*, which is defined for a directed network as the probability that a message can be sent from a particular source node to any of the nodes of some specified node subset K .

In general, the underlying network of the reliability problem can be directed, undirected, or both. In certain examples, some edges and / or nodes are perfectly reliable, and the corresponding probabilities are all 1, while the rest have an associated probability of operation. These differences bring about difficulties in finding a unified solution to any type of network reliability problem. Some network transformations have been used to eliminate these differences, but they do not cover the entire spectrum of networks. They are described below with the sole purpose of revealing their limitations, but they pose no relevance to the material that follows them.

Transformation of Undirected into Directed Graphs

If the specifications of a reliability problem refer to an undirected graph, the problem cannot be easily converted to use a directed graph. A natural method to consider involves replacing each undirected edge by two anti-parallel directed arcs, as shown in Figure 2. However the states of the two new arcs are not statistically independent because the failure in one direction of the originally undirected connection typically implies the failure of the connection in the opposite direction as well. Therefore the first assumption about mutually statistically independent component states would be violated.

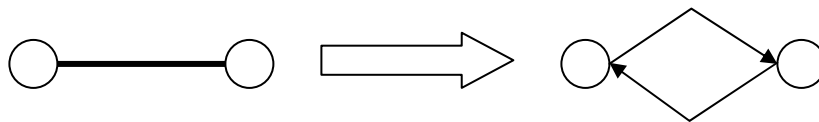


Figure 2

Transformation of Failing Nodes into Non-failing Nodes

For a graph that has failing nodes, an equivalent reliability graph with perfect nodes can be constructed using only unreliable arcs. This can be done for each node separately. For a given node v with a probability p_v of operating, two nodes are created instead, v_i and v_o , as shown in Figure 3. All the inbound arcs into node v are redirected towards v_i , while all the outbound arcs from v will now originate from v_o . A new arc is added from v_i to v_o , with an associated probability of operating equal to p_v . This is a reliability-preserving

transformation [1, 2, 3]. This transformation is however restrictive since it is applicable for directed graphs only. Eliminating unreliable nodes is not possible for undirected graphs.

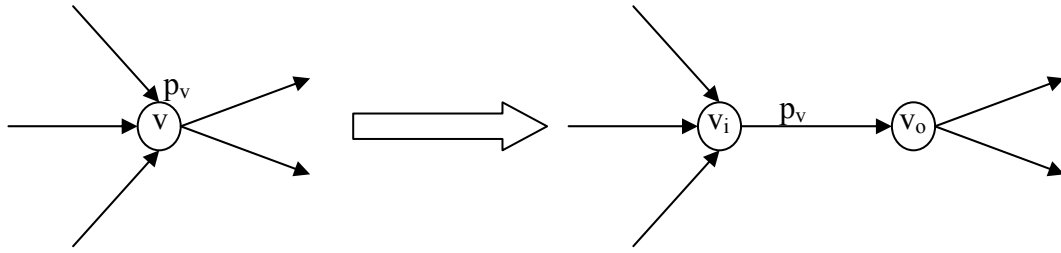


Figure 3

Transformation of Failing Edges or Arcs into Non-failing Edges or Arcs

Given a graph with unreliable edges or arcs, it is possible to construct an equivalent reliability graph for which all edges and arcs are perfect. For an edge or arc e that has a probability p_e of operating, a node v can be added in such a way that e is split into two perfectly reliable edges or arcs, respectively; for arcs, the directions are maintained to provide the same functionality as the original arcs. The node v is assigned a probability of operating equal to p_e , as illustrated in Figure 4.

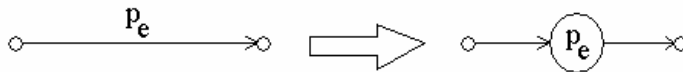


Figure 4

4. Problem Description

A more general framework that provides a unified treatment of the concepts in all of the network reliability problems involves stochastic coherent binary systems (SCBS). An SCBS can be constructed for any network reliability problem.

A *stochastic binary system (SBS)* is a system $\Sigma = (T, \psi)$, where T is the component set, and ψ is the structure function of the system. This system is said to fail randomly as a function ψ of the random failure of its components in T . At any instant of time, each component e in the component set T assumes randomly and independently one of two

states, *operating* or *failing*, with probabilities p_e and $q_e = 1 - p_e$, respectively. The structure function is defined for each $S \subseteq T$ by

$$\psi(S) = \begin{cases} 1 & \text{if, when S operates and T - S fails, the system operates} \\ 0 & \text{if, when S operates and T - S fails, the system fails.} \end{cases} \quad (1)$$

An *SBS* is *coherent* if $\psi(\emptyset) = 0$, $\psi(T) = 1$ and $\psi(S') \leq \psi(S)$ for any $S' \subset S$. The third condition guarantees that the failure of any component can only have a detrimental effect on the operation of the system. The reliability of Σ is the probability that Σ is operating:

$$\text{Rel}(\Sigma, \mathbf{p}) = P[\psi(W) = 1, \text{ where } W \text{ is a random set}^2 \text{ of operative components}], \quad (2)$$

where $P[\cdot]$ is the probability function and $\mathbf{p} = (p_e)_{e \in T}$. For any *SCBS*, a *pathset* is defined as a set of components whose operation implies system operation, and a *minpath* is a minimal pathset; similarly, a *cutset* is defined as a set of components whose failure implies system failure, and a *mincut* is a minimal cutset. The minpaths and / or the mincuts of Σ are useful in computing the reliability of the system [3, 8, 18].

Example 2

In the problem described in Example 1, the operation of the system at a given time translates into the existence of a path of operating components between terminals s and t . For simplicity, it can be assumed that s and t are perfect terminals, with probability of operating equal to 1. The set $\{a, c, d, e, f, R_1, R_3\}$ is a pathset since it contains two paths between s and t , (d, R_3, f) and (e, R_3, f) , that allow for a message to get from s to t . There are five minpaths for this system: $\{a, R_1, b, R_2, c\}$, $\{d, R_3, f\}$, $\{d, R_3, g\}$, $\{e, R_3, f\}$ and $\{e, R_3, g\}$.

The set $\{a, b, d, f, g, R_2\}$ is a cutset because there is no path of operating components from s to t which involves only c , e , R_1 and / or R_3 . The mincuts corresponding to this cutset are $\{a, f, g\}$, $\{b, f, g\}$ and $\{R_2, f, g\}$. There are fifteen mincuts for this graph.

² A random set is a probabilistic random variable having sets of elements as values.

Any network reliability problem can be reduced to an SCBS problem by listing the minpaths of the network problem. For example, an appropriate model for perfect-node two-terminal reliability is the SCBS $\Sigma = (E, \psi)$, where E is the set of edges of the network and $\psi(S) = 1$ if and only if S contains the edges of a path from the source node s to the terminal node t . Note that S is a pathset, while the edges of a path from s to t form a minpath. In general, for a perfect-node k -terminal reliability problem let the SCBS system be $\Sigma = (E, \psi)$, where E is still the set of edges of the network and $\psi(S) = 1$ if and only if S contains the edges of a spanning-tree of K (= minimal set of edges that would make nodes in K connected). Note that S is a pathset, while the edges of the spanning-tree form a minpath [8].

5. Existing Approaches

There are many ways to evaluate the network reliability measures, including exact algorithms, analytic bounds and Monte Carlo simulations. Exact algorithms are always precise, but often impractical due to their complexity. (Essentially all reliability problems of interest are #P-Complete³). Analytic bounds exist for certain problem classes, and for these classes, the accuracy of the bounds and the running time of the associated algorithm are factors to consider. The well-studied Monte Carlo method of simulating the stochastic behavior of a system can be applied to reliability problems. It does so by examining a small fraction of the states, chosen randomly, and constructing a point estimate of the reliability measure, along with confidence intervals for the measure. It can produce inaccurate estimates [3]. The present paper considers only exact algorithms.

Efficient exact algorithms exist only for restricted classes of networks. Most of these algorithms use reliability-preserving reductions. The most basic reductions are: the

³ If a class of recognition problems (problems that search for only one solution) is NP or NP-Complete, the corresponding class of counting problems (problems that search for all solutions) is #P or #P-Complete, respectively [3]. It is also possible for easier than NP or NP-Complete recognition problems to give rise to #P or #P-Complete counting problems. For instance, the problem of determining whether an SCBS $\Sigma = (T, \psi)$ is operational under any circumstances has a trivial solution; if the system is not operational when all its components are operational ($\psi(T) = 0$), then the system is never operational. However counting the number of different, not-necessary minimal, operational states is a #P-Complete problem, even when all the minpaths are available [25].

elimination of irrelevant edges, the contraction of mandatory edges, and the series and parallel reductions. Such reductions can be applied very efficiently. Most network reliability problems cannot be completely solved using only these reliability-preserving reductions. However, these reductions can prove valuable in dealing with the more general reliability problems, as they can simplify the problem and they are not difficult to implement.

If the reliability-preserving reductions fail to reduce a given network to a restricted class of networks for which an efficient exact algorithm exists, a potentially exponential time method has to be used. Several exact algorithms for network reliability are proposed in the literature. They can be classified as: state enumeration, factoring or decomposition, inclusion-exclusion, and sum of disjoint products.

The state enumeration method is very simple, but not very efficient. As the name might imply, all states of the network are generated so as to find all operational states. Once the operational states are available, the reliability formula can be computed easily. This can be generalized for an SCBS system $\Sigma = (T, \psi)$. The reliability formula is given by

$$\text{Rel}(\Sigma) = \sum_{\psi(S)=1} \left(\left(\prod_{e \in S} p_e \right) \cdot \left(\prod_{e \notin S} (1 - p_e) \right) \right). \quad (3)$$

One way to deal with the inefficiency of generating all states is to group operating states together. The *factoring* or *pivotal decomposition* method makes use of the Factoring Theorem, which concentrates on the state of an individual component e :

$$\text{Rel}(\Sigma) = p_e \cdot \text{Rel}(\Sigma|e \text{ is operative}) + (1 - p_e) \cdot \text{Rel}(\Sigma|e \text{ is failed}) [17, 18]. \quad (4)$$

Factoring can be exploited to reduce the size of a system to be analyzed by defining component contraction and deletion, which are to be used to represent the two cases where a component operates or fails. An undirected edge e in a graph G can be contracted from G by identifying its end points, removing e , but retaining all multiple arcs or loops that arise, giving a multi-graph $G \cdot e$. If an undirected edge e has failed, then it can be deleted from G , thus obtaining a new graph $G - e$. A similar approach can be employed for an

SCBS $\Sigma = (T, \psi)$. A component e can be contracted by defining $\psi_{\Sigma \cdot e}(T)$ to be 1 whenever $\psi_{\Sigma}(T \cup \{e\}) = 1$. A failed component e can be deleted by defining $\psi_{\Sigma - e}(T)$ to be 1 whenever $\psi_{\Sigma}(T) = 1$. Using this notation, the Factoring Theorem becomes

$$\text{Rel}(\Sigma) = p_e \text{Rel}(\Sigma \cdot e) + (1 - p_e) \text{Rel}(\Sigma - e) \quad [3, 7]. \quad (5)$$

For this method not to degenerate into complete state enumeration, certain reductions should be carried out during the recursive steps. For example, even though no reliability-preserving reductions can be performed on Σ , it is possible that $\Sigma \cdot e$ or $\Sigma - e$ can be simplified using these types of reductions.

The state enumeration and factoring / decomposition techniques are more easily employed when the graph representation of the network is available, and they are used most efficiently when the size of the system is small or can be reduced considerably using all possible reductions. Once all possible reductions have been applied, these methods are usually very impractical, unless the size of the system has been reduced considerably. On the other hand, the inclusion-exclusion and the sum of disjoint products methods take advantage of the enumeration of the minpaths or mincuts of the system. Given a network $G = (V, E)$, a stochastic coherent binary system $\Sigma = (T, \psi)$ can be constructed for it, together with an arbitrary ordered set of all minpaths $\{P_i\} (P_1, P_2, \dots, P_h)$, where h represents the number of minpaths. To reiterate, a minpath P_i is a minimal set of components whose functionality is sufficient for communication through the network. A probabilistic event EP_i can be associated with the minpath P_i , stating that all components in minpath P_i are operating. It is easy to see that the reliability measure is just the probability that at least one of the events EP_i occurs:

$$\text{Rel}(\Sigma) = \text{P}[EP_1 \vee EP_2 \vee \dots \vee EP_h], \quad (6)$$

also called the pathset probability.

The notation for sets of components can be simplified by juxtaposing the elements of a set. As an example, the set $\{a, c, d, e, f\}$ can be written $acdef$. Minpaths and mincuts follow the same notation.

Example 3

The 2-terminal reliability problem corresponding to Figure 5 has two minpaths: $P_1 = ab$ and $P_2 = cd$. The probabilistic events associated with the minpaths are $EP_1 = a \wedge b$ and $EP_2 = c \wedge d$.

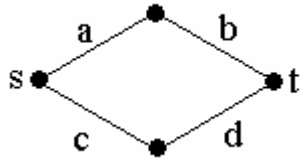


Figure 5

In a similar fashion, an arbitrary ordered set of all m mincuts C_i (C_1, C_2, \dots, C_m) can also be used, where the mincuts describe the failure rather than the operation of the system. The probabilistic event EC_i denotes that all components in mincut C_i have failed.

Example 4

The mincuts for the system in Figure 5 are $C_1 = ac$, $C_2 = ad$, $C_3 = bc$ and $C_4 = bd$. The probabilistic events associated with the mincuts are $EC_1 = \bar{a} \wedge \bar{c}$, $EC_2 = \bar{a} \wedge \bar{d}$, $EC_3 = \bar{b} \wedge \bar{c}$ and $EC_4 = \bar{b} \wedge \bar{d}$. Notice that mincuts have a similar form to minpaths, except that individual variables are negated. In the literature it is customary to drop the negations from the variables with the understanding that the negations are implicit or the probabilities of the variables have been adjusted. For a variable x , p_x becomes $1 - p_x$.

The reliability measure is just the probability that none of the events EC_i occurs:

$$\text{Rel}(\Sigma) = \text{P}[\overline{EC}_1 \wedge \overline{EC}_2 \wedge \cdots \wedge \overline{EC}_m] = 1 - \text{P}[EC_1 \vee EC_2 \vee \cdots \vee EC_m], \quad (7)$$

where \overline{EC}_i denotes the complement of the event EC_i . Since the reliability of the system is just one minus the cutset probability, the theory behind using the mincuts is similar to the one used for minpaths. The only relevant distinction is the cardinality of the sets of minpaths and mincuts. In some cases, there are fewer mincuts than minpaths, and thus performing the reliability analysis on the set of mincuts can improve the performance of the algorithm [3]. From now on the discussion will only refer to the minpaths and their associated probabilistic events, with the understanding that mincuts can be used instead as well.

It is important to notice though that the EP_i events are not disjoint events, and thus the system reliability is not just the sum of the probabilities of EP_i . The following result can be generalized using Poincare's principle of inclusion-exclusion:

$$\text{P}[EP_1 \vee EP_2] = \text{P}[EP_1] + \text{P}[EP_2] - \text{P}[EP_1 \wedge EP_2]. \quad (8)$$

Example 5

In the problem described in

The notation for sets of components can be simplified by juxtaposing the elements of a set. As an example, the set $\{a, c, d, e, f\}$ can be written $acdef$. Minpaths and mincuts follow the same notation.

Example 3, the reliability measure is given by $\text{P}[EP_1 \vee EP_2] = \text{P}[(a \wedge b) \vee (c \wedge d)] = \text{P}[a \wedge b] + \text{P}[c \wedge d] - \text{P}[a \wedge b \wedge c \wedge d]$.

In general, the reliability of a system can be obtained through the standard inclusion-exclusion expansion

$$\text{Rel}(\Sigma) = \sum_{j=1}^h (-1)^{j+1} \sum_{I \subseteq \{1, \dots, h\}, |I|=j} \text{P}[E_I], \quad (9)$$

where E_I is the event that all minpaths P_i with $i \in I$ are operational [3]. More needs to be done in order to make this approach more efficient than state enumeration, however it is beyond the scope of this paper to present the necessary details.

6. Sum of Disjoint Products Technique

One approach to computing the reliability of a network is to use Boolean algebra, and in particular an *SDP* (sum of disjoint products) technique. This technique is explained below in much detail since it represents the basis of a new technique, which will be introduced later in the paper.

A set of minpaths $\{P_i\}$ corresponding to all the minimal operating states is given for an SCBS $\Sigma = (T, \psi)$. The probabilistic event that all components in minpath P_i are operating is denoted by EP_i , while its complement is denoted by $\overline{EP_i}$. When the EP_i events are not disjoint, the idea of this type of algorithm is to make these events disjoint and then add the corresponding probabilities to obtain the reliability of the entire system.

A very simple method of making the events EP_i disjoint is the following. Let $D_1 = EP_1$, and in general, $D_i = \overline{EP_1} \wedge \overline{EP_2} \wedge \dots \wedge \overline{EP_{i-1}} \wedge EP_i$. Since the events D_i are disjoint, the reliability measure becomes

$$\text{Rel}(\Sigma) = P[EP_1 \vee EP_2 \vee \dots \vee EP_h] = P[D_1 \vee D_2 \vee \dots \vee D_h] = \sum_{i=1}^h P[D_i]. \quad (10)$$

Therefore, a formula for $P[D_i]$ in terms of the operating probabilities of the network components needs to be established. Because of the non-symmetrical pattern in which the EP_i contribute to the D_i , the order in which the minpaths are listed can and does have an impact on the performance of any algorithm that is used.

Fundamental differences among existing SDP techniques stem from the observation that a simple Boolean expression needs to be obtained for D_i so that $P[D_i]$ can be directly computed from an arithmetic expression involving only probabilities of component states and their complements. A Boolean expression for an event E is denoted by $RF(E)$, the

reliability formula of E . Each component of the network gets a corresponding Boolean variable x_a . A Boolean product of variables is a conjunction of single un-complemented variables, x_a (indicating that component a is operating), and / or single complemented variables, \bar{x}_a (indicating that component a has failed). The Boolean expression for EP_i , denoted by $RF(EP_i)$, is very simple; it is just the Boolean product (also called minproduct) of the variables corresponding to minpath P_i . More complex Boolean expressions consist of disjunctions of multiple Boolean terms. A Boolean term can include any type of Boolean operator. If the reliability formula for a Boolean expression E contains only mutually disjoint Boolean terms, then it is considered a disjoint reliability formula and it is denoted by $DRF(E)$.

As a shorthand notation for Boolean expressions, juxtaposition denotes conjunction of variables. Furthermore, the negation operator has the highest precedence, followed by the conjunction operator, and then by the disjunction operator. Another notational device is the use of the addition sign for a disjunction of Boolean products / terms (hence the name *Sum of Disjoint Products*). This helps differentiate the disjunction of Boolean terms from the disjunction within a Boolean term. As an example, the Boolean expression $\bar{x}_1 \vee (x_1 \wedge \overline{x_2 \vee (x_3 \wedge x_4)})$ can be expressed using two Boolean terms $\bar{x}_1 + x_1 \overline{x_2 \vee x_3 x_4}$. Reliability algorithms that produce disjoint products are sometimes called Boolean algebra methods. These techniques differ primarily in the logical structure of the output.

7. Literature Review on Sum of Disjoint Products Techniques

Currently there are only two fundamentally different paradigms for making products disjoint. The first paradigm is based on single variable inversion (*SVI*), and it was introduced by Fratta and Montanari [9]. The Boolean sum of the products corresponding to the minpaths is transformed into a sum of disjoint products of variables, from which the reliability is directly obtained. To be more precise, each D_i is simplified to a sum of disjoint *SVI Boolean products*. Algorithms that use SVI have been proposed by: Fratta et al. [9], Abraham [1], Beichelt and Spross [5, 6], and Locks and Wilson [14, 15, 26]. It is worth mentioning that Abraham's algorithm [1] and a network example that he introduced

have subsequently been used as a basis of comparison between algorithms. One reason for this is that the Abraham algorithm requires shorter computational time [5, 14] than the other aforementioned algorithms.

Example 6

The beginning of the minpath list could be $P_1 = abcx$, $P_2 = acdy$, and $P_3 = xyz$. The goal is to put $D_3 = \overline{EP_1} \wedge \overline{EP_2} \wedge EP_3$ into a sum of disjoint products form. The way to think about it is that the $x \wedge y \wedge z$ event has to occur while the $a \wedge b \wedge c \wedge x$ and $a \wedge c \wedge d \wedge y$ events should not occur. To make EP_3 disjoint from EP_1 , at least one of the a , b , or c variables should be negated. Since it is not known if the component corresponding to variable a is failed or operating, both cases are considered. In the event that a is failed, EP_3 becomes disjoint from both EP_1 and EP_2 . Therefore, one disjoint product is $\overline{a}xyz$. If a is not failed, then c may fail. Having c failed implies again that EP_3 is disjoint from both EP_1 and EP_2 , thus obtaining another disjoint product, $\overline{a}cxyz$. Last but not least, if both a and c are operating, then b has to fail to make EP_3 disjoint from EP_1 , and d has to fail to make EP_3 disjoint from EP_2 , giving rise to the disjoint product $\overline{a}cb\overline{d}xyz$. This results in the Boolean expression $\overline{a}bcx \wedge \overline{a}cdy \wedge xyz$ being expressed as a sum of three disjoint Boolean terms $\overline{a}xyz + \overline{a}cxyz + \overline{a}cb\overline{d}xyz$.

The second kind of logical algorithm uses multiple-variable inversion (*MVI*); the technique was introduced by Grnarov et al. [10], though Heidtmann [11] was the first to use MVI to improve on the Abraham algorithm. Rather than generating disjoint products of variables, the reliability formula is a sum of disjoint *MVI Boolean terms*. An MVI Boolean term is a logical conjunction of single variables and complemented groups of variables⁴ (meaning at least one of the components in each group has failed); a variable cannot appear more than once in the term. Several SVI Boolean products can be combined into one MVI Boolean

⁴ A complemented group of variables is equivalent to a logical disjunction of single complemented variables.

term. This can be demonstrated by the following example: $\overline{x_1 x_2 x_3} = \overline{x_1} x_3 + \overline{x_2} x_3$. This Boolean expression represents at least three components, having components 1 or 2 in the fail state and component 3 in the operative state, while the rest of the variables are not mentioned. This new method considerably reduces the number of terms in the resulting reliability formula using a more compact product expression. Some of the most important algorithms employing the MVI approach are the following: KDH-88 [11], GKG-VT [24] and its subsequent improvement I_VT [16], and CAREL [20].

Example 7

Considering Example 6, note that EP_3 can be made disjoint from both EP_1 and EP_2 if at least one of a or c is negated. The MVI Boolean term that expresses this is $\overline{ac}xyz$. The other possible case is when both a and c are operating, and it is treated the same as in Example 6. Hence, in the MVI framework, the $\overline{abcx} \wedge \overline{acdy} \wedge xyz$ Boolean expression can be expressed using two disjoint Boolean terms $\overline{ac}xyz + ac\overline{b} \overline{d}xyz$, as opposed to three in the SVI framework.

Much of the subsequent work has concentrated on reducing the number of terms in the resulting reliability formula and making them disjoint efficiently. Most authors noted that the order of the minpaths has a huge impact on the size of the reliability formula regardless of the type of SDP algorithm used. Consequently, much work has been done to assess simple strategies for reordering the minpaths, also called *preprocessing*. A typical heuristic used is to sort the minpaths increasingly according to the number of components in a minpath. While this heuristic might work well in some cases, it does not always produce a minimal reliability formula. Moreover, it has not been sufficiently recognized that the type of preprocessing needed depends heavily on the algorithm being used and its internal mechanism for making the products disjoint [23].

The purpose of this paper is to introduce and analyze a completely new technique for making terms disjoint called, *generalized sum of disjoint products (GSDP)*. A GSDP-based algorithm, called BT-03, is also introduced in this paper. This technique generates disjoint

GSDP Boolean terms, which are Boolean expressions using any kind of Boolean operations. A GSDP Boolean term is constrained to have no variable appearing more than once; this constraint guarantees that a simple formula will describe the probability of the event corresponding to a term. The goal of this generalization is to reduce the size of the resulting reliability formula by combining multiple MVI terms into one GSDP term. One problem with the fact that a GSDP term can have a variable appear at most once is that it puts a limit on how many terms can be grouped together.

Example 8

Extending Example 7, $\overline{EP_1} \wedge \overline{EP_2} \wedge EP_3$ can be expressed in words as follows: x , y , and z have to operate, and one of a or c should fail, or otherwise b and d have to fail together. The *GSDP* framework allows this statement to be expressed using only one Boolean term, $(\overline{ac} \vee \overline{bd})xyz$.

To analyze the performance of SDP algorithms, two measures are used: computational efficiency and the resulting number of disjoint products. The algorithms have to be efficient enough to run in a practical amount of time, considering the fact that these problems are #P-Complete [3, 11, 21, 24]. On the other hand, a smaller number of resulting terms facilitates numerical evaluation and reduces computation and rounding errors [11].

8. Computing the Network Reliability Parameter from the *GSDP* Reliability Expression

The Sum of Disjoint Products technique converts the probabilistic events EP_i corresponding to the minpaths P_i into disjoint events D_i , and then sums the probabilities of the events D_i . The Boolean expression for D_i needs to be simplified so that $P[D_i]$ can be directly computed. All current SDP techniques, including GSDP, construct D_i in a manner that satisfies the following conditions:

1. The Boolean expression for D_i is a sum of Boolean terms that involves only variables and their complements.
2. No variable appears more than once in a Boolean term.
3. It is assumed that the state of any component is statistically independent from the state of any other component of the network.

These conditions allow the corresponding probability of each disjoint term, and hence the reliability parameter, to be easily computed. The only rules needed are:

- Negation Rule: $P[\bar{x}] = 1 - P[x]$;
- Conjunction Rule: $P[x \wedge y] = P[x] \cdot P[y]$ (as x and y are statistically independent);
- Disjunction Rule: $P[x \vee y] = P[x] + P[y] - P[x \wedge y] = P[x] + P[y] - P[x] \cdot P[y]$ (as x and y are statistically independent).

Example 9

The goal is to compute the probability of the Boolean *GSDP* term $D = \overline{a \wedge b} \vee (\bar{c} \wedge d)$. In order to evaluate $P[D]$, the substitutions $x = \overline{a \wedge b}$ and $y = \bar{c} \wedge d$ can be made. In that case, $P[x] = P[\overline{a \wedge b}] = 1 - P[a \wedge b] = 1 - p_a \cdot p_b$, while $P[y] = P[\bar{c} \wedge d] = P[\bar{c}] \cdot p_d = (1 - p_c) \cdot p_d$. Hence, $P[D] = P[x \vee y] = P[x] + P[y] - P[x] \cdot P[y] = 1 - p_a \cdot p_b + (1 - p_c) \cdot p_d - (1 - p_a \cdot p_b) \cdot (1 - p_c) \cdot p_d$.

If the *GSDP* term contains any other Boolean operations, those operations should first be converted to an equivalent form using only NOT($\bar{\quad}$), AND(\wedge) and OR(\vee) operators.

9. Generalized Sum of Disjoint Products and the BT-03 Algorithm

The Generalized Sum of Disjoint Products (*GSDP*) paradigm attempts to combine several *SVI* products and *MVI* terms into one *GSDP* term, thus reducing the size of the reliability formula generated. The rest of the paper introduces one way to achieve such a goal through the *BT-03* algorithm. It does so by performing a number of reliability-preserving transformations and reductions, and then using recursion to solve the reduced problem. As

stated in the Existing Approaches section, other algorithms have used this idea of reducing the problem using reliability-preserving reductions, but only a very limited number of classes of reliability problems are solved completely using just these reductions. Most of the existing approaches assume that these reductions have already been performed on the initial input to the problem, and therefore do not refer to them anymore. Since the GSDP algorithm used in the paper is recursive, these reductions prove themselves to be an essential part of the mechanism.

Algorithm Overview

The disjoint reliability formula is generated recursively. Within the recursive step, attempts are made to reduce the problem to smaller ones using the transformations and reductions introduced in the following section, solve each of the smaller problems separately using recursion, and then combine the results to form the disjoint reliability formula for the bigger problem.

Data Structure

Throughout the algorithm, Boolean products are combined using Boolean operations thus forming nested Boolean expressions. Examples of such expressions are:

- $(a \wedge b \wedge c) \vee ((d \vee e) \wedge (f \vee g))$
- $a \wedge d \wedge h \wedge (\bar{b} \vee (\bar{e} \wedge \overline{f \wedge i}))$.

Consequently, a tree structure is necessary to store such Boolean expressions in a computer. Internal nodes are Boolean operators, while leaf nodes are variables. A node corresponding to a unary operator has only one descendant, whereas a non-unary operator node is allowed to have two or more descendants. Even though GSDP allows any type of Boolean operators, the BT-03 algorithm uses only the unary operator NOT($\bar{\quad}$), and the non-unary operators AND(\wedge) and OR(\vee). The data structure has been made more efficient by allowing n-ary conjunctions and disjunctions, as shown in Example 10.

Unary and binary operations between Boolean expressions that do not share any variables can be easily performed using this data structure. Furthermore, using a recursive procedure

on a Boolean expression stored in this manner, it is easy to determine for a given system configuration the Boolean value of the expression.

Example 10

The tree structure for the Boolean expression $a \wedge d \wedge h \wedge (\bar{b} \vee (\bar{e} \wedge \overline{f \wedge i}))$ is shown in Figure 6.

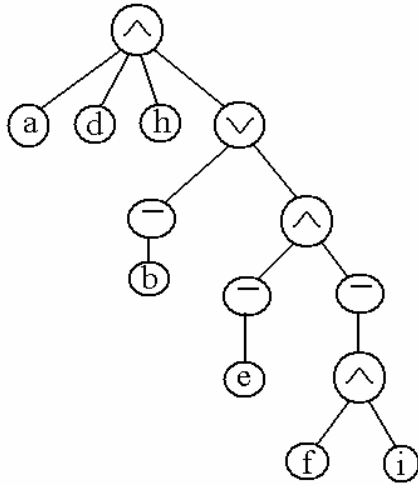


Figure 6

Preprocessing

At various stages of the algorithm it is worthwhile to preprocess the minproducts. Compelling reasons exist especially for SDP algorithms to order the minproducts in increasing order according to the number of variables in the minproducts [23]. Preprocessing influences the number of disjoint terms in the final reliability formula because of the non-symmetric structure of the event $D_i = \overline{EP_1} \wedge \overline{EP_2} \wedge \dots \wedge \overline{EP_{i-1}} \wedge EP_i$. However the preprocessing method depends heavily on the algorithm being used, and finding an optimal one for any given SDP algorithm is a challenging task, if not impossible. This paper does not focus on improvements due to preprocessing, and therefore the most popular method is used instead, which is to list minproducts in order of increasing size.

The BT-03 Algorithm

Given a list of minproducts, the `Process_Pathset()`, `Process_Parallel_Free_Pathset()`, and `Process_Canonical_Pathset()` subroutines return a list of disjoint GSDP terms of the reliability formula. An additional subroutine, `Process_Cutset()`, produces the same outcome, but its input is not a list of minproducts, but a list of mincut products with implicit negations.

The main steps of the algorithm are:

- `Process_Pathset()` subroutine tests for parallel subsystems reductions;
- `Process_Parallel_Free_Pathset()` subroutine tests for series subsystems reductions on a system with no parallel subsystems;
- `Process_Canonical_Pathset()` subroutine performs standard SDP reductions together with component-level series reductions on a system that has no parallel or series subsystems. For this step, the minproducts should be preprocessed.

The main recursive call is to `Process_Pathset()` with the list of all minproducts. The algorithm attempts a sequence of reductions and continues using direct and indirect recursion⁵.

10. GSDP Transformations and Reductions in the BT-03 Algorithm

Transformation of Minpaths into Mincuts and Vice-versa

In order to solve the reliability problem, the inclusion-exclusion and the sum of disjoint products methods typically use either an enumeration of the minpaths or an enumeration of the mincuts as input. According to [3], since the cardinality of the mincut set is sometimes less than that of the minpath set, some algorithms can improve their performance by using the list of mincuts as input. Some algorithms actually look at the cardinality of both minpath and mincut sets and use the set with the lowest cardinality. With the aim of

⁵ Direct recursion refers to a subroutine that contains an explicit call to itself. Indirect recursion (also referred to as mutual recursion) refers to a set of subroutines that contain calls to each other. The simplest case of indirect recursion is when subroutine A contains a call to subroutine B, and subroutine B contains a call to subroutine A.

determining the pathset reliability formula, the BT-03 algorithm needs to determine both the pathset and cutset reliability formulas of several subsystems. In order to do that, a method is needed to construct the set of mincuts from the set of minpaths and vice-versa. This process is called *inverting*.

A system is operating when at least one minpath's components are all operating, so the system reliability formula inputted to the BT03 algorithm is $RF(\Sigma) = EP_1 \vee EP_2 \vee \dots \vee EP_h$. A system is also operating when none of the mincuts succeed in failing the system, or when at least one component is operating in each mincut. With this interpretation, the reliability of the system is $RF(\Sigma) = \overline{EC_1} \wedge \overline{EC_2} \wedge \dots \wedge \overline{EC_m}$. By negating the pathset reliability formula of the system, the cutset failure formula of the system is obtained, in which mincut variables have implicit negations:

$$\overline{EP_1 \vee EP_2 \vee \dots \vee EP_h} = \overline{RF(\Sigma)} = \overline{\overline{EC_1} \wedge \overline{EC_2} \wedge \dots \wedge \overline{EC_m}} = EC_1 \vee EC_2 \vee \dots \vee EC_m.$$

Changing the notation into Boolean products notation, the identity becomes

$$\overline{EP_1 + EP_2 + \dots + EP_h} = EC_1 + EC_2 + \dots + EC_m. \quad (11)$$

Negating both sides of (11), its converse is obtained:

$$EP_1 + EP_2 + \dots + EP_h = \overline{EC_1 + EC_2 + \dots + EC_m}. \quad (12)$$

A very simple and straightforward technique for inverting has been introduced by Locks [12, 13]. It has been made more efficient by Shier and Whited by extending the number of shortcuts taken by the algorithm [19]. This technique is the same regardless of which set is being inverted, minpaths or mincuts. Two fundamental operations are used: inversion and absorption. The inversion operation uses de Morgan's theorems for conjunctions and disjunctions:

- $\overline{x_1 x_2 \dots x_j} = \overline{x_1} + \overline{x_2} + \dots + \overline{x_j} \quad (13)$

- $\overline{x_1 + x_2 + \dots + x_j} = \overline{x_1} \overline{x_2} \dots \overline{x_j}. \quad (14)$

The absorption operation is a minimizing operation, in that it eliminates the Boolean terms that are not minimal:

- $a + ab = a$. (15)

The BT-03 algorithm uses the `Convert_minpaths_to/from_mincuts(Products)` subroutine for inverting minproducts and it follows the steps presented in [19]. The mincut variables have implicit negations, and therefore minproducts have the same format as the mincut products. The `Convert_minpaths_to/from_mincuts()` subroutine is used in `Process_Cutset()` subroutine, which constructs the disjoint pathset reliability formula from the mincuts. The `Process_Cutset()` subroutine does three things: converts the mincut products to minproducts, makes the minproducts disjoint using `Process_Pathset()` subroutine, and complements all variables in the resulting formula because of the implicit negations of the mincut variables (see Appendix A).

Example 11

The network in Figure 7 has three minpaths: ab , ade and cd . The non-disjoint reliability formula is $ab + ade + cd$. The inverse of the reliability formula using de Morgan's theorems is $\overline{ab + ade + cd} = (\bar{a} + \bar{b})(\bar{a} + \bar{d} + \bar{e})(\bar{c} + \bar{d})$. Expanding this formula using the distributive law and absorbing non-minimal terms results in $\bar{a}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{d} + \bar{b}\bar{e}\bar{c}$. Thus the mincuts are: ac , ad , bd and bce . In the mincuts notation, the variables' negations are implicit. Inverting the mincuts would yield back the minproducts.

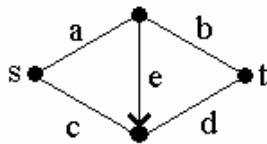


Figure 7

Parallel Reductions

In an undirected graph, the simplest example of a parallel reduction appears in the case of two edges e_1 and e_2 joining the same two nodes. In this instance the two edges are

mutually redundant and therefore they will never appear together in the same minpath. Moreover, they are substitutes of each other and therefore if a minpath contains e_1 , then there has to exist another minpath with the same components, but having e_2 instead of e_1 , and vice-versa. In this case, the network can be reduced by replacing the two edges e_1 and e_2 with an edge e that has the probability of operating equal to $p = 1 - (1 - p_1) \cdot (1 - p_2)$, where p_1 and p_2 are the probabilities of operation of e_1 and e_2 , respectively [3]. This comes from the fact that either one or the other edge can be used:

$$\mathbf{P}[e_1 \vee e_2] = \mathbf{P}[\overline{e_1 \vee e_2}] = \mathbf{P}[\overline{e_1} \wedge \overline{e_2}] = 1 - \mathbf{P}[\overline{e_1}] \cdot \mathbf{P}[\overline{e_2}] = 1 - (1 - p_1) \cdot (1 - p_2). \quad (16)$$

A generalization of this idea can be applied to SCBS systems in a slightly different way. Rather than work at the component level, parallel subsystems are considered. Two subsystems of an SCBS system are in parallel if the subsystems are substitutes for each other. The BT-03 algorithm detects a very simple form of minproduct parallelism using the `Parallel_Division()` subroutine (see Appendix A), in which the operation of one subsystem is independent of the operation of the other subsystem. This also entails that the operation of any of the subsystems implies the operation of the entire system. The idea is that if none of the products in a subset of all minproducts EP_i contain any variables that appear in the remaining minproducts, the two subsystems are connected in parallel.

Example 12

The network system in Figure 8 has four minproducts: ab , ac , de and df . The subsystem containing $\{a, b, c\}$ has two minproducts: ab and ac , while the subsystem containing $\{d, e, f\}$ has the other two minproducts: de and df . The operation of a system is determined by its minpaths. Since the minproducts of the two subsystems do not share any components, their operation is independent of each other. Also the minproducts of the two subsystems are minproducts of the entire system as well. Therefore $\{a, b, c\}$ and $\{d, e, f\}$ are parallel subsystems.

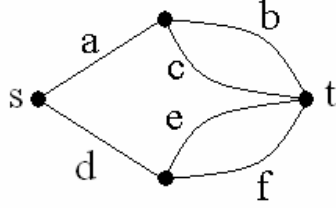


Figure 8

If the system can be divided into two parallel subsystems, having PS_1 and PS_2 as the sum of minproducts for each subsystem, then $RF(\Sigma) = RF(PS_1) + RF(PS_2)$. A disjoint reliability formula can be obtained for each parallel subsystem separately, $DRF(PS_1)$ and $DRF(PS_2)$. What matters the most is the way the formulas are combined in the end to form a reliability formula of the entire system in which all Boolean terms are mutually disjoint.

The simplest way is to have

$$DRF(PS_1) \vee DRF(PS_2), \quad (17)$$

which is a valid GSDP Boolean term as long as $DRF(PS_1)$ and $DRF(PS_2)$ are GSDP Boolean terms themselves.

In the case that $DRF(PS_1)$ is a GSDP Boolean term and $DRF(PS_2)$ is a sum of disjoint GSDP Boolean terms, then

$$DRF(PS_1) + \overline{DRF(PS_1)} \wedge DRF(PS_2) \quad (18)$$

should be used, since $\overline{DRF(PS_1)}$ is also a GSDP Boolean term, and all terms that are generated using the conjunction distributive law are mutually disjoint.

The most complex case appears when neither of $DRF(PS_1)$ and $DRF(PS_2)$ is a GSDP Boolean term, but rather sums of disjoint GSDP Boolean terms. Since $\overline{DRF(PS_1)}$ can contain sub-terms that are not GSDP Boolean terms, a different method should be used to

obtain a disjoint reliability formula for $\overline{DRF(PS_1)}$. According to (11), $\overline{DRF(PS_1)}$ is precisely the reliability problem with PS_1 products considered mincuts with implicit variable negations. Thus a reliability formula for the entire system is

$$DRF(PS_1) + DRF(Cutset(PS_1)) \wedge DRF(PS_2), \quad (19)$$

for which the conjunction distributive law is used.

The `Process_Pathset()` subroutine of the BT-03 algorithm attempts to detect parallel subsystems. For systems with no parallel subsystems, the `Process_Parallel_Free_Pathset()` subroutine is invoked (see Appendix A). The `Process_Cutset()` subroutine constructs the $DRF(Cutset(PS_1))$ expression.

Example 13

Continuing Example 12, the system is divided into two subsystems; PS_1 has the minproducts ab and ac , while PS_2 has the other two minproducts de and df . Two equivalent disjoint reliability formulas for PS_1 are $ab + a\bar{b}c$ and $a(b \vee c)$. Similarly, $DRF(PS_2) = de + d\bar{e}f = d(e \vee f)$. For demonstration purposes, the following cases can appear:

- If $DRF(PS_1)$ and $DRF(PS_2)$ are each GSDP terms:
 - $DRF(PS_1) = a(b \vee c)$
 - $DRF(PS_2) = d(e \vee f)$
 - Then $DRF(PS_1 + PS_2) = a(b \vee c) \vee d(e \vee f)$.
- If $DRF(PS_1)$ is a GSDP term, while $DRF(PS_2)$ is a sum of disjoint GSDP terms:
 - $DRF(PS_1) = a(b \vee c)$
 - $DRF(PS_2) = de + d\bar{e}f$
 - Then $DRF(PS_1 + PS_2) = a(b \vee c) + \overline{a(b \vee c)}de + \overline{a(b \vee c)}d\bar{e}f$.
- If $DRF(PS_1)$ and $DRF(PS_2)$ are each sums of disjoint GSDP terms:

- $DRF(PS_1) = ab + a\bar{b}\bar{c}$
- $DRF(PS_2) = de + d\bar{e}\bar{f}$
- The $Cutset(PS_1)$ are a and bc , with implicit variable negations.
- One possible $DRF(Cutset(PS_1)) = \bar{a} + a\bar{b}\bar{c}$.
- Then $DRF(PS_1 + PS_2) = ab + a\bar{b}\bar{c} + \bar{a}de + \bar{a}d\bar{e}\bar{f} + a\bar{b}\bar{c}de + a\bar{b}\bar{c}d\bar{e}\bar{f}$, which has six terms.
- If the $DRF(Cutset(PS_1))$ is improved to have only one GSDP term $\bar{a} \vee \bar{b}\bar{c}$, then $DRF(PS_1 + PS_2)$ has only four GSDP terms: $ab + a\bar{b}\bar{c} + (\bar{a} \vee \bar{b}\bar{c})de + (\bar{a} \vee \bar{b}\bar{c})d\bar{e}\bar{f}$.

Series Reductions

In a directed graph, the simplest case of series reduction appears in the case of a perfect degree-2 non-terminal node with one inbound arc e_1 and one outbound arc e_2 . These two arcs are said to be in series. Since the functioning of one arc is of no use without the functioning of the other, the two arcs can be replaced by one arc e , with a probability of operating equal to $p = p_1 \cdot p_2$, where p_1 and p_2 are the probabilities of operation of e_1 and e_2 , respectively [3]. This is valid since both arcs are needed:

$$P[e_1 \wedge e_2] = P[e_1] \cdot P[e_2] = p_1 \cdot p_2. \quad (20)$$

In the Boolean algebra methods however, where the graph representation is not available, there are two ways to think of the series reduction idea: one at the component level, and the other at the system level.

At the component level, it should be noted that when two arcs e_1 and e_2 are in series, then it must be true that any minpath either contains both of them, or contains neither of them; otherwise the minpaths would not be minimal. As a consequence, whenever two variables a and b always appear together in all the minproducts, the BT-03 algorithm replaces them with one variable x . With this substitution, the problem has been reduced in the number of

variables. Once the newly reduced problem has been solved and the reliability expression obtained, then all occurrences of the x variable are replaced by $(a \wedge b)$. The BT-03 algorithm performs these reductions in the `Process_Cutset()` subroutine (see Appendix A).

Example 14

The network system in Figure 9 has four minproducts: $abcde$, abg , $fcde$ and fg . It is clear that components a and b always appear together, and the same is true for c , d and e . If the former group of components is replaced by x , and the latter by y , the minproducts become xy , xg , fy and fg , which are easier to analyze. Assuming that $(x \vee f)(y \vee g)$ is a one-term reliability formula for the modified system then $(ab \vee f)(cde \vee g)$ is a reliability formula for the original system.

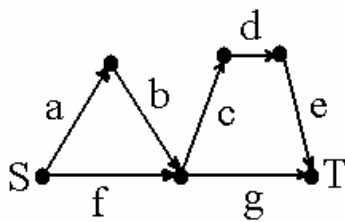


Figure 9

The other way of using series reductions is to consider subsystems that are in series. Two subsystems are in series if the functionality of one subsystem is independent of the functionality of the other and they always need to operate together. This also means that the failures of the subsystems are substitutes.

The BT-03 algorithm detects a simple form of subsystem series connections. Two subsystems are detected to be in series if the failure of one subsystem is independent of the failure of the other subsystem, and the failure of either subsystem implies the failure of the entire system. This definition is similar to the one for parallel subsystems, except that it deals with system failure. In other words, if the system described by mincuts can be divided in parallel subsystems, it means that the system described by minpaths can be divided into series subsystems.

The way to accomplish this is to first convert the minproducts of the system into mincut products. Second, the mincut products should be separated into two parallel subsystems, PS_1^{cutset} and PS_2^{cutset} . Third, each mincut subsystem should consequently be converted back into minproducts, thus obtaining two series subsystems SS_1 and SS_2 of the original system. Fourth, a disjoint reliability formula can be obtained for each series subsystem separately. Fifth, the terms that result from $DRF(SS_1) \cdot DRF(SS_2)$ using the conjunction distributive law are all disjoint.

1. $RF(\Sigma) = \sum EP_i$
2. $\overline{RF(\Sigma)} = \sum EC_j = PS_1^{\text{cutset}} + PS_2^{\text{cutset}} = \text{Cutset}(\Sigma)$
3. $SS_1 = \text{Pathset}(PS_1^{\text{cutset}}) = \overline{PS_1^{\text{cutset}}}$, $SS_2 = \text{Pathset}(PS_2^{\text{cutset}}) = \overline{PS_2^{\text{cutset}}}$
4. $RF(\Sigma) = \overline{PS_1^{\text{cutset}} + PS_2^{\text{cutset}}} = DRF(SS_1) \cdot DRF(SS_2)$
5. $DRF(\Sigma) = DRF(SS_1) \wedge DRF(SS_2)$. (21)

Example 15

The minproducts in Example 14 are $abcde$, abg , $fcde$ and fg . Using the `Convert_minpaths_to/from_mincuts(Products)` subroutine, the following mincuts are obtained: af , bf , cg , dg and eg . Using the `Parallel_Division()` subroutine, two parallel subsystems in the mincuts are obtained. PS_1^{cutset} consists of af and bf , while PS_2^{cutset} consists of cg , dg and eg . Converting each of the subsystems into minpaths, SS_1 will contain ab and f , while SS_2 will contain cde and g . One possible $DRF(SS_1)$ is $ab \vee f$, and $DRF(SS_2)$ is $cde \vee g$, and therefore $DRF(\Sigma) = (ab \vee f) \wedge (cde \vee g)$.

Standard SDP Reductions

Whenever the series and parallel reductions cannot be performed due to the structure of the minpaths, then the simple method employed by most SDP algorithms [1, 5, 11, 14, 16, 18,

20, 24] can always be used to reduce the problem further. To reiterate, the problem is to determine a disjoint reliability formula for $RF(\Sigma) = EP_1 + EP_2 + \dots + EP_h$. The approach of the SDP algorithms is to use the equivalent form

$$DRF(\Sigma) = DRF(D_1) + DRF(D_2) + \dots + DRF(D_h), \quad (22)$$

where $D_1 = EP_1$, and in general, $D_i = \overline{EP_1} \wedge \overline{EP_2} \wedge \dots \wedge \overline{EP_{i-1}} \wedge EP_i$. The Boolean expressions D_i are mutually disjoint by construction, and therefore the problem has been reduced to finding a disjoint reliability formula for each D_i .

A feature to take advantage of is that each EP_j is a Boolean product with no complemented variables. Therefore for two Boolean products EP_i and EP_j , the relative complement $EP_j - EP_i$ can be defined, which is a Boolean product formed of all variables in EP_j that are not in EP_i . Accordingly, EP_i and $EP_j - EP_i$ share no variables. As an example, $abcd - bdef = ac$. The intersection of EP_i and EP_j , $EP_j \cap EP_i$, is defined as the Boolean product formed by all variables that are shared by both EP_i and EP_j . The following relationship is true: $EP_j = (EP_j - EP_i) \wedge (EP_j \cap EP_i)$.

It can be shown that $D_i = \overline{EP_1 - EP_i} \wedge \overline{EP_2 - EP_i} \wedge \dots \wedge \overline{EP_{i-1} - EP_i} \wedge EP_i$.

Proof

For each j between 1 and $i-1$,

$$\begin{aligned} (\overline{EP_j} \wedge EP_i) &= \overline{(EP_j - EP_i) \wedge (EP_j \cap EP_i)} \wedge EP_i \\ &= \overline{(EP_j - EP_i) \vee (EP_j \cap EP_i)} \wedge EP_i \\ &= \overline{(EP_j - EP_i) \wedge EP_i} \vee \overline{(EP_j \cap EP_i) \wedge EP_i} \\ &= \overline{EP_j - EP_i} \wedge EP_i, \text{ since } \overline{EP_j \cap EP_i} \wedge EP_i \text{ is an impossible event.} \end{aligned}$$

$$D_i = \overline{EP_1} \wedge \overline{EP_2} \wedge \dots \wedge \overline{EP_{i-1}} \wedge EP_i$$

$$\begin{aligned}
&= (\overline{EP_1} \wedge EP_i) \wedge \cdots \wedge (\overline{EP_{i-1}} \wedge EP_i), \text{ by using idempotency;} \\
&= (\overline{EP_1 - EP_i} \wedge EP_i) \wedge \cdots \wedge (\overline{EP_{i-1} - EP_i} \wedge EP_i) \\
&= \overline{EP_1 - EP_i} \wedge \overline{EP_2 - EP_i} \wedge \cdots \wedge \overline{EP_{i-1} - EP_i} \wedge EP_i, \text{ by using idempotency.}
\end{aligned}$$

◇

This implies that it is sufficient to negate only the relative complements $\{EP_j - EP_i\}$, and not the actual products $\{EP_j\}$. The disjoint reliability formula becomes

$$\begin{aligned}
DRF(D_i) &= DRF(\overline{EP_1 - EP_i} \wedge \overline{EP_2 - EP_i} \wedge \cdots \wedge \overline{EP_{i-1} - EP_i}) \wedge EP_i \\
&= DRF(\overline{(EP_1 - EP_i) + (EP_2 - EP_i) + \cdots + (EP_{i-1} - EP_i)}) \wedge EP_i. \quad (23)
\end{aligned}$$

The same trick that was used for (19) can be used for (23). The expression $DRF(\overline{(EP_1 - EP_i) + (EP_2 - EP_i) + \cdots + (EP_{i-1} - EP_i)})$ can represent the pathset reliability expression of a system for which the relative complements $\{EP_j - EP_i\}$ can be considered mincuts with implicit variable negations. The disjoint reliability formula for D_i is

$$DRF(D_i) = DRF(\text{Mincuts}(\{EP_j - EP_i\}_{j=1}^{i-1})) \wedge EP_i, \quad (24)$$

for which the conjunction distributive law is used. The BT-03 algorithm uses the Process_Cutset() subroutine to construct $DRF(\text{Mincuts}(\{EP_j - EP_i\}_{j=1}^{i-1}))$.

Example 16

The minproducts of a system are $abcx$, $acdy$ and $wxyz$. $DRF(D_1)$ is just the first minproduct $abcx$. D_2 is the Boolean expression for $acdy$ made disjoint from $abcx$. The relative complement is $abcx - acdy = bx$. When bx is inverted using the Convert_minpaths_to/from_mincuts() routine, b and x are obtained, and so $b \vee x$ is the disjoint formula for them. Because of the implicit variable negations, each variable is complemented, and therefore $DRF(\text{Cutset}(EP_1 - EP_2)) = \overline{b} \vee \overline{x}$. This means that $DRF(D_2) = acdy(\overline{b} \vee \overline{x})$. D_3 is the Boolean expression for $wxyz$ made disjoint from the first two minproducts. The two relative complements are abc and

acd . They are considered mincuts with implicit variable negations. When converted to minpaths, a , c and bd are obtained. A disjoint reliability formula for them is $a \vee c \vee bd$. Because of the implicit variable negations, each variable is complemented: $\bar{a} \vee \bar{c} \vee \bar{b} \bar{d}$. The $DRF(D_3)$ has become $(\bar{a} \vee \bar{c} \vee \bar{b} \bar{d})_{wxyz}$. The overall reliability formula for the system is $DRF(\Sigma) = abcx + acdy(\bar{b} \vee \bar{x}) + (\bar{a} \vee \bar{c} \vee \bar{b} \bar{d})_{wxyz}$.

A slight improvement on the idea above exists, and it is possible only using GSDP. In the expression for $DRF(\Sigma)$ in (22), $EP_1 + EP_2$ requires two terms, $DRF(EP_1) + DRF(\overline{EP_1} \wedge EP_2)$. However, with GSDP, $EP_1 + EP_2$ can always be expressed using one term: $EP_1 + EP_2 = ((EP_1 - EP_2) \vee (EP_2 - EP_1)) \wedge (EP_1 \cap EP_2)$. Therefore

$$DRF(\Sigma) = ((EP_1 - EP_2) \vee (EP_2 - EP_1)) \wedge (EP_1 \cap EP_2) + \\ + DRF(D_3) + DRF(D_4) + \dots + DRF(D_h). \quad (25)$$

Example 17

For the system in Example 16, the first two terms in the final reliability formula can be combined into one: $(bx \vee dy)ac$. Therefore $DRF(\Sigma) = (bx \vee dy)ac + (\bar{a} \vee \bar{c} \vee \bar{b} \bar{d})_{wxyz}$.

11. Experimental Results

This section describes how well the BT-03 algorithm performs in comparison to the KDH-88 algorithm [11]. While BT-03 is GSDP-based, KDH-88 is an MVI-based algorithm. Similar implementations have been provided for both of them, however the data structure used and the process of making Boolean terms disjoint is simpler for KDH-88 than for BT-03. The main goal is to improve the number of terms generated for the resulting reliability formula. Fifteen benchmark networks have been used for experimentation, and they are listed in Table 1. For each system, only the list of minproducts is given as input to the algorithm, though Table 1 shows both the number of minpaths as well as the number of

mincuts for additional insight. In addition to those, the number of disjoint Boolean terms in the resulting reliability formula and the actual time the computer⁶ took to run each of the two algorithms are listed as well.

No	Number of Minpaths	Number of Mincuts	Number of Variables	Number of disjoint terms		Time ⁷ (seconds)		Comments ⁸
				BT-03	KDH-88	BT-03	KDH-88	
1	4	4	5	3	4	0.008	0.006	Bridge Network
2	7	9	8	6	7	0.029	0.002	6 nodes, 8 links
3	9	8	8	8	11	0.041	0.004	5 nodes, 8 links
4	13	9	9	13	16	0.092	0.006	Modified Arpanet
5	13	28	12	12	18	0.147	0.006	Arpanet in 1971
6	14	18	15	13	27	0.156	0.006	7 nodes, 15 links
7	18	110	21	120	101	4.342	0.012	11 nodes, 21 links
8	18	24	13	20	31	0.224	0.010	9 nodes, 13 links
9	24	19	12	29	41	0.487	0.008	Modified Arpanet ⁹
10	20	20	12	23	32	0.297	0.010	Modified Arpanet
11	25	20	12	32	51	0.529	0.014	7 node, 12 link
12	29	29	13	52	75	1.168	0.016	8 node, 13 links
13	5	6	7	1	7	0.008	0.004	Figure 1
14	4	4	6	1	6	0.002	0.004	Figure 2
15	44	25	14	55	87	1.820	0.045	Reduced Arpanet

Table 1 – Results from Experimentation on Benchmark Networks

12. Discussion of Results

Table 1 shows that in general the BT-03 algorithm outperforms the KDH-88 algorithm with respect to the number of Boolean terms generated for the resulting reliability formula. At the same time, there is no doubt that the computer time is significantly higher for BT-03 than for KDH-88. This is expected since the modules of the BT-03 algorithm that detect parallel and series subsystems and components in series constitute a high computational overhead.

⁶ The computer used to run the experiments is a Pentium 4, 2GHz processor, 512M memory, with Red Hat Linux 7.3.

⁷ The time listed is the sum of user time and system time in seconds.

⁸ All networks except the 13th and the 14th have been taken from [21].

⁹ This is Abraham's example [1]. Abraham's algorithm is SVI-based and it produces 71 disjoint products for this example.

One example stands out from the rest, as it is characterized by a number of differences. Specifically, for the seventh network the number of disjoint terms generated is larger for BT-03 than it is for KDH-88: 120 as opposed to 101. Another thing to notice though is that the system features a significantly larger number of mincuts than of minpaths, while none of the other systems do. This suggests that other types of reductions besides parallel and series reduction might be needed before the standard SDP reductions should be employed. This is because the standard SDP reductions make extensive use of the mincuts of the systems. In contrast to other algorithms such as KDH-88 that perform the reliability analysis using only the minpaths or the mincuts of the system, BT-03 uses both concepts concomitantly.

Suggested Future Studies

This section lists six ways the BT-03 algorithm can be improved. The first three are concerned with improving the number of terms of the resulting reliability formula. The other three ways have to do with reducing BT03 computer.

First, a preprocessing heuristic for BT-03 needs to be developed. Preprocessing heuristics are typically algorithm-specific. It has been observed that the optimal preprocessing strategy may differ for various SDP algorithms [23]. Preprocessing is necessary when the algorithm is sensitive to the order in which the minproducts are considered for analysis. It is however desirable to have a preprocessing heuristic that would come close most of the time to minimizing the size of the resulting reliability formula. The BT-03 algorithm can benefit enormously from preprocessing. The current version orders the minproducts increasingly by size.

Second, the parallel and series reduction tests should be extended in order to reduce the number of terms in the resulting reliability formula. The current version of BT-03 detects parallel and series subsystems only in the simple case in which the subsystems cover the entire system. In practice other types of parallel and series subsystems exist and they should be detected as well.

Third, a contraction / deletion type reduction (also called pivoting or factoring) can be included in the BT-03 algorithm. This type of reduction might prove beneficial in cases when no series and parallel reductions are possible, and the system has a large number of mincuts relative to the number of minpaths. When this is the case, the application of a standard SDP reduction forces subsystems to be analyzed in terms of their mincuts and this can generate a large number of disjoint terms of the resulting reliability formula.

Fourth, a complexity analysis for the BT03 algorithm should be performed to pinpoint the parts of the algorithm that require relatively much more computer time. This however can be a daunting thing to do. Not only that SDP algorithms are #P-Complete, but the complexity can be difficult to analyze even in the number of minpaths since BT03 makes use of both minpaths and mincuts of subsystems and the number of mincuts can be exponential in the number of minpaths.

Fifth, the implementation details for the current version should be reviewed, including the choice of the data structure. The current implementation was more concerned with the bigger picture of the algorithm, rather than with the efficiency at the lower level. This could have affected the computer time for the BT-03 algorithm.

Sixth, heuristics can be developed to determine when it is advisable to attempt series and parallel reductions. Since BT03 can detect only simple forms of parallelism and series, as the system grows in size it becomes more unlikely that the system can be reduced using simple series and parallel reductions, while it also becomes time costly to attempt such reductions.

14. Conclusion

One approach to reliability analysis is to use methods based on disjoint products in the framework of Boolean algebra. An example problem in reliability analysis is to compute the probability that a path exists between two nodes of a given stochastic network graph. More generally, stochastic coherent binary systems (SCBS) are considered together with

the minproducts that specify the operation of the systems. Making the minproduct expressions disjoint makes it easier to evaluate the reliability parameter.

Previous research has focused on two paradigms: single variable inversions (SVI) [1] and multiple variable inversions (MVI) [11]. These paradigms differ in the logical structure of the output. Much work has been done to improve both these paradigms to obtain a minimal reliability formula in the end. This paper introduced GSDP, a third possible paradigm, which extends the first two, and an algorithm that uses it, BT-03. So far, this algorithm still has some shortcomings in runtime but shows promise in reducing the size of the resulting reliability formula. Even though BT-03 is not practical with respect to running time, its functionality might be useful in cases when the reliability measure of a system is needed repeatedly and changes often, but the corresponding reliability formula remains the same over time. This is often the case with networks in which the topology is fixed, but the component probabilities of operation vary with time. BT-03 can benefit from improvements such as preprocessing, choice of data structures and implementation decisions. It also opens the possibility for advancements in the network reliability field as more understanding of the underlying complexities of the process becomes available.

Acknowledgements

I would like to thank Prof. Lorenzo Traldi in the Mathematics department at Lafayette College for his constant support in the last three years, and without whom none of this would have come to light. We have spent a huge amount of time together understanding and discussing concepts related to network reliability. I believe our collaboration has proven beneficial for both of us and I am forever grateful to him.

I would also like to thank Prof. Chun Wai Liew in the Computer Science department at Lafayette College for his confidence in me, the patience, and the guidance he has provided me on numerous occasions.

I thank Prof. Penny Anderson and Prof. Gary Gordon for being members of my thesis committee.

Appendices

Appendix A - Pseudo-code

Parallel Division()

Input: Products – list of minproducts: Products[i], i between 1 and h

Output: Top – list of minproducts: Top[i] – it contains no parallel subsystems

Bottom – list of minproducts: Bottom[i] – it might contain parallel subsystems

1. Put all minproducts *Products[i]* in a collection called *Free*
 2. Have a set of components called *Covered*, which is initially empty
 3. Let *P* be any product from *Free*
 4. Remove product *P* from *Free* and put it in another collection called *Taken*, and include all of its components into *Covered*
 5. If there is a product in *Free* that contains some variables from *Covered*, call it *P* and go to step 4
 6. If *Free* is empty, there are no parallel subsystems
 7. If *Free* is not empty, then *Taken* is in parallel with *Free*
 8. Let *Top* equal *Taken*.
 9. Let *Bottom* equal *Free*
- RETURN *Top* and *Bottom*

End

Process Pathset()

Input: Products – list of minproducts: Products[i], i between 1 and h

Output: Result – list of disjoint terms: Result[i]

1. If all products are non-overlapping, join them all with OR operators
Result = (Products[1] OR Products[2] OR ... OR Products[h])
RETURN *Result*
- // Testing for parallel systems in minpaths:
2. Split the products into 2 groups: *Top* and *Bottom*, such that they are in parallel
Top, Bottom = *Parallel_Division(Products)*
// *Top* is a parallel free system
// *Bottom* is not necessarily a parallel free system
// *Bottom* might be empty – in which case the original products were parallel free

3. $topDS = Process_Parallel_Free_Pathset(Top)$
4. If *Bottom* is empty,
 $Result = topDS[i]$
RETURN *Result*
5. $bottomDS = Process_Pathset(Bottom)$ // recursive call
6. If *topDS* and *bottomDS* all have one term, join them with an OR operator
 $Result = (topDS[1] \text{ OR } bottomDS[1])$
RETURN *Result*
7. Swap *Top* and *Bottom* (and *topDS* and *bottomDS*) such that *Top* has less terms than *Bottom*
8. If *topDS* has only one term, then
 $Result = topDS[1] + ((NOT\ topDS[1]) \text{ AND } bottomDS[i])$
RETURN *Result*
9. $topCuts = Process_Cutset(Top)$
 $Result = topDS[i] + (topCuts[j] \text{ AND } bottomDS[k])$

RETURN *Result*

End

Process Parallel Free Pathset()

Input: Products – list of minproducts *Products[i]*

Output: Result – list of disjoint terms *Result[i]*

// Testing for series reductions (parallel mincuts)

1. $Cuts = Convert_minpaths_to/from_mincuts(Products)$ // variable negations are implicit
2. Split the cuts into 2 groups: *Top* and *Bottom*, such that they are in parallel.
3. If *Bottom* is empty, then no series reductions can be done
 $Result = Process_Canonical_Pathset(Products)$
RETURN *Result*
4. $TopPaths = Convert_minpaths_to/from_mincuts(Top)$ // variable negations are implicit

5. *BottomPaths* = *Convert_minpaths_to/from_mincuts*(*Bottom*) // variable negations are implicit

// The implicit negations have canceled out

6. *TopDS* = *Process_Pathset*(*topPaths*)

7. *BottomDS* = *Process_Pathset* (*bottomPaths*)

8. *Result* = (*TopDS*[*j*] AND *BottomDS*[*k*])

RETURN *Result*

End

Process Canonical Pathset()

Input: Products – list of minproducts: *Products*[*i*]

Output: Result – list of disjoint terms: *Result*[*i*]

1. If *Products* has only one term

Result = *Products*

RETURN *Result*

2. Order the *Products* list // for lack of better preprocessing strategy, increasing order can be used

// as an extension, first two can be treated together.

3. *Result* = Empty

4. *Previous_Products* = *Products*[1]

5. FOR (*i* between 2 and *h*)

Result = *Result* + *Get_Di_Formula*(*Products*[*i*], *Previous_Products*)

Previous_Products = *Previous_Products* + *Products*[*i*]

RETURN *Result*

End

Get Di Formula()

Input: Current_Product – minproduct

Previous_Products – list of minproducts: *Previous_Products*[*i*]

Output: Result – list of disjoint terms: Result[i]

1. *Rel_Cmpl = Relative_Complements(Current_Product, Previous_Products)*
2. *Disjoint_Cmpl = Process_Cutset(Rel_Cmpl)*
3. *Result = (Current_Product AND Disjoint_Cmpl[i])*

RETURN Result

End

Process Cutset()

Input: Products – list of mincuts: Product[i]

Output: Result – list of disjoint terms: Result[i]

1. *Min_Cuts = First_Contract_Component_Series(Products)*
2. *Min_Paths = Convert_minpaths_to/from_mincuts(Min_Cuts) // variable negations are implicit*
4. *Min_Paths = Second_Contract_Component_Series(Min_Paths)*
5. *If Min_Paths is Empty*
Result = Empty
RETURN Result
6. *Disjoint_Cmpl = ProcessPathset(Min_Paths)*
7. *Disjoint_Cmpl = Restore_Second_Contract_Component_Series(Disjoint_Cmpl)*
8. *Disjoint_Cmpl = Negate_Each_Variable(Disjoint_Cmpl)*
// takes care of the implicit variable negations
9. *Disjoint_Cmpl = Restore_First_Contract_Component_Series(Disjoint_Cmpl)*

RETURN Result

End

Bibliography

- [1] J. A. Abraham, "An Improved Method for Network Reliability", *IEEE Trans. Reliability*, vol R-28, 1979 April, pp 58-61.
- [2] K. K. Aggarwal, J. S. Gupta, K. B. Misra, "A Simple Method for Reliability Evaluation", *IEEE Trans. Communications*, vol COM-23, 1975 May, pp 563-566.
- [3] M. O. Ball, C. J. Colbourn, J. S. Provan, "Network Reliability" Technical Research Report TR 92-74, Systems Research Center, University of Maryland, 1992.
- [4] M. O. Ball, G. L. Nemhauser, "Matroids and a Reliability Analysis Problem", *Mathematics of Operations Research*, Vol. 4, No. 2, May 1979, pp 132-143.
- [5] F. Beichelt, L. Spross, "An Improved Abraham-Method for Generating Disjoint Sums", *IEEE Trans. Reliability*, Vol. R-36, No. 1, 1987 April, pp 70-74.
- [6] F. Beichelt, L. Spross, "Comment on 'An Improved Abraham-Method for Generating Disjoint Sums'", *IEEE Trans. Reliability*, Vol. 38, No. 4, 1989 October, pp 422-424.
- [7] M. Chari, C. Colbourn, "Reliability polynomials: a survey", *Journal of Combinatorics, Information and System Sciences*, Vol. 22, 1997, pp 177-193.
- [8] K. Dohmen, "Inclusion-exclusion and network reliability", *The Electronic Journal of Combinatorics*, Vol. 5, No. 1, 1998, paper no. 36.
- [9] L. Fratta, U. G. Montanari, "A Boolean Algebra Method for Computing the Terminal Reliability in a Communication Network", *IEEE Trans. Circuit Theory*, vol. C-20, 1973 May, pp 203-211.
- [10] A. Grnarov, L. Kleinrock, M. Gerla, "A New Algorithm for Network Reliability Computation", *Proc. Computer Networking. Symp.*, 1979 Dec.
- [11] K. D. Heidtmann, "Smaller Sums of Disjoint Products by Subproduct Inversion", *IEEE Trans. Reliability*, vol. 38, No. 3, 1989 August, pp 305-311.
- [12] M.O. Locks, "Inverting and Minimizing Path Sets and Cut Sets", *IEEE Trans. Reliability*, Vol. R-27, No. 2, June 1978, pp 107-109.
- [13] M.O. Locks, "Inverting and Minimizing Boolean Functions, Minimal Paths and Minimal Cuts: Noncoherent System Analysis", *IEEE Trans. Reliability*, Vol. R-28, No. 5, December 1979, pp 373-375.
- [14] M.O. Locks, "A Minimizing Algorithm for Sum of Disjoint Products", *IEEE Trans. Reliability*, Vol. R-36, No. 4, 1987 October, pp 445-453.

- [15] M.O. Locks, J. M. Wilson, "Note on Disjoint Products Algorithms", *IEEE Trans. Reliability*, Vol. 41, No. 1, 1992 March, pp 81-84.
- [16] T. Luo, K. S. Trivedi, "An Improved Algorithm for Coherent-System Reliability", *IEEE Trans. Reliability*, Vol. 47, No. 1, 1998 March, pp 73-78.
- [17] F. Moskowitz, "The analysis of redundancy networks", *AIEE Trans. Commun. Electron.*, vol. 39, 1958, pp627-632.
- [18] S. Rai, M. Veeraraghavan, K. S. Trivedi, "A Survey of Efficient Reliability Computation Using Disjoint Products Approach", *Networks*, vol. 25, No. 3, May 1995, pp. 147-163.
- [19] D. R. Shier, D. E. Whited, "Algorithms for Generating Minimal Cutsets by Inversion", *IEEE Trans. Reliability*, Vol. R-34, No. 4, 1985 October, pp 314-319.
- [20] S. Soh, S. Rai, "Computer Aided Reliability Evaluator for Distributed Computing Networks", *IEEE Trans. Parallel and Distributed Systems*, Vol. 2, No. 2, April 1991, pp 199-213.
- [21] S. Soh, S. Rai, "Experimental Results on Preprocessing of Path/Cut Terms in Sum of Disjoint Products Technique", *IEEE Trans. Reliability*, Vol. 42, No. 1, 1993 March, pp 24-33.
- [22] L. Traldi, "Non-minimal Sums of Disjoint Products", 2003.
- [23] L. Traldi, A. O. Balan, "Preprocessing Minpaths for Sum of Disjoint Products", *IEEE Trans. Reliability*, to appear.
- [24] M. Veeraraghavan, K. S. Trivedi, "An Improved Algorithm for Symbolic Reliability Analysis", *IEEE Trans. Reliability*, Vol. 40, No. 3, 1991 August, pp 347-358.
- [25] D. J. A. Welsh, "Complexity: knots, colourings and counting", London Mathematical Society Lecture Notes Series, Vol. 186, Cambridge University Press, 1993.
- [26] J. M. Wilson, "An Improved Minimizing Algorithm for Sum of Disjoint Products", *IEEE Trans. Reliability*, Vol. 39, No. 1, 1990 April, pp 42-45.