Efficient Evaluation of Multistate Network Reliability

by

Guanghan Bai

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Engineering Management

Department of Mechanical Engineering

University of Alberta

Abstract

In network reliability context, it is often assumed that both the components and the systems can take two possible states, completely working or totally failed. However, in many real-world network systems, the component states and the system state can take more than two values. This multi-state phenomenon has to be addressed when specific performance measures such as capacity are taken into consideration. The network must not just be connected but function at a certain performance level. For a two-terminal multistate network with a source node, *s*, and a sink node, *t*, the reliability is defined as the probability of successfully sending a required amount of flow, *d*, from node *s* to node *t*, which is the probability that the flow throughput is not less than *d*. The capacity (state) of each component can take discrete, non-negative integer values from 0 to its maximum capacity, following a certain probability distribution.

The overall objective of multistate network reliability is to provide engineers and managers useful tools to enhance their ability for design and maintenance of such networks. However, despite the increasing complexity of modern networks, the size of the network that can be analyzed by existing methods is still rather modest, given a modest number of states for each component. This is expected since the network reliability analysis problem is NP-hard. Consequently, research aimed at improving the efficiency of reliability evaluation is needed. In addition, most reported works focus on one specific demand at a time. However, during the design phase or operation phase, we are often interested in system reliability with respect to multiple possible demand levels, in order to obtain a complete picture of the system capability. Thus, an efficient and systematic method is desirable for network reliability evaluation with respect to all possible *d* values.

This thesis documents research contribution for efficient evaluation of multistate network reliability using the indirect approaches based on minimal path vectors. A *d*-MP refers to a minimal path vector with respect to demand level *d*.

- Firstly, an improved backtracking algorithm based on depth-first search is developed for finding all minimal paths (MPs) for binary networks. These MPs are used as building blocks for generating *d*-MPs for multistate networks.
- Secondly, given all the MPs, a recursive algorithm to search for all the *d*-MPs for all possible *d* values is developed based on breadth-first search. These *d*-MPs are used for reliability evaluation of multistate networks.
- Thirdly, given all *d*-MPs, ordering heuristics are proposed to improve the efficiency of the Recursive Sum of Disjoint Product (RSDP) method for evaluating multistate network reliability.
- Fourthly, given all *d*-MPs, an improved State Space Decomposition (SSD) algorithm is developed for evaluating multistate network reliability. Thorough efficiency investigations are conducted to compare the efficiency of the reported direct approaches and indirect methods, including the proposed improved SSD method and the RSDP method with ordering heuristics, for evaluating multistate network reliability.

With efficient reliability evaluation algorithms and methods, the research results out of this thesis provide the reliability engineers and facility managers a more powerful tool for the design and maintenance of more complex networks.

Contents

List of Tables

List of Figures

Acronyms

MP	Minimal Path
MC	Minimal Cut
***d*-MP**	*d*-Minimal Path
***d*-MC**	*d*-Minimal Cut
IE	Inclusion Exclusion
SDP	Sum of Disjoint Products
RSDP	Recursive Sum of Disjoint Products
SSD	State Space Decomposition
LRB	Lower Reliability Bound
URB	Upper Reliability Bound
DFS	Depth First Search
WFS	Width First Search
i.i.d.	Independent and Identically Distributed
UGFM	Universal Generating Function Method
MMDD	Multistate Multivalued Decision Diagrams

Chapter 1

Introduction

1.1 Background

With development in science and technology, the complexity of networks has grown enormously in the past few decades. As their complexity increases, it becomes more difficult to keep the networks constantly reliable. A failure of a single component can lead to cascading failure of the entire network. As well, failure to maintain the operation of such a complex network at its satisfactory level can be devastating. Take the case of power transmission and distribution networks. On August 14, 2003, the northeast blackout affected nearly 50 million people across US and Canada, damaged over 400 transmission lines and 531 generating units at 261 power plants [1] (see Figure 1.1). Shortly after the blackout of North America, another blackout took place in Southern Sweden and Eastern Denmark on September 23, with a total of 1.6 million people and 2.4 million people affected in Sweden and Denmark, respectively [1]. Only five days later, a tree flashover caused a sequence of events, leading to a blackout in Italy, causing a power shortage of 6400 MW [2]. More recently in 2012, a similar blackout took place in India (see Figure 1.2). With 620 million people been affected, the Indian blackout is recorded as the world's largest power blackout to date.

In order to measure and predict the reliability of such networks, researchers make use of system reliability analysis, where reliability is defined as the ability of a system to function

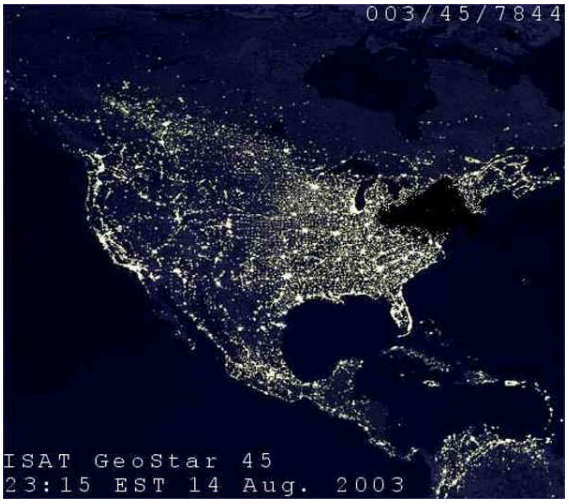

Figure 1.1: Northeast blackout 2003 [2]

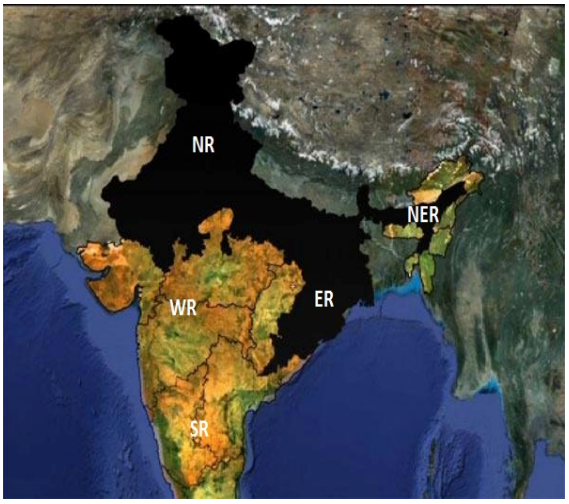

Figure 1.2: India blackout 2012 [4]

on a satisfactory level under stated condition for a specified time period. A system is made up of a set of components. A random variable is assigned to each component to indicate its reliability. In analyses of network reliability, a network is viewed as a system, and its constituent parts - a set of nodes and a group of links - are regarded as its components.

One important task of system reliability analysis is to find the relationship between component reliabilities and system reliability. A structure function is used to achieve this goal by mapping the states of the components into system states. We can define different structure functions in terms of different system reliability measures. For instance, consider a network with a set of nodes, K, and a node, $s \in K$, the k-terminal reliability measure evaluates the ability to perform certain "task" from s to each node in K. One important special case is the two-terminal reliability measure for which $|K| = 2$.

In traditional network reliability analysis, it is assumed that a system is a binary network, in which both the components and system can take two possible states, completely working or totally failed. The system operates if there exists at least one live connection from s to each node in K. This class of reliability measure is also called the connectivity measure [5]. However, in many practical network systems, the system states and component states can, in fact, take

values other than 0 and 1. This issue arises when certain physical features, such as capacity, length and delay, are assigned to each link. Thus, the network must not just be connected but must function at a certain performance level. We refer to this kind of reliability problem as multistate network reliability.

In analyses of multistate network reliability, a network's reliability is evaluated relative to a certain performance criterion. They may, for instance, identify the length of the shortest path required for traveling between a pair of nodes, or the maximum time duration required for one node to reach another. In this study, we focus on the maximum flow problem, that is, the maximum amount of flow can be sent from the source node to the sink node. Maximum flow analyses are commonly applied to multistate networks ranging from oil/gas production and transportation networks [6], power transmission and distribution networks [7], supply networks [8], manufacturing networks [9], to computer networks [10]. In the case of a power transmission and distribution network, each component is assumed to represent a power transmission line. The source node represents the power plant and the sink nodes represent the power consumers. Demand is the amount of power required by each consumer. A key variable in the analysis is the capacity of each component, that is, the amount of power that can be sent over the transmission line. The capacity has a limit, which is the maximum amount of power it can send. For instance, the heating of conductors due to line losses sets a thermal limit. If too much current is drawn, conductors may sag too close to the ground, or conductors and equipment may be damaged by overheating. Due to long-term wear on the exposed component, different maintenance strategies, and environmental effects, the capacity of each component can be in more than two levels in practice.

This raises complex considerations for those undertaking the analysis. Theoretically, the random variable associated with the capacity of each component can take any type of distribution. It has been reported that finite discrete distribution is used for most cases [5]. In such situations, the number of states and corresponding probabilities are estimated based on

historical data. In addition, we limit ourselves to a two-terminal case in this study. We can transform a k–terminal problem to a two–terminal problem by adding an artificial component to each sink node and connecting all these additional components to a single artificial sink node. The capacity of each artificial component is the same as the demand of its corresponding sink node with reliability equal to 1.

Consider such a two-terminal network with a source node '*s*' and a sink node '*t*'. The reliability is defined as the probability of sending required amount of flow *d* from '*s*' to '*t,*' successfully, i.e., the probability that the maximal flow is no less than *d*. The capacity of each component can take discrete, non-negative integer values following a certain probability distribution. Thus, the capacity of each component can be regarded as an independent, discrete random variable taking values from 0 to its maximum capacity.

This has important implications for engineering design and management. For instance, during the network design stage, one can come up with several feasible designs under a budget limit and evaluate the reliability of each design based on historical data. Then the manager can select the 'optimal' design by picking the one with the highest reliability. However, the increasing level of network complexity implies that the problem of evaluating reliability itself is a challenge. Theoretically, the evaluation of two-terminal multistate network reliability is an NP-hard problem [5]. Thus, finding efficient methods for evaluating the reliability of such networks is important. However, despite the increasing complexity of modern networks, the size of the network that can be analyzed by existing studies is still rather modest, given a modest number of states for each component. Consequently, research aimed at improving the efficiency of reliability evaluation is needed. In addition, most previous studies have focused on one specific demand at a time. An efficient and systematic method is desirable for reliability evaluation of multistate network considering all possible *d* values.

1.2 Literature review

1.2.1 Early studies on multistate network reliability

Given its multi-disciplinary nature, the reliability evaluation of multistate network has gained the attention of researchers from different fields, including system reliability, graph theory, operations research, electric engineering, etc. We may summarize the early studies from these fields as follows.

The earliest work concerning this problem appeared in the field of electric engineering in 1965 by Frank and Hakimi [11]. An evaluation method based on maximum flow theory [13] and characteristic function was proposed. Frank and Hakimi [11] also pointed out that main difficulty is the computation, particularly when the component state distribution is continuous.

In the field of operations research in 1972, Doulliez and Jamoulle [12] studied a transportation network, in which the capacity of each component is assumed to be an independent discrete random variable. Doulliez and Jamoulle [12] proposed an efficient evaluation method based on the decomposition principle. This principle, formally known as the State Space Decomposition (SSD) method, was later improved and used in many approaches for reliability evaluation of multistate networks.

In 1975, from the perspective of graph theory, Evans [14] formally introduced the problem of two-terminal stochastic flow networks and obtained their reliability in terms of K-Lattices that are generated from each collection of minimal cuts (MCs) one by one. It requires all MCs as prior information. The number of total collections grows exponentially with the number of MCs. It further requires combining K-lattices to obtain disjoint lattices for simplifying the probability evaluation.

In the field of reliability engineering, early studies on multistate networks were discussed in a broader sense, in terms of multistate systems. In the 1970s, the primary concepts, including

structure function, minimal path/cut vectors, and coherency, were studied in El-Neweihi et al. [15], Barlow & Wu [16], and Griffith [17]. As pointed out by Barlow & Wu [16], a stochastic flow network can be regarded as a multistate system, where link capacities correspond to component states. This built a bridge between multistate system and stochastic flow network. Thus, the term *multistate network reliability* was formally introduced. Researchers began to focus on developing efficient methods and algorithms. There are generally two types of approaches among reported works, namely the direct approaches and the indirect approaches, which will be reviewed extensively in the next two subsections.

1.2.2 The direct approaches

The direct approaches take network configuration and component state distribution as input, and evaluate the reliability directly. They first apply a maximum flow algorithm, such as the Ford-Fulkerson flow-augmenting method [13], to find a component state vector to demand level *d*. This vector is used to decompose the state space into a set of accepted states, a set of unaccepted states, and disjoint sets of unspecified states. Each set of unspecified states is recursively decomposed using different component state vectors derived from the maximum flow algorithm until no sets of unspecified states are left. Then the reliability is the sum of probabilities of all the sets of accepted states.

As can be seen, the direct approaches are mainly based on decomposition principle, which was first proposed by Doulliez and Jamoulle [12]. Alexopoulos [18] corrected some errors contained in [12]. Jane and Laih [19] presented a correct method based on direct approaches and verified that the proposed method is efficient in comparison with the complete enumeration method.

The advantage of the SSD approach is that it is an integrated and direct method. It appears to be effective when the demand is close to the largest maximum flow value [18]. However, it still requires a lot of computational effort. Each implementation of the state space decomposition

requires an implementation of maximum flow algorithm and pivotal decomposition. It also requires large storage space to maintain the list of undetermined sets. In addition, it can only evaluate the reliability of a multistate network for a specific demand at a time.

With the expensive computational effort of this method, several studies suggested a two-phase approach to approximate the result. Firstly, one can execute the SSD method recursively until the efficiency is less than a specified level. Secondly, one can obtain the lower reliability bounds using the acceptable states already decomposed and the upper reliability bounds by using 1 minus the sum of the unacceptable states already decomposed [20]. Alternatively, one can also obtain the approximated result by conducting a crude Monte Carlo simulation [21] or a Monte Carlo simulation with advanced importance sampling plans [22] [23].

1.2.3 The indirect approaches

Before 1990s, the direct approaches were recognized as efficient methods for reliability evaluation of multistate networks [5]. However, in 1993 and 1995, Jane et al. [15] [16] pointed out that the reliability evaluation of such a network can, in fact, be carried out in terms of minimal path vectors (*d*-MPs) or minimal cut vectors (*d*-MCs), which are lower boundary points and upper boundary points of each level *d*. This idea was inspired by [14], which indicated that the computational effort can be saved if all the lower and upper boundary points can be obtained. Thus, given the network configuration and component state distribution, the problem can be solved in two stages: (1) search for all the *d*-MPs or *d*-MCs; and (2) obtain the reliability by calculating the union probability of the vectors for which the component state vector is greater (smaller) than or equal to at least one of *d*-MP (*d*-MC). Given that *d*-MPs & *d*-MCs are dual, we focus on reported works for searching *d*-MPs and evaluating union probability using *d*-MPs. Readers are refer to [24] and [73] for methods using *d*-MCs. Yeh et al. [73] recently proposed a new method to find *d*-MCs by incorporating additional properties to verify *d*-MC candidates, which further improved the efficiency of finding all *d*-MCs.

Stage 1: Search for all minimal path vectors (*d*-MPs)

There are two approaches to search for all *d*-MPs for a specific *d* value reported in the literature. The first approach requires all binary minimal paths (MPs) as prior information. Given a two-terminal network with one source node and one sink node, a path is a sequence of links or nodes that connect the source node to the sink node. A minimal path is a path without cycle. There are three types of algorithms for searching for MPs in recent literature: symbolic expression-based algorithms [26] [27], augmentation-based algorithms [28] [29], and direct search-based algorithm [30] [31].

The symbolic expression-based algorithms represent the paths as symbolic forms and generate MPs based on Boolean algebra and/or other algebraic operators. Rai and Aggarwal [26] proposed a symbolic expression-based algorithm for finding all MPs using path polynomial and Boolean functions. Based on Universal Generating Function Method (UGFM), Yeh [27] defined a generalized composition operator to find all MPs.

The augmentation-based search algorithms are type of heuristic algorithms to search for MPs. It starts with only all the nodes of the original network, and add the link one at a time. Each time a link in included, new MPs are generated. All MPs can be found when all the links have been included. The augmentation-based search algorithm was first proposed by Al-Ghanim [28]. Yeh [29] further improved Al-Ghanim's algorithm by eliminating the chance of generating duplicate MPs.

The direct search–based algorithm implements the depth-first search (DFS) mechanism to find all the MPs, which is a simple and efficient search strategy. It also used simple data structures such as connection matrix, and linked list to represent the network. Most of the reported studies on finding MPs are within this category. Based on graph theory and dual principle, Shen [69] proposed a DFS algorithm to search for all MPs using connection matrix.

Kobayashi and Yamamoto [70] further improved Shen's algorithm by incorporating additional processes base on the level set of nodes. Colbourn [30] reported an DFS algorithm which has a time complexity of $O(|E|\cdot\pi)$, where $|E|$ denotes the number of links, and π denotes the total number of MPs. Chen and Lin [31] reported another direct search-based algorithm to search for all MPs based on the concept of backtracking. It has a better efficiency of $O(\lambda\cdot\pi)$, where λ is the average number of links contained in a MP.

Given that all the MPs have been found, Lin et al. [25] proposed an MP-based formulation with three constraints to obtain the *d*-MP candidates using implicit enumeration. Then each generated *d*-MP candidate is verified through recursive comparison and all *d*-MPs are obtained for a particular *d* level. Lin [32] further proved that the second constraint was redundant and reduced the constraints proposed in [25] from 3 to 2. For cyclic networks, Yeh [33] further suggested removing unsatisfied solutions from generated *d*-MP candidates by detecting cycles if all cycles of the network were known.

There are also reported studies that generate *d*-MPs without using MPs. Instead of using MPs as a formulation base, Yeh [34] proposed a component-based formulation with 3 constraints to obtain the *d*-MP candidates using implicit enumeration. Ramirez-Marquez et al. [35] proposed another method using a so-called information sharing mechanism. It does not require MPs as prior knowledge, and the information-sharing mechanism seems to reduce the search space compared to implicit enumeration.

Stage 2: Evaluating union probability using *d*-MPs

Given that all the *d*-MPs for all the *d* values have been found, we can evaluate the reliability by calculating the probability of the union of the *d*-MPs for each *d* value. Hudson and Kapur [36] [37] proposed methods using the Inclusion-Exclusion (IE) principle and Sum of Disjoint Products (SDP) principle to evaluate this union probability given all *d*-MPs. However, these methods are not systematic and not efficient. Aven [38] proposed an algorithm based on SSD method, which

provided a systematic way of evaluating the union probability no matter how many *d*-MPs exist. It has been proved to be much more efficient than the IE method, except in situations in which the number of *d*-MPs is much smaller than the number of components, which exist in very few real-world network systems. Zuo et al. [39] proposed a recursive method based on the SDP principle, named the Recursive Sum of Disjoint Product (RSDP) method, for the union probability evaluation given all *d*-MPs. It is found that RSDP [39] is more efficient than the algorithm in [38] when the number of components of a network is not too small. Yeh [72] proposed another method based on RSDP method, namely iSDP (improved Sum-of-Disjoint Product) method. By incorporating additional simplification procedures to reduce the number of multiplications and summations, It is claimed to be more efficient than RSDP method in [39].

Another method for reliability evaluation of multistate networks is the Decision Diagram method [74] [75]. Shrestha et al. [71] proposed two approaches for implementing the Multistate Multivalued Decision Diagrams (MMDD) for reliability evaluation of multistate networks. The first approach is to generate the MMDD given all *d*-MPs and then evaluate the reliability. The second approach is to generate the MMDD directly from the multistate network and then evaluate the reliability. An efficient recursive algorithm was proposed in [71] to generate the MMDD based on the multistate networks.

1.2.4 Approximating network reliability

As the size of the networks becomes large, it is still cumbersome to obtain the exact reliability. Alternatively, the reliability bounds can provide approximate reliability values with less computational effort. One can choose a proper tradeoff between the computational effort and the evaluation precision. The multistate network reliability is calculated as the probability of the union of events, with each event involving a *d*-MP. Thus, if only a subset of *d*-MPs is available, the lower reliability bounds (LRB) can be obtained. The tradeoff between the computational effort and the evaluation precision can be achieved in two stages. In the first stage, instead of

using reported algorithms to fins all the *d*-MPs, one can develop an algorithm to find a subset of *d*-MPs. In the second stage, given that all the *d*-MPs have been found, one can use smaller number of the found *d*-MPs to obtain the LRB.

For the first stage, Satitsatian and Kapur [39] developed an efficient algorithm to generate a subset of *d*-MPs. The rationale behind the algorithm is to use a subset of binary minimal cuts (MCs) to generate a subset of *d*-MPs. When using more MCs, more *d*-MPs can be generated, resulting in improved LRB. All the *d*-MPs can be found without using all the MCs.

For the second stage, given that all the *d*-MPs have been found, Hudson and Kapur [41] proposed a method of obtaining the LRBs using a subset of *d*-MPs for multistate systems. The idea is that first LRB is computed using one *d*-MP. Then additional *d*-MPs are included one at a time and the final LRB is the exact reliability when all are included. These bounds are always between 0 and 1. In addition, these LRBs are always monotonically increasing. Hudson and Kapur [41] also discovered that the computation time is highly dependent on the order of *d*-MPs introduced and suggest lexicographic ordering and reversed lexicographic ordering for *d*-MPs.

In addition to bound evaluation, a Monte Carlo simulation offers another way of approximation. Ramirez-Marquez et al. [42] proposed a standard Monte-Carlo simulation method for estimating the reliability of multistate networks given all the *d*-MCs or *d*-MPs. Bulteau and Khadiri [23] proposed a new importance sampling Monte Carlo method. It used SSD method during the simulation process to transform the sampling in a given subset into the sampling in a smaller set recursively until it is not possible to accomplish new decompositions. They showed that their new sampling principle can offer substantial speedups and perform much better when highly reliable networks are analyzed.

1.3 Research scope and objective

Despite the great amount of reported works devoted to the indirect approaches using *d*-MPs/*d*-MCs, there are some limitations that need to be addressed.

Firstly, the size of the network that can be analyzed is still rather modest, given a modest number of states for each component, for example, a 30-component network with each component taking 3 possible states. Therefore, research aimed at developing more efficient algorithms will be quite worthwhile. First, with more efficient algorithms, the size of the networks being analyzed is extended. As a result, the reliability analysis of many practical complex networks becomes viable. Second, recall that one objective of multistate network reliability is to provide a useful tool to enhance the design of such networks. With more efficient algorithms, the optimal design of multistate networks, which requires iterative reliability evaluations, will also be more viable. Finally yet importantly, as more efficient algorithms are being developed, more insight is gained on such NP-hard problem.

Secondly, all the previous studies have focused only on one specific demand at a time. However, when we evaluate the reliability of multistate network systems during the design phase or operation phase, we are often interested in the system reliability with respect to each of the system performance levels, in order to obtain a complete picture of the system capability. For the minimal path vectors approach, this requires all *d*-MPs for all *d* levels to be generated. Thus, research aimed at developing an efficient and systematic algorithm to search for all *d*-MPs for all *d* values is of vital importance.

Thirdly, there is a lack of thorough experimental investigations to compare the efficiency of the reported direct approaches and indirect methods, including the proposed improved SSD method and the RSDP method with ordering heuristics, for evaluating multistate network reliability. There is no reported works conducting efficiency comparison between the direct approaches and the indirect approaches. Because the efficiency of the indirect approaches has

been improved in this thesis, it is necessary to conduct a thorough efficiency investigation between the reported direct approaches and the improved indirect approaches in this work. In addition, it has been claimed that RSDP method is more efficient than SSD when the number of components is not too small (greater than 15) [39]. However, the computational experiments in [39] are only conducted using hypothetical networks. In these hypothetical networks, the maximum number of *d*-MPs considered is 50. However, the number of *d*-MPs in many real networks is much bigger than 50 [19] [20]. Because the efficiency of both methods have been improved in this thesis, it is necessary to conduct a thorough efficiency investigation between RSDP method with ordering heuristics and the improved SSD method using both hypothetical networks and networks with known structures.

Based on the motivations discussed above, this thesis focuses on indirect approaches using *d*-MPs. Several research topics are defined, in the sense that the efficiency of each step will be improved, which are shown in Figure 1.3.

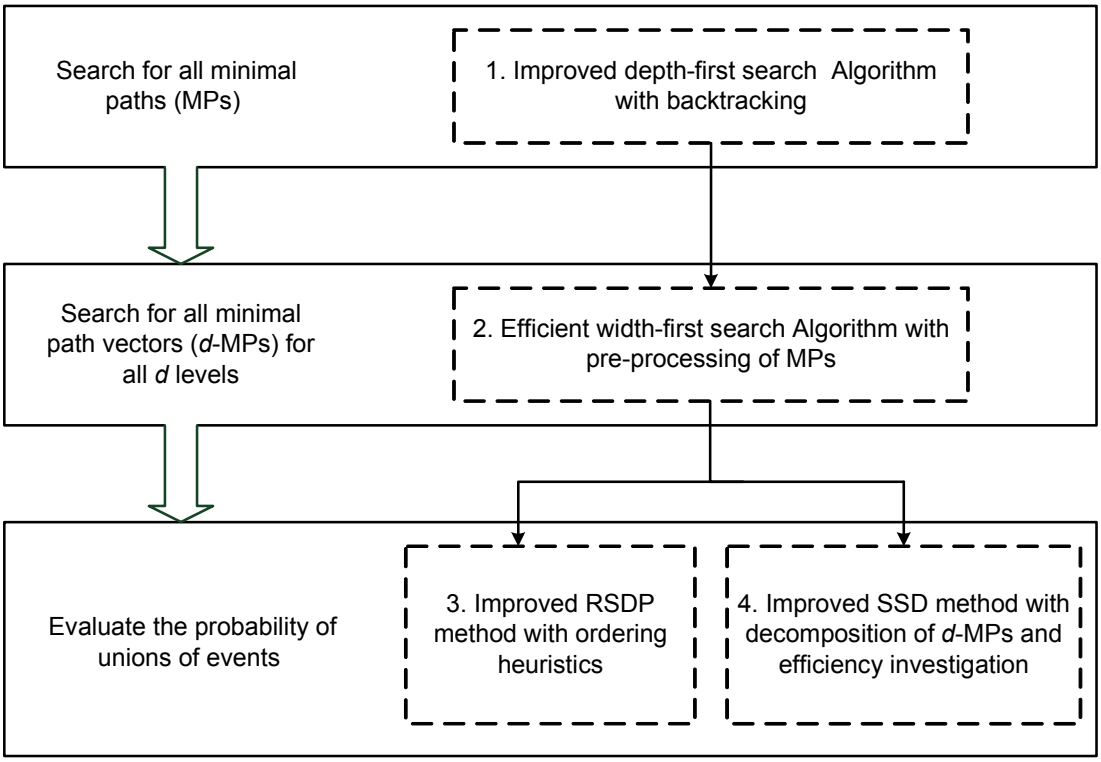

Figure 1.3: Outline of research topics

In the first stage, we aim at finding all the MPs. MPs play an important role in network reliability evaluation for binary networks. Currently, a direct search-based algorithm by Chen and Lin [31] is recognized as an efficient algorithm for finding all MPs. In [31], a linked path structure indexed by links is used to represent the network and a depth-first search algorithm with backtracking is proposed to find all MPs. However, we find that the algorithm can be improved by using a linked path structure indexed by nodes, which accepts both directed and undirected form of networks. Furthermore, based on the distance between each node and the sink node, additional conditions for backtracking can be incorporated to improve the efficiency of the algorithm. With the newly introduced linked path structure, and the additional backtracking conditions, an improved backtracking algorithm for finding all MPs is developed. These MPs are used as building blocks for generating *d*-MPs for multi-state networks in the next stage. This is covered in Chapter 3.

In the second stage, we focus on searching for *d*-MPs given all the MPs. As discussed earlier, existing studies on generating *d*-MPs are for a particular *d* value. However, if all *d*-MPs for all possible integer *d* values are required, we need to apply such methods multiple times with respect to all *d* values. A more efficient method is desirable to generate all *d*-MPs. We develop a recursive algorithm based on breadth-first search to search for all the *d*-MPs for all possible *d* values. Each *d*-MP candidate can be generated by a combination of one (*d*-1)-MP and the vector form of one binary minimal path. Thus, we can use binary MPs as building blocks to generate 2-MP candidates, and use 2-MPs and binary MPs as building blocks to generate 3-MP candidates and so forth. When the *d*-MPs with respect to the maximum *d* value have been found, all the *d*-MPs for all possible *d* values are obtained. A heuristic for pre-processing the MPs is proposed to improve the efficiency of the proposed algorithm. In addition, given all MPs, we compare the efficiency of the proposed algorithm with that of existing algorithm. We also compare the efficiency of the proposed algorithm with that of existing algorithm without MPs as prior information. This is described in Chapter 4.

In the third stage, we focus on obtaining the reliability of multistate networks by evaluating the unions of all *d*-MPs. We first focus on the RSDP method. In existing RSDP approach, all *d*-MPs are treated equally. However, we find that the importance of each *d*-MP is different, and different orderings affect the efficiency of reliability evaluation. Based on the observations above, we introduce the length definitions for *d*-MPs in a multistate two-terminal network, and develop four ordering heuristics, called O1, O2, O3, and O4, to improve the efficiency of the RSDP method for evaluating network reliability. This is discussed in Chapter 5.

In the fourth stage, we turn our attention to the SSD method. During each recursive call to decomposition, the existing method selects qualified *d*-MPs by comparing all *d*-MPs with the upper limiting point. However, we find that the set of *d*-MPs can also be decomposed recursively, and only those qualified *d*-MPs from previous set of unspecified states are needed to be compared with the current upper limiting point. Based on the observation above, we propose an algorithm to improve the efficiency of SSD method. An improved heuristic rule is also proposed for choosing a proper *d*-MP to decompose each set of unspecified states. Then, thorough efficiency investigations are conducted to compare the efficiency of the reported direct approaches and indirect approaches, including the proposed improved SSD method and the RSDP method with ordering heuristics, for evaluating multistate network reliability. This is covered in Chapter 6.

With all the algorithms and results in this thesis, it is expected that reliability engineers and facility managers will have a more powerful tool for the design and maintenance of more complex networks.

It is important to note that throughout this thesis, a number of fundamental assumptions are made as follows.

Assumptions

1) The structure function of the multistate network is coherent, that is, an improvement in any

component's state will not make the whole network's state worse [50].

2) All the nodes except the source node and the sink node satisfy flow conservation law [13].
3) The capacity of each component is a non-negative integer-valued random variable, which takes successive integer values from 0 to its maximum capacity according to the given probability distribution.
4) Components are weakly homogeneous, i.e. the components can have different levels of capacity, yet for the common levels, the capacities must be equal [35].
5) The network contains no parallel links. For networks with parallel links, a simple parallel reduction technique reported in [5] can be used to replace such parallel links by one single link. This parallel reduction is reliability preserving.
6) There are no self loops in the network [27].
7) There are no common-cause outages in the network [29].
8) All nodes of the networks are perfect. When both nodes and edge are failure prone, an approach reported in [30] can be used to transform the network into one with all nodes perfect.

1.4 Thesis organization

This paper-based thesis is prepared following the guidelines from the Faculty of Graduate Studies and Research (FGSR) at the University of Alberta. This thesis is composed of 7 chapters.

- Following the introduction in Chapter 1, Chapter 2 presents the fundamental knowledge of multistate network reliability, including state distribution, structure function, computational complexity, algorithm complexity analysis, data structure, search algorithm, and reliability evaluation methods.

- Chapter 3 reports an efficient recursive algorithm for finding all MPs. Some of the results of this chapter have been accepted for publication in the journal *Reliability Engineering and System Safety* [43], and presented at the *9th International Conference on Mathematical Methods in Reliability (MMR)* [44].
- Given all the MPs generated in Chapter 3, Chapter 4 introduces the relationships among *d*-MPs for different *d* levels and proposes a recursive algorithm based on breadth-first search to search for all the *d*-MPs for all possible *d* values. The major contributions of this chapter have been published in the journal *Reliability Engineering and System Safety* [45], and the conference proceedings of *Reliability and Maintainability Symposium (RAMS)* [46].
- Chapter 5 evaluates the unions of all the *d*-MPs found in Chapter 4 to obtain the reliability using RSDP method. Four heuristic ordering methods are developed to improve the efficiency of the RSDP method. Results in this chapter have been published in the journal *Reliability, IEEE Transactions on* [47], and presented at the *6th Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modelling (APARM)* [48].
- Chapter 6 evaluates the unions of all the *d*-MPs found in Chapter 5 to obtain the reliability using SSD method. All *d*-MPs are incorporated into the decomposition procedure and an improved heuristic for decomposition is proposed. In addition, computational experiments are conducted to compare 1) the proposed algorithm with exiting algorithm using SSD method; 2) the proposed algorithm with RSDP method incorporating ordering heuristic O1; 3) the indirect approaches incorporating the proposed algorithm with exiting direct approaches. The results of this chapter have been documented in [49].

- Chapter 7 summarizes the conclusions with observations and discussions. Possible directions for future works are also given.

Chapter 2

Fundamentals of Multistate Network Reliability

This chapter covers the fundamental knowledge of multistate network reliability, including state distribution, structure function, computational complexity, algorithm complexity analysis, data structure, search algorithm, and general reliability evaluation methods. Particularly, we discuss the interpretation of state distribution for multistate network, the complexity classes of recognition & optimization problems and their relationship to network reliability problem.

A typical directed two–terminal bridge network, shown in Figure 2.1, is used to illustrate some of the concepts covered in this chapter. The network consists of 4 nodes and 6 links. The nodes are perfectly reliable and the 6 links (components), represented by $a_1, a_2, \ldots, a_6$, can work at different capacities. Node 1 is the source node and node 4 is the sink node.

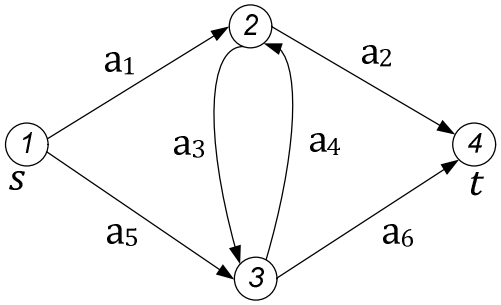

Figure 2.1: A two-terminal bridge network

A list of symbols is given, which is applicable to all chapters from Chapter 2 to Chapter 6. In the following chapters from Chapter 3 to Chapter 6, we only give list of symbols that are not included here and/or used for different meanings.

Notation list

$G=(V,E)$: the given network, where V is the set of all nodes, and E is the set of all links.

s : the source node.

t: the sink node.

d : the demand of flow from the source node to the sink node.

c : number of cycles in the network.

n : the number of components in the network.

L: the number of *d*-MPs in the network.

a_i: the i th component in the network system, $i=1,2,\ldots,n$.

$\mathbf{x}$: the component state vector, $\mathbf{x}=(\mathrm{x}_1,\ \mathrm{x}_2,\ \ldots,\ \mathrm{x}_n)$.

x_i: a discrete random variable representing the state of the i th component, x_i takes values $0,1,2,\ldots,M^i,\ i=1,2,\ldots,n$. where M^i represents the maximum state of component i .

$\mathbf{X}^{\mathbf{Max}}$: maximum state vector, $\mathbf{X}^{\mathbf{Max}}=(M^1,M^2,\ \ldots,\ M^i,\ldots,M^n)$.

S: the state space of considered multistate network.

$\phi(\bullet)$: the system structure function of the network.

π : the number of binary MP in the network.

λ: the average number of links for each MP.

η: the average number of nodes for each MP.

z^{i}: the $i\mathrm{th}$ *d*-MP of the considered multistate network.

$\mathrm{Pr}U(\bullet)$: the recursive function of the RSDP algorithm.

$\mathrm{TM_i}$: the $i\mathrm{th}$ term in the SDP calculation.

$\oplus$: the maximum operator, $z^{\mathrm{i}}\otimes z^{\mathrm{j}}\equiv\left(\max\left(\mathrm{z}_{\mathrm{k}}^{\mathrm{i}},\mathrm{z}_{\mathrm{k}}^{\mathrm{j}}\right)\right),1\le\mathrm{k}\le\mathrm{n}$.

$\otimes$: the minimum operator, $c^{\mathrm{i}} \otimes c^{\mathrm{j}} \equiv \left(\min\left(\mathrm{c}_{\mathrm{k}}^{\mathrm{i}}, \mathrm{c}_{\mathrm{k}}^{\mathrm{j}}\right)\right), 1 \le \mathrm{k} \le \mathrm{n}$.

$\mathbf{Y}^{\mathrm{i,j}}$: a vector generated by the $\oplus$ operator, $\mathbf{Y}^{\mathrm{i,j}} = \mathbf{z}^{\mathrm{i}} \oplus \mathbf{z}^{\mathrm{j}}$.

$\overline{\mathbf{z}}^{i}$: the logically equivalent vector of z^{i}.

c^{i}: the $i\mathrm{th}$ *d*-MC of the considered multistate network.

$\mathbf{ZH}^{\mathrm{k}}$: the $k\mathrm{th}$ highest state vector.

a^{k}: the number of elements that a *d*-MP has in common with $\mathbf{ZH}^{\mathrm{k}}$.

$F(\bullet)$: the score function.

$\left|\overline{\mathbf{z}}^{\mathrm{i}} - \overline{\mathbf{z}}^{\mathrm{j}}\right|$: the relative difference of the two *d*-MPs, $\mathbf{z}^{\mathrm{i}}$ and $\mathbf{z}^{\mathrm{j}}$.

C : the set of unspecified states.

$\boldsymbol{b}^0/\boldsymbol{b}$: the upper/lower limiting state point for the set of unspecified states.

A: the set of acceptable states.

U: the set of unacceptable states.

$\boldsymbol{v}^0/\boldsymbol{v}$: the associated critical value of corresponding set A/B.

τ: the total number of unspecified sets generated.

θ: the average number of qualified *d*-MPs.

ρ: the average number of *d*-MPs from the parental node of each unspecified set.

$\boldsymbol{w}$: the index vector of qualified *d*-MPs for each set of unspecified states.

$H(\bullet)$: the score function for selecting proper *d*-MP for pivotal decomposition.

Q: the stack for storing all the *d*-MPs.

B^0/B: the matrix for storing all the upper/lower limiting state points of the current sets of unspecified states.

2.1 Basic concepts of multistate reliability

Unless otherwise stated, the materials in this section are based on [50] and [5].

2.1.1 State distribution for multistate network reliability

In a multistate network, both the components and the network can work at different capacities (states). "State distribution" is used to describe the probabilities of a component or a network operating at different states. Suppose a component or a network has a maximum capacity of M. It has a total of $M+1$ states. A vector $\boldsymbol{p} = (p_0, p_1, \dots, p_M)$ is used to represent the state distribution of the component or the network. For example, Table 2. lists the state distribution of each component in the network (shown in Figure 2.1). Component a_1 can be in 4 states, ranging from 0 to 3. The probabilities of component a_1 being in state 0, 1, 2, and 3 are 0.05, 0.1, 0.25, and 0.6, respectively. Component a_2 can be in 3 states, ranging from 0 to 2. The probabilities of component a_2 being in state 0, 1, and 2 are 0.1, 0.3, and 0.6 respectively.

Table 2.1: State distribution, of the components in the example network

State	0	1	2	3
a_1	0.05	0.10	0.25	0.60
a_2	0.10	0.30	0.60	——
a_3	0.10	0.90	——	——
a_4	0.10	0.90	——	——
a_5	0.10	0.90	——	——
a_6	0.05	0.25	0.70	——

For repairable systems, "Availability" is often used to describe the component or the system's probability of operation. As well, "Reliability" is used for non-repairable system. Generally throughout this thesis, we use the term reliability to indicate the probability that a component or system operates. Thus, two interpretations for state distribution can be used. If

the component and the system are not repairable, p_i represents the probability of the component or the system be in capacity i, or state $i+1$ within specified time t. If the component and the system are repairable, p_i represents the probability of the component or the system be in capacity i at time t as t approaches infinity.

2.1.2 Minimal path (cut) vector

A general way for reliability evaluation of binary networks is using the minimal path (cut) set. A minimal path set is a smallest set of components that connect the source node to the sink node. The network operates if all the components contained in a minimal path set are functioning. For example, there are four MPs in the network (Figure 2.1), given as follows:

$$\mathrm{MP}_1=\{a_1,a_2\};\ \mathrm{MP}_2=\{a_5,a_6\};\mathrm{MP}_3=\{a_1,a_3,a_6\};\ \mathrm{MP}_4=\{a_2,a_4,a_5\}.$$

A minimal cut set for a binary network is a smallest set of components such that if all the components are simultaneously failed, the network is failed. For example, there are four MCs in the network (Figure 2.1), given as follows:

$$\mathrm{MC}_1=\{a_1,a_5\};\ \mathrm{MC}_2=\{a_2,a_6\};\mathrm{MC}_3=\{a_1,a_4,a_6\};\ \mathrm{MC}_4=\{a_2,a_3,a_5\}.$$

Correspondingly, the reliability evaluation of multistate networks can be carried using the minimal path (cut) vectors. Let $\mathbf{x}=(x_1,x_2,...,x_n)$ denotes the component state vector, where component i is in state x_i. A component state vector, $\mathbf{x}$, is called a minimal path vector to system state *d,* if $\phi(\mathbf{x})\geq d$, and $\phi(\mathbf{y})<d$ for any $\mathbf{y}<\mathbf{x}$, where $\phi(\bullet)$ is the system structure function and $\mathbf{y}<\mathbf{x}$ means $y_i<x_i$ for all i. Such a minimal path vector is also called a *d*-MP for short. For example, there are three *d*-MPs in the network (Figure 2.1) for *d* equals to 3, given as follows:

$$3-\mathrm{MP}_1=(3,2,1,0,0,1);3-\mathrm{MP}_2=(2,2,0,0,1,1);3-\mathrm{MP}_3=(2,1,1,0,1,2).$$

A component state vector, $\mathbf{x}$, is called a minimal cut vector to system state *d*, if $\phi(\mathbf{x}) < d$, and $\phi(\mathbf{y}) \geq d$ for any $\mathbf{y} \geq \mathbf{x}$, where $\mathbf{y} \geq \mathbf{x}$ means $y_i \geq x_i$ for all i and there exist at least one i that $y_i > x_i$. Such a minimal cut vector is also called a *d*-MC for short.

2.1.3 Structure function for multistate network reliability

Given the state distribution of each component, we are interested in finding the state distribution of the network. "Structure function" is used to identify the relationship between the component states and the system state. Let $\mathbf{x} = (x_1, x_2, ..., x_n)$ denotes the component state vector, where component i is in state x_i. $\phi(\mathbf{x})$ denotes the system state as a function of the component states, where $\phi(\bullet)$ is the system structure function. The structure function of multistate networks can be defined as follows. Let MC_1, MC_2, ... , MC_q be the sets of all minimal cut sets in a two-terminal network, the system sate $\phi(\mathbf{x})$ is given as [11]:

$$\phi(\mathbf{x}) = \min\left(\left|MC_1\right|,\ \left|MC_2\right|,\ \ldots,\ \left|MC_q\right|\right), \tag{2.1}$$

where $\left|MC_i\right|$ is the summation of component states included in this minimal cut set. For example, the structure function of the network (shown in Figure 2.1) is given as follows:

$$\phi(\mathbf{x}) = \min\left(a_1 + a_5, a_2 + a_6, a_1 + a_4 + a_6, a_2 + a_3 + a_5\right).$$

Assume the states of component a_1, a_2, …, a_6 are 3, 2, 1, 0, 0, 1, respectively, the state the of the network is given as follows:

$$\begin{aligned}\phi(\mathbf{x}) &= \min\left(a_1 + a_5, a_2 + a_6, a_1 + a_4 + a_6, a_2 + a_3 + a_5\right)\\ &= \min\left(3+0,\ 2+1,\ 3+0+1,\ 2+1+0\right)\\ &= \min\left(3,\ 3,\ 4,\ 3\right) = 3.\end{aligned}$$

With the structure function and demand *d*, the reliability of a multistate network is given as:

$$\Pr\left[\phi(\mathbf{x}) \geq d\right] = \Pr\left[\min\left(\left|MC_1\right|,\ \left|MC_2\right|,\ \ldots,\ \left|MC_q\right|\right) \geq d\right], \tag{2.2}$$

which further leads to:

$$\Pr\left[\phi(\mathbf{x}) \geq d\right] = \Pr\left(\left|MC_1\right| \geq d,\ \left|MC_2\right| \geq d,\ \ldots,\ \left|MC_q\right| \geq d\right). \tag{2.3}$$

The reliability of the multistate network (Figure 2.1) to demand level *d* is given as follows:

$$\Pr\left[\phi(\mathbf{x}) \geq d\right] = \Pr\left(a_1 + a_5 \geq d,\ a_2 + a_6 \geq d,\ a_1 + a_4 + a_6 \geq d,\ a_2 + a_3 + a_5 \geq d\right).$$

2.2 The computational complexity for multistate network reliability

In this section, we discuss several classes of complexity and give the class of complexity for multistate network reliability evaluation. The complexity analysis is also covered to measure and compare the efficiency of algorithms. Unless otherwise stated, the materials in this section are based on [50], [5] and [51].

2.2.1 Classes of computational complexity

There are two criteria for analyzing the efficiency of algorithm, namely time complexity and space complexity. Time complexity is often more important than space complexity. Time complexity evaluates the computation time growth of algorithms as a function of problem size. We introduce the following classes of problems depending on the time complexity of best existing algorithm.

Class P: An algorithm is polynomial time algorithm if its growth rate in computation time is, in the worst case, bounded by a polynomial function of problem size. We say a problem belongs to class P if a polynomial algorithm exists.

Class NP: NP refers to nondeterministic polynomial time. We say a problem is in class NP if a solution to the problem can be verified as “Yes” or “No” in polynomial time. The complexity class P is contained in NP. An open problem is this field is P=NP.

Class NP-complete: NP-complete problems are the hardest problems contained in NP. Every problem in NP is reducible to NP-complete problem in polynomial time. One important

property of NP-complete problem is that if a polynomial time algorithm exists for one NP-complete problem, there exists a polynomial algorithm for every problem in NP.

All the classes of complexity discussed above are for recognition and optimization problems. However, the problem of evaluating network reliability is essentially a counting problem. A class of counting problem analogous to class NP is called #P, in which testing whether or not it satisfies a property can be accomplished in polynomial time. As well, the counting version of NP-complete is named #P-complete. We define the last class of problem that applied to recognition, optimization, and counting problems as follows.

Class NP-hard: Any problem that is at least as hard as an NP-complete (#P-complete) problem is NP-hard. Thus, NP-hard problems can be reduced to NP-complete (#P-complete) problem through a polynomial time algorithm. NP-complete (#P-complete) problem is contained in NP-Hard problem. The relationship among these sets of problems can be found in Figure 2.2.

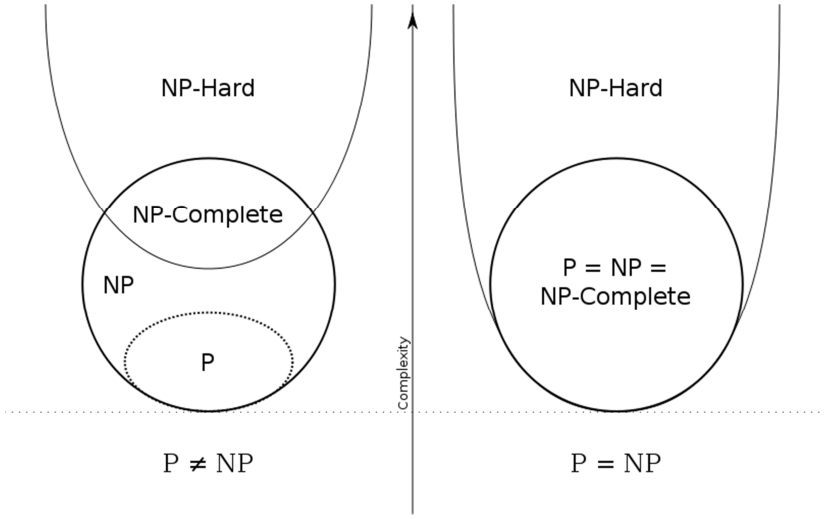

Figure 2.2: Euler diagram for P, NP, NP-complete, and NP-hard set of problems [52]

It has been reported that the reliability evaluation of two-terminal binary network is a NP-hard problem. This is because enumerating all the MPs/MCs for a two-terminal network is NP-hard. Because multistate network is a generalization of binary network, the reliability evaluation of multistate network is also an NP-hard problem. The indirect approaches for reliability

evaluation of multistate network consist of three sub-problems. Each sub-problem, searching for all MP, searching for all *d*-MPs, and evaluating the probability of unions of events are all NP-hard.

2.2.2 Algorithm complexity analysis

An intuitive method of measuring and comparing the efficiency of algorithms is to test their performances on real examples, such as real networks with different sizes (number of components) and structures. The following criteria are often used, including CPU time, number of arithmetic operations, and number of recursive calls. While the results may be biased depending on the examples, it remains to be a part of procedures when we analyze the efficiency of algorithms.

Another method is to analyze the complexity of algorithms mathematically. We often use time complexity and space complexity to measure the CPU time and the memory needed respectively. Both complexities can be expressed as functions of one or several parameters of the problem, such as the number of components, the number of states of each component, and the number of MPs. Given the term time complexity and space complexity, we are interested in the growth rate of these two complexities as the parameters of the problem increase. The mathematical symbol O is used to represent the growth rate of the complexity of an algorithm. The O notation describes the asymptotic upper bound on the growth rate of parameters of the problem. For example, Colbourn [30] reported an algorithm with an efficiency of $O(n \cdot \pi)$, where n is the number of components (links) of the network and π is the number of MPs.

2.3 Fundamentals for multistate network reliability evaluation

Algorithms are computerized procedures to solve problems. The main objective of algorithmic research is to design such procedure that the problem can be solved using less computational

resources. In this section, we discuss some fundamentals of algorithms used for evaluating multistate network reliability. Unless otherwise stated, the materials in this section are based on [50], [53]-[54].

2.3.1 Data structure of network

Before we develop efficient computer algorithms for network reliability analysis, we need data structures to represent the network configurations. In this thesis, the data structure of a network is used to search for all MPs, the first step for evaluating multistate network reliability. It contains the information of all the components and their neighboring components.

A network, $G = (V, E)$ consists of a set of nodes, V together with a set of undirected edges or directed arcs, E. In this thesis, we use the term "links" for both undirected edges and directed arcs.

One data structure for representing a network is "adjacency matrix". Adjacency matrix describes the direct relationship between each pair of nodes. If there is no direct connection from node i to node j, the entry at position (i, j) of the adjacency matrix is zero. Otherwise, the entry is one. The adjacency matrix of the network in Figure 2.1 is given as follows.

Node	1	2	3	4
1	1	1	1	0
2	0	1	1	1
3	0	1	1	1
4	0	0	0	1

Another popular data structure for representing a network is "adjacency list", or "linked list". A linked list is a structure in which objects refer to the same kind of object. For each element in the list, it contains index to previous element in the list, index to next element in the list and data. Consider the network in Figure 2.1. $L(0) = (1,2)$, indicating that the source node has two outgoing links, c_1 and c_2. $L(1) = (3,4)$, indicating that link 1 point to a node with two outgoing

links, c_3 and c_4. We use -1 to represent the sink node. The corresponding linked path structure for the undirected network is given as follows:

$$L = \{(1,2), (3,4), (5,6), (5,6), (-1), (-1), (3,4)\}.$$

Let $|V|$ and $|E|$ denote the number of nodes and links respectively. It requires $O(|V|^2)$ to store the adjacency matrix and $O(|E|)$ to store the linked list.

2.3.2 Search algorithm

Generally, searching for MPs and *d*-MPs can be regarded as constraint satisfaction problems, which aim at finding results that satisfy several constraints. The constraints are often expressed as mathematical equations and inequations.

Search tree algorithms are one class of algorithms to solve constraint satisfaction problem. There are mainly two types of tree search strategies, namely depth-first search and width-first search (or breadth-first search). The depth-first search (DFS) starts at the root node and explores as deep as possible. The breadth-first search (BFS) starts at the root node and explores all the neighboring nodes first, before moving to the next level neighbors. Let $|V|$ and $|E|$ denote the number of nodes and links in the search tree respectively. It takes $O(|V| + |E|)$ time to traverse the entire tree for both depth-first search and width-first search.

There are various heuristics to enhance the two general search strategies. Backtracking is used together with depth-first search for solving constraint satisfaction problems. It implements a depth-first search mechanism that incrementally builds candidates to the solutions and abandons each partial candidate as soon as it determines that the candidate cannot possibly be completed to a valid solution.

2.3.3 General methods for reliability evaluation

A general method for the reliability evaluation of multistate networks is using minimal path (cut) vectors, namely *d*-MPs (*d*-MCs). Given that all *d*-MPs (*d*-MCs) have been found, the issue becomes how to evaluate the probability of the union of these vectors. For example, given all *L* *d*-MPs, denoted by $\mathbf{z}^1, \mathbf{z}^2, ?\ , \mathbf{z}^L$, the reliability can be expressed as follows:

$$\Pr\left(\phi(\mathbf{x}) \geq d\right) = \Pr\left(\left\{\mathbf{x} \geq \mathbf{z}^1\right\} \cup \left\{\mathbf{x} \geq \mathbf{z}^2\right\} \cup \cdots \cup \left\{\mathbf{x} \geq \mathbf{z}^L\right\}\right), \tag{2.4}$$

where $\mathbf{x}$ is the component state vector, and $\phi(\bullet)$ is the system structure function. There are many methods for the probability evaluation of the union of *d*-MPs (*d*-MCs). In this Chapter, we introduce two classical methods that used for probability evaluation of the union of events, namely inclusion-exclusion (IE) method and sum-of-disjoint-product (SDP) method. More efficient methods for evaluating probability of the union of *d*-MPs (*d*-MCs), namely recursive sum-of-disjoint-product (RSDP) method and state space decomposition (SSD) method will be covered in Chapter 5 and Chapter 6 respectively.

Inclusion-exclusion method

IE method successively calculates upper and lower bounds of probability of the union of events by Bonferroni inequalities. It keeps including and excluding terms and eventually converges to the exact probability value. Let E_j be the event that the component state vector is greater than or equal to the *d*-MP, $\mathbf{z}^j$, i.e. $\left\{\mathbf{x} \geq \mathbf{z}^j\right\}$. Let S_k represents the sum of the probabilities that the component state vector is greater than or equal to any k *d*-MPs. It can be expressed as follows:

$$S_k = \sum_{1 \leq j_1 < \cdots j_k \leq L} \Pr\left(E_{j_1} \cap E_{j_2} \cap \cdots \cap E_{j_k}\right). \tag{2.5}$$

Then, the reliability is given as follows:

$$\Pr\left(\phi(\mathbf{x}) \geq \mathrm{d}\right) = \sum_{\mathrm{k}=1}^{\mathrm{L}} (-1)^{\mathrm{k}-1} \mathrm{S}_{\mathrm{k}}. \tag{2.6}$$

For example, there are three *d*-MPs in the network (shown in Figure 2.1) for *d* equals to 3, given as $\mathbf{z}^1=(3,2,1,0,0,1);\mathbf{z}^2=(2,2,0,0,1,1);\mathbf{z}^3=(2,1,1,0,1,2)$. The reliability of this multistate network to system demand level 3 can be calculated using IE method as follows:

$$\begin{aligned}\Pr(\phi(\mathbf{x})\geq \mathrm{d}) &= \Pr\left(\{\mathbf{x}\geq \mathbf{z}^1\}\cup\{\mathbf{x}\geq z^2\}\cup\{\mathbf{x}\geq z^3\}\right)\\ &=\Pr(\mathbf{x}\geq \mathbf{z}^1)+\Pr(\mathbf{x}\geq \mathbf{z}^2)+\Pr(\mathbf{x}\geq \mathbf{z}^3)\\ &\quad -\left[\Pr(\mathbf{x}\geq \mathbf{z}^1,\mathbf{x}\geq \mathbf{z}^2)+\Pr(\mathbf{x}\geq \mathbf{z}^1,\mathbf{x}\geq \mathbf{z}^3)+\Pr(\mathbf{x}\geq \mathbf{z}^2,\mathbf{x}\geq \mathbf{z}^3)\right]\\ &\quad +\Pr(x\geq \mathbf{z}^1,x\geq \mathbf{z}^2,x\geq \mathbf{z}^3).\end{aligned}$$

Sum-of-disjoint-product method

SDP method applies addition law of probabilities to evaluate the probability of the union of events. Let E_j be the event that the component state vector is greater than or equal to the *d*-MP, $\mathbf{z}^j$, i.e. $\{\mathbf{x}\geq \mathbf{z}^j\}$. The reliability of multistate networks to system demand level *d* is given as follows:

$$\Pr(\phi(\mathbf{x})\geq \mathrm{d})=\Pr(\mathrm{E}_1)+\Pr(\overline{\mathrm{E}}_1\mathrm{E}_2)+\Pr(\overline{\mathrm{E}}_1\overline{\mathrm{E}}_2\mathrm{E}_3)+\cdots\Pr(\overline{\mathrm{E}}_1\cdots\overline{\mathrm{E}}_{L-1}\mathrm{E}_L). \tag{2.7}$$

For example, the reliability of the multistate network (shown in Figure 2.1) to system demand level 3 can be calculated using SDP method as follows:

$$\begin{aligned}\Pr(\phi(\mathbf{x})\geq \mathrm{d}) &= \Pr\left(\{\mathbf{x}\geq \mathbf{z}^1\}\cup\{\mathbf{x}\geq \mathbf{z}^2\}\cup\{\mathbf{x}\geq \mathbf{z}^3\}\right)\\ &= \Pr\{\mathbf{x}\geq \mathbf{z}^1\}+\Pr\{\overline{\mathbf{x}\geq \mathbf{z}^1},\mathbf{x}\geq \mathbf{z}^2\}+\Pr\{\overline{\mathbf{x}\geq \mathbf{z}^1},\overline{\mathbf{x}\geq \mathbf{z}^2},\mathbf{x}\geq \mathbf{z}^3\}.\end{aligned}$$

The signs of terms for IE method change between plus and minus. The minus signs are due to double counting in the previous inclusion operations. On the other hand, all terms have the plus signs in SDP method. Both methods need to evaluate 2^L-1 terms to obtain the reliability in the worst-case scenario, where L is the number of *d*-MPs. The complexities of evaluating the reliability of multistate networks grow exponentially with the number of *d*-MPs for both IE method and SDP method.

Chapter 3

An Improved Algorithm for Finding All Minimal Paths in a Network

Recall the overall framework of the topics in this thesis, as shown in Figure 1.3. In this chapter, we focus on the first topic, which is finding all MPs. This is the first step of the indirect approaches for reliability evaluation of multistate networks. An introduction of searching for minimal paths is provided in Section 3.1. The linked path structure indexed by nodes is introduced in Section 3.2. The definition of distance between each node and the sink node is given, together with a simple algorithm for labeling the distance of each node. Based on the definition of distance, a property is proposed for additional backtracking condition. Section 3.3 presents the improved algorithm to search for all MPs, together with an illustrative example. The space and time complexity of the proposed algorithm is also analyzed. Section 3.4 compares the efficiency of proposed algorithm with that of Chen and Lin's algorithm [31] for two-terminal networks. An extension of the proposed algorithm to search for all the MPs in networks consisting of multiple source nodes and/or multiple sink nodes is discussed in Section 3.5. Section 3.6 concludes the study. Some of the results of this chapter have been documented in paper [43], and presented at the *9th International Conference on Mathematical Methods in Reliability (MMR)* [44].

3.1 Introduction

Chen and Lin's algorithm [31] is currently recognized as an efficient DFS algorithm. Chen and Lin's algorithm used a linked path structure to represent the network and conduct the search

algorithm based on the concept of backtracking. It starts with picking one outgoing link from the source node and visiting the node pointed by the link. Then it picks one outgoing link of that visited node and visit the node pointed by that link, and so on. When the sink is reached, or a cycle is detected, or no links can be found to proceed, it will backtrack to the previous node and pick another outgoing link to restart the visit. Each time it reaches the sink, one MP is found. All MPs can be found when all the links have been visited. The pseudo-code of Chen and Lin's algorithm can be found in Section 3 of their paper [31]. We focus on "Algorithm 2" in [31], which searches for all MPs only.

However, the current version of Chen and Lin's algorithm [31] contains two limitations. The first limitation is the input data structure, which is shown in Step 1 of "Algorithm 2" in reference [31]. The input data structure is the linked path structure indexed by links, which only accepts directed networks. For undirected network, a transformation is required which adds one reverse directed link for each link that does not connect to the source node. Each time a MP is found, it needs to be transformed back to the undirected form, which is shown in Step 5 of "Algorithm 2" [31]. These transformations not only consume additional computational effort themselves, but also create additional links for the network, resulting in more computational consumption. A MP can actually be represented either by a sequence of links or by a sequence of nodes. If we use a linked path structure indexed by nodes, it can accept directed network, undirected network, and mixed network. Thus, there is no need to create additional links and perform transformations, which can save a lot of computational effort. In addition, MP represented by a sequence of nodes contains both the nodes and the links (as pairs of ordered nodes), which allows both nodes and links to be failure prone in the network.

Another limitation of Chen and Lin's algorithm [31] is the lack of backtracking conditions. There are three conditions when a backtrack is triggered in the current version of Chen and Lin's algorithm [31]: 1) a cycle is detected, as shown in Step 7 of "Algorithm 2" [31]; 2) the sink node is reached, i.e. an MP is found, as shown in Step 5 of "Algorithm 2" [31]; 3) No nodes can

be found to proceed, as shown in Step 9 of "Algorithm 2" [31]. However, the three backtrack conditions do not take the structure of the network into full consideration. When all nodes that are adjacent to the sink have been visited already, it will keep visiting other nodes. It also keeps visiting other nodes when all nodes that are closer to the sink have been visited already. As pointed out in Kobayashi and Yamamoto [70], if all nodes with equal distance to the sink node have been searched under current searching branch, searching nodes with larger distance is useless. Thus, under those search branches, no MPs can be found. As the network size becomes large, more and more search branches as such are in fact useless for the purpose of searching for MPs. Thus, there is a need to introduce additional backtracking condition for Chen and Lin's algorithm to eliminate these search branches.

With the two observations above, we believe that Chen and Lin's algorithm [31] can be further improved. We first limit our discussions to two-terminal networks, and then extend to networks consisting of multiple source nodes and/or multiple sink nodes.

Notation list

d : the distance between a pair of nodes.

$\bar{c}$ *:* the number of cycles visited by the proposed algorithm.

$u / \bar{u}$: the current visiting node/ unpicked adjacent/outgoing node of the current node

$Ind / \overline{Ind}$: the set containing cardinality of each current node/updated node.

P: the minimal path set.

$W = \left\{ v_0, v_1, v_2, \cdots, v_{|E|} \right\}$: the linked path structure by links, where v_0 represents outgoing links from the source node, v_i, $i = 1,2,\cdots \left| E \right|$ represents the outgoing links from the node pointed by link i.

$L = \left\{ u_0, u_1, u_2, \cdots, u_{|V|} \right\}$: the linked path structure indexed by nodes, where u_0 represents adjacent nodes and outgoing nodes from the source node, u_i, $i = 1,2,\cdots \left| V \right|$ represents the non-source

adjacent nodes of node i if the link connected to node i is undirected, and the outgoing nodes of node i if the link connected to node i is directed.

$L_d = \left(d_0, d_1, d_2, \cdots, d_{|V|}\right)$: the distance between each node and the sink node, where d_0 is the distance between the sink node and the source node, and d_i is the distance between the sink node and node i

$Q = \left\{q_0, q_1, q_2, \cdots q_{d^{\max}}\right\}$: the set containing number of nodes with the same distance, where q_i, $i = 0,1,2,\cdots, d^{\max}$, is the number of nodes with distance $d = i$, and $d^{\max} = \max(L_d)$.

$S = \left\{s_0, s_1, s_2, \cdots, s_{d^{\max}}\right\}$: the distance checking list, where $s_i = 0$, $i = 0,1,2,\cdots, d^{\max}$, is the number of nodes with distance i that have been visited under the current search branch.

3.2 Preliminaries

In this section, we focus on two-terminal networks. The linked path structure indexed by nodes is introduced. The definition of distance between each pair of nodes is given in terms of the number of nodes, together with a simple recursive algorithm for labeling the distance. Based on the distance between each node and the sink node, a property is proposed for additional backtracking condition.

3.2.1 Linked path structure indexed by nodes

Chen and Lin [31] used linked path structure indexed by links, $W = \left\{v_0, v_1, v_2, \cdots, v_{|E|}\right\}$, to represent a network. As an example, consider the following network example given by Chen and Lin [31], as shown in Figure 3.1 (a).

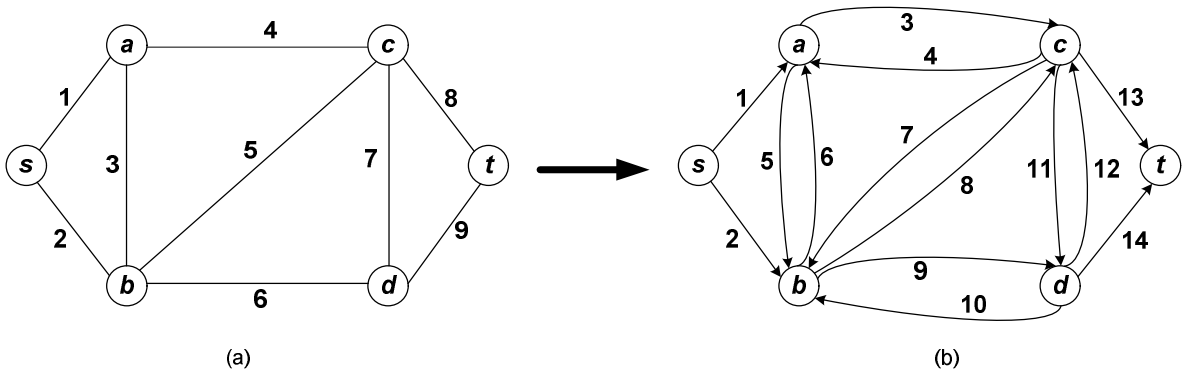

Figure 3.1: A network example: (a) the undirected form of the network; and (b) the directed form of the network [31]

Since it is an undirected network, in order to apply Chen and Lin's algorithm [31], a transformation is required to transform the current network into its directed form, as shown in Figure 3.1 (b). The corresponding linked path structure indexed by links can be given as follows:

$$W = \{(1,2),(3,5),(6,8,9),(4,7,11,13),(3,5),(6,8,9),(3,5),(6,8,9),$$
$$(4,7,11,13),(10,12,14),(6,8,9),(10,12,14),(4,7,11,13),(-1),(-1)\},$$

where $W(0)=(1,2)$ represents that the outgoing links from the source node are link 1 and link 2. $W(i)$, $i=1,2,\cdots|E|$, represents the outgoing links from the node pointed by link i. -1 is used to represent the sink node. When an MP is found, for example, $P=\{1,3,7,9,14\}$, it needs to be transformed back to the undirected form, denote as $P=\{1,4,5,6,9\}$.

In this study, we use linked path structure indexed by nodes to represent a network. Let $L=\{u_0,u_1,u_2,\cdots,u_{|V|}\}$ represent the linked path structure indexed by nodes. The linked path structure indexed by nodes can be used directly to represent the networks in their undirected, directed, and mixed forms, respectively. For example, consider the undirected network shown in Figure 1.1 (a), and for the convenience of presentation, we label the nodes by numbers, which is shown in Figure 3.2.

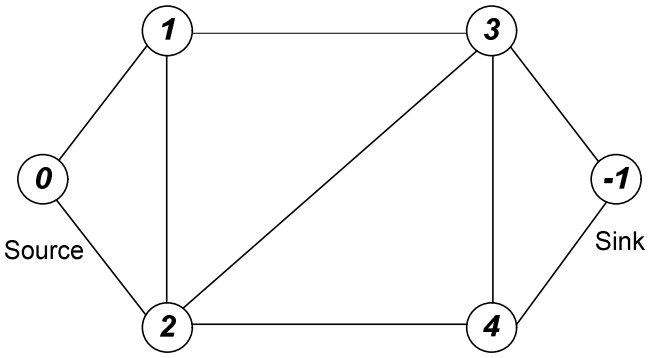

Figure 3.2: An undirected network example indexed by node

Let $L(0) = u_0 = (1,2)$, indicating that the source node has two adjacent nodes, node 1 and node 2. Let $L(1) = u_1 = (2,3)$, indicating that node 1 has two non-source adjacent nodes, node 2 and node 3. We use -1 to represent the sink node. Thus, $L(3) = u_3 = (1,2,4,-1)$, and $L(4) = u_4 = (2,3,-1)$. The corresponding linked path structure for the undirected network, shown in Figure 3.2, is given as follows:

$$L = \{(1,2),(2,3),(1,3,4),(1,2,4,-1),(2,3,-1)\}.$$

Unlike the linked path structure indexed by links, as used in [31], there is no need to transform the undirected networks to its directed forms.

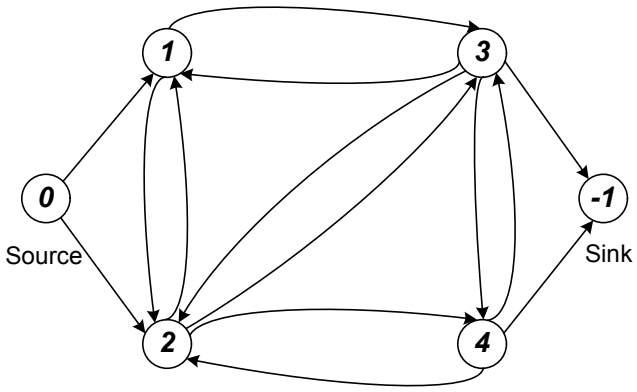

Figure 3.3: An directed network example indexed by node

For directed network, consider the directed network shown in Figure 1.1 (b), and for the convenience of presentation, we label the nodes by numbers, as shown in Figure 3.3. Let $L(0) = u_0 = (1,2)$, indicating that the source node has two outgoing nodes, node 1 and node 2. Let $L(1) = u_1 = (2,3)$, indicating that node 1 has two outgoing nodes, node 2 and node 3. Note that the linked path structure is the same as the one for undirected form of this network. Comparing to the linked path structure based on links with 15 elements, the linked path structure based on nodes is much simpler, which contains only 5 elements. Thus, the linked path structure indexed by nodes uses less storage space. In addition, for undirected and mixed networks, the linked path structure indexed by nodes can save extra computational efforts since it does not create additional links and no transformation is required when each MP is found.

3.2.2 The distance between node pair

The distance is defined as the minimal number of edges in some paths between any two terminal nodes in [70], and there is no discussion on methods of labeling the distance in [70]. In this work, we define the distance between a pair of nodes based on number of nodes and provide a simple a simple recursive algorithm to label the distance.

First, we define the distance between any pair of nodes as follows.

Definition 1 *The distance between a pair of nodes is the minimum number of nodes for one node to reach the other node.*

Consider the network in Figure 3.2. There are 5 nodes, namely, 0, 1, 2, 3, and 4, and one sink node -1. We want to find the distance between the sink node and other nodes. Take node 1 as an example. It can reach node -1 via 1 node, which is node 3; or 2 nodes, which are node 2 and node 4; or 3 nodes, which are node 2, node 3 and node 4. The distance is the minimum number of nodes required, which is 1. The distance between source 0, node 1, node 2, node 3, node 4 and the sink node is 2, 1, 1, 0, and 0, respectively, in this example.

With *Definition 1*, one can apply the well-known Dijkstra's Algorithm [56], which finds the shortest path between one node and every other node in polynomial time $O\left(|V|^2\right)$, where $|V|$ is the number of nodes. The length between each pair of adjacent nodes is 1 in our network. Then the distance is the length of the shortest path. In this paper, given the linked path structure indexed by node, we apply the main idea of Dijkstra's Algorithm and give a simple recursive algorithm, namely *distance*, for labeling all the distances between the sink node and other nodes.

step 1 Input the linked path structure based on nodes of the network, L. Obtain the number of elements in L, denote as S. Create a set, $Ind=\{-1\}$, to store the current node, where it contains only the sink node -1 currently; another set, $\overline{Ind}=\phi$ for updating Ind, and an array of size S to store the distances for elements in L, denote as L_d, where $L_d=\left(d_0,d_1,d_2,\cdots,d_{|V|-2}\right)$. Set $d_i=0$, where $i=0,1,2,\cdots,|V|-2$. Set the current distance, $d=0$.

step 2 For each element in L, if it contains any node in Ind, label the distance of the element as $d_i=d$. Store the cardinality of the element to $\overline{Ind}=\phi$. Empty this element.

step 3 If all elements in L are empty, stop and the labeling process is finished.

Otherwise, let $Ind=\overline{Ind}$, $d=d+1$, and go to step 2.

We illustrate the algorithm *distance* using the network example in Figure 3.2.

1. Input the linked path structure as follows:

$$L=\{(1,2),(2,3),(1,3,4),(1,2,4,-1),(2,3,-1)\}.$$

There are 5 nodes, namely, 0, 1, 2, 3 and 4, and one sink node -1. Obtain the number of elements, $S=5$. Create the set $Ind=\{-1\}$, the set $\overline{Ind}=\phi$, and L_d, where $L_d=\left(d_0,d_1,d_2,d_3,d_4\right)$. Set $d_i=0$, where $i=0,1,2,3,4$. Set the current distance, $d=0$.

2. By checking each element of L, $L(3)=(1,2,4,-1)$ and $L(4)=(2,3,-1)$ contain -1. Label their distances as $d_3=d_4=d=0$, and store their cardinalities as $\overline{Ind}=\{3,4\}$. Empty these elements and *L* becomes:

$$L=\{(1,2),(2,3),(1,3,4),(\ ),(\ )\}.$$

3. Since there are non-empty elements in L, let $Ind=\overline{Ind}$, $d=d+1=1$, and go to step 2.

4. By checking each element of L, $L(1)=(2,3)$, and $L(2)=(1,3,4)$ contain 3, and $L(2)=(1,3,4)$ contains 4. Label the distances as $d_1=d_2=d=0$, and store the cardinalities of these element to $\overline{Ind}=\{1,2\}$. Empty these elements and *L* becomes:

$$L=\{(1,2),(\ ),(\ ),(\ ),(\ )\}.$$

5. Since there is non-empty element in L, let $Ind=\overline{Ind}$, $d=d+1=2$, and go to step 2.

6. By checking each element, $L(1)=(1,2)$ contains1 and 2 in Ind. Label the distance as $d_0=2$, and store the cardinalities of these element to $\overline{Ind}=\{0\}$. Empty these elements and *L* becomes:

$$L=\{(\ ),(\ ),(\ ),(\ ),(\ )\}.$$

7. Since all elements in L are empty, the labeling process is finished. We have the distance as follows:

$$L_d=(2,1,1,0,0).$$

3.2.3 Additional backtracking condition

Based on the distance between the sink node and other nodes, a property for additional backtracking condition is given as follows.

Property 1 *during the search process, if all the nodes with distance, k, have been visited, and the current visiting node has distance larger than k, no MPs can be found under this search branch.*

When $k = 0$, all the nodes with distance 0 have been visited, i.e. all the nodes that are adjacent to the sink node have been visited already. If the current visiting node has distance larger than 0, it can only reach the sink node by first visiting at least one of nodes that are adjacent to the sink node again. However, by visiting a node with distance 0 twice, a cycle is formed which will trigger the backtracking process (backtrack to the previous node). Thus, it will never reach the sink node and no MPs can be found under this search branch.

When $k = 1$, all the nodes with distance 1 have been visited. If the current visiting node has distance larger than 1, it can only reach the sink node by first visiting at least one of nodes with distance 1 and then visiting at least one of nodes that are adjacent to the sink node. However, by visiting a node with distance 1 twice, a cycle is formed which will trigger the backtracking process. Thus, it will never reach the sink node and no MPs can be found under this search branch.

When $k > 1$, all the nodes with distance *k* have been visited. If the current visiting node has distance larger than *k*, it can only reach the sink node by first visiting at least one of nodes with distance *k*, then visiting at least one of nodes with distance $k - 1$,…, and finally visiting at least one of nodes with distance 0. However, by visiting a node with distance *k* twice, a cycle is formed which will trigger the backtracking process. Thus, it will never reach the sink node and no MPs can be found under this search branch.

Consider the following example shown in Figure 3.4, which is the implementation of Chen and Lin's algorithm [31] on the network, shown in Figure 3.1 (b). One visiting path goes as follows. $(1,2)$→$(3,5)$→$(4,7,11,13)$→$(10,12,14)$. When it reaches $(10, 12, 14)$, all nodes that are adjacent to the sink node, $(4, 7, 11, 13)$ and $(10, 12, 14)$ have been visited. Because all nodes

with distance 0 have been visited, visiting node with distance large than or equal to 0 is useless. As can be seen, the search branch contained in the dashed frame has no MPs found, i.e. cutting the outgoing branches 10 and 12 from $(10, 12, 14)$ would not affect the MPs generated.

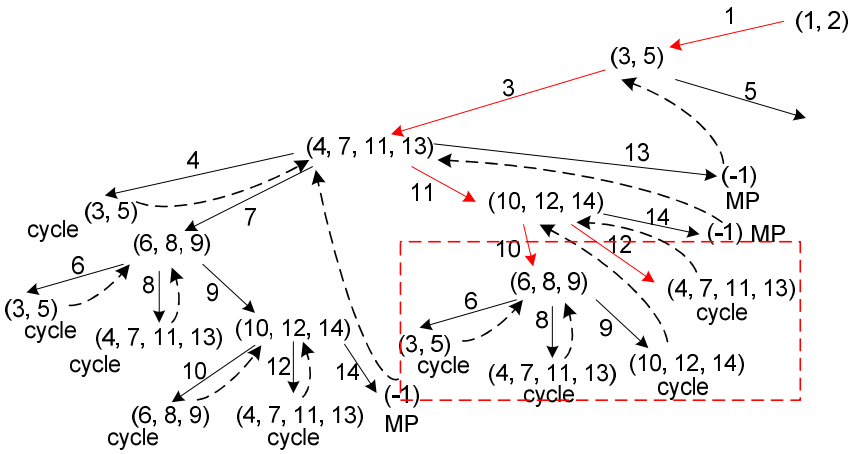

Figure 3.4: Implementation of Chen and Lin's algorithm

3.3 The proposed algorithm

In this section, the improved algorithm to search for MPs is presented for two-terminal networks. A simple example is given to illustrate the steps of the algorithm. At last, the space and time complexity analysis is presented at the end of the section.

3.3.1 The algorithm

With Property 1 and Definition 1, we propose the following algorithm to search for all minimal paths. The detailed procedure of the algorithm is given as follows.

step 1 Input the linked path structure indexed by nodes of the network, L. Implement the algorithm *distance* to label the distance between each node and sink node, denoted as L_d. Obtain the numbers of nodes with the same distance, $Q = \{q_0, q_1, q_2, \cdots q_{d^{\max}}\}$. q_i, $i = 0,1,2,\cdots,d^{\max}$, is the number of nodes with distance $d = i$, and $d^{\max} = \max(L_d)$.

step 2 Initialize the current visiting node, $u = 0$, the set of minimal paths, $P = \phi$, the distance checking list, $S = \left\{ s_0, s_1, s_2, \cdots, s_{d^{\max}} \right\}$, where $s_i = 0$, $i = 0,1,2,\cdots, d^{\max}$, is the number of nodes with distance i that have been visited under the current search branch. Go to step 3.

step 3 If all the adjacent/outgoing nodes of the current visiting node u have been picked, check the following condition:

If $u = 0$, stop the search; otherwise, backtrack to the previous node.

Otherwise, select one unpicked adjacent/outgoing node of the current node, $\bar{u} \in L(u)$, and go to step 4.

step 4 Check the following conditions for the selected adjacent/outgoing node $\bar{u}$,

step 4.1 If $\bar{u} = -1$, the adjacent/outgoing node is the sink node and an MP is found. Output P and *backtrack* to the previous node. Go to step 3.

step 4.2 If $\bar{u} \in P$, the adjacent/outgoing node has been visited under the current search branch and a cycle is detected. *Backtrack* to the previous node. Go to step 3.

step 4.3 If $s_k = q_k$ and $L_d(\bar{u}) \geq k$, all nodes with distance k have been visited and the current adjacent/outgoing node has distance larger than or equal to k. *Backtrack* to the previous node; Go to step 3.

step 4.4 If the adjacent/outgoing node doesn't satisfy any of the conditions above, it is included in the current MP set, $P = \left\{ \bar{u} \right\} \cup P$, and becomes the next visiting node, $u = \bar{u}$. Update the distance checking list, $s_i = s_i + 1$, $i = L_d(\bar{u})$. Go to step 3.

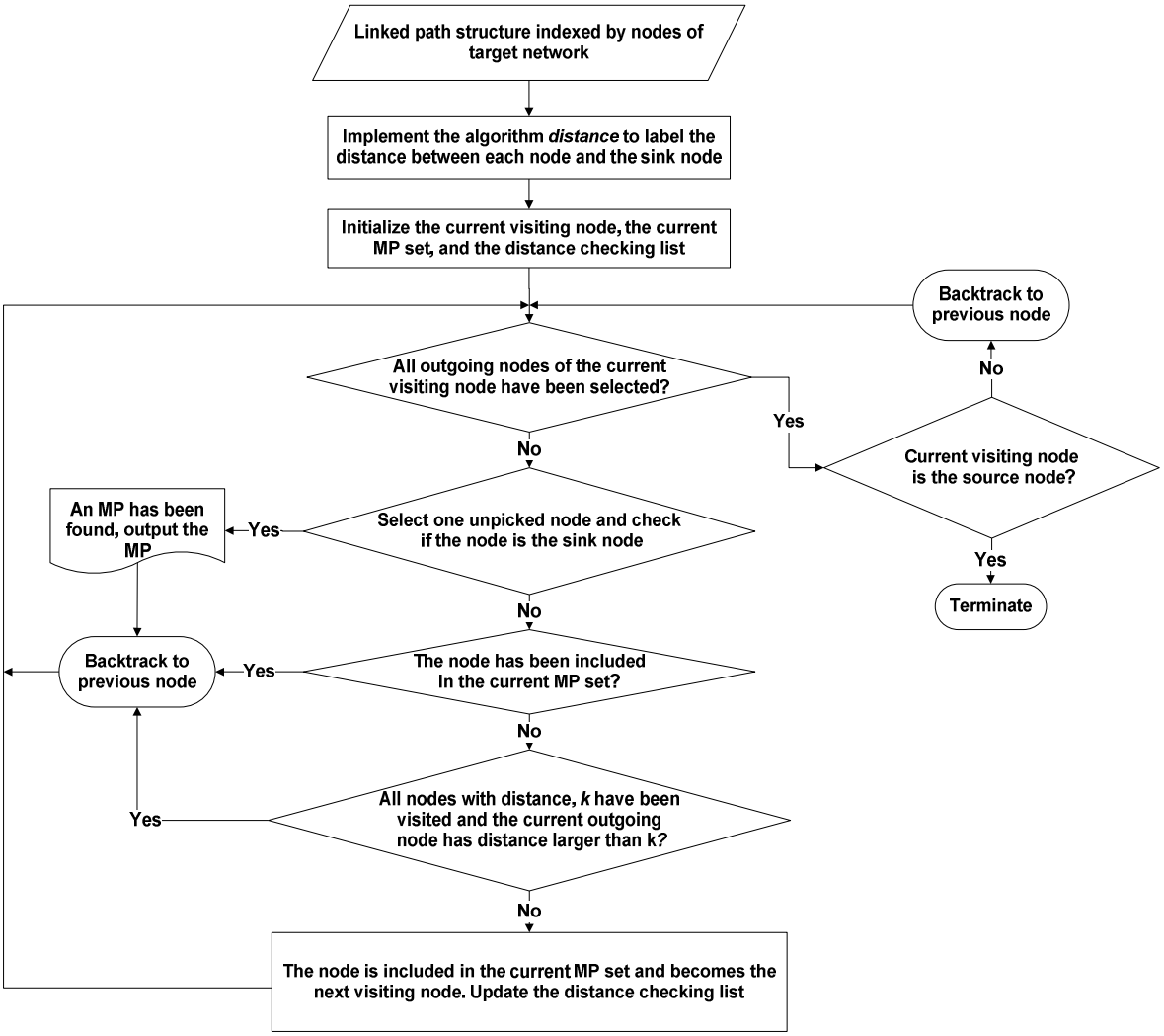

Figure 3.5: Flow chart of proposed algorithm

As can be seen, the proposed algorithm first employs the recursive algorithm in Section 3.2.2 to label the distance between each node and the sink. Then it starts to visit one of adjacent/outgoing nodes of the source node, and proceeds to visit one of adjacent/outgoing nodes of the visited node, and so on. It backtracks to the previous level and picks up another adjacent/outgoing node of the visited node to restart the search when one of the following conditions is met:

1) The sink node has been reached [31];

2) A cycle is detected [31];

3) All the nodes with distance, *k*, have been visited, and the current visiting node has distance larger than *k*;
4) No nodes can be found to proceed [31].

Each time it reaches the sink, one MP is found. All MPs can be found when the whole process is exhausted. Note that backtracking condition (3) is newly added comparing to Chen and Lin's algorithm [31]. The flow chart of the proposed algorithm is given in Figure 3.5.

3.3.2 An illustrative example

Consider the network example shown in Figure 3.2 in Section 3.2.

step 1 Input the linked path structure of the network as follows:

$$L=\{(1,2),(2,3),(1,3,4),(1,2,4,-1),(2,3,-1)\}.$$

Implement the algorithm *distance* to label the distance between each node and the sink node as $L_d=(2,1,1,0,0)$. Obtain the numbers of nodes with the same distance, $Q=\{2,2,1\}$, where $Q(0)=q_0=2$ represents that the number of nodes with distance 0 is 2. Obtain the maximum distance between all the nodes and the sink node, which is, $d^{\max}=\max(L_d)=2$.

step 2 Initialize the current visiting node, $u=0$, the set of minimal paths, $P=\phi$, the distance checking , $S=\{s_0,s_1,s_2\}$. Set $s_i=0$, $i=0,1,2$.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=0$, select one unpicked outgoing node of the current node, $\bar{u}=1\in L(0)$, and go to step 4.

step 4 Since $\bar{u}\neq -1$, $v\notin P$ and $s_i\neq q_i$ for all $i=0,1,2$, it is included in the current MP set, $P=\{\bar{u}\}\cup P=\{1\}$, and becomes the next visiting node, $u=\bar{u}=1$. Update the distance checking list, $s_i=s_i+1=1$, $i=L_d(\bar{u})=1$, $S=\{0,1,0\}$. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=1$, select one unpicked outgoing node of the current node, $\bar{u}=2\in L(1)$, and go to step 4.

step 4 Since $\bar{u}\neq -1$, $v\notin P$ and $s_i\neq q_i$ for all $i=0,1,2$, it is included in the current MP set, $P=\{\bar{u}\}\cup P=\{1,2\}$, and becomes the next visiting node, $u=\bar{u}=2$. Update the distance checking list, $s_i=s_i+1=2$, $i=L_d(\bar{u})=1$,and $S=\{0,2,0\}$. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=2$, select one unpicked outgoing node of the current node $\bar{u}=1\in L(2)$, and go to step 4.

step 4 Since $\bar{u}=1\in P$, the outgoing node has been visited under the current search branch and a cycle has been detected, backtrack to the previous node. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=2$, select one unpicked outgoing node of the current node $\bar{u}=3\in L(2)$, and go to step 4.

step4 Since $\bar{u}\neq -1$, $v\notin P$ and $L_d(\bar{u})=0\le k=1$, for $s_1=q_1$,it is included in the current MP set, $P=\{\bar{u}\}\cup P=\{1,2,3\}$, and becomes the next visiting node, $u=\bar{u}=3$. Update the distance checking list, $s_0=s_0+1=1$, $i=L_d(\bar{u})=0$, and $S=\{1,2,0\}$. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=3$, select one unpicked outgoing node of the current node, $\bar{u}=1\in L(3)$, and go to step 4.

step 4 Since $\bar{u}=1\in P$, the outgoing node has been visited under the current search branch and a cycle has been detected, backtrack to the previous node 3. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=3$, select one unpicked outgoing node of the current node $\bar{u}=2\in L(3)$, and go to step 4.

step 4 Since $\bar{u}=2\in P$, the outgoing node has been visited under the current search branch and a cycle has been detected, backtrack to the previous node. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=3$, select one unpicked outgoing node of the current node $\bar{u}=4\in L(3)$, and go to step 4.

step 4 Since $\bar{u}\neq -1$, $v\notin P$ and $L_d\left(\bar{u}\right)=0\le k=1$, for $s_1=q_1$, it is included in the current MP set, $P=\left\{\bar{u}\right\}\cup P=\left\{1,2,3,4\right\}$, and becomes the next visiting node, $u=\bar{u}=4$. Update the distance checking list, $s_0=s_0+1=2$, $i=L_d(\bar{u})=0$, and $S=\left\{2,2,0\right\}$. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=4$, select one unpicked outgoing node of the current node $\bar{u}=2\in L(4)$, and go to step 4.

step 4 Since $\bar{u}=2\in P$, the outgoing node has been visited under the current search branch and a cycle is detected, backtrack to the previous node. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=4$, select one unpicked outgoing node of the current node $\bar{u}=3\in L(4)$, and go to step 4.

step 4 Since $\bar{u}=3\in P$, the outgoing node has been visited under the current search branch and a cycle has been detected, backtrack to the previous node. Go to step 3.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=4$, select one unpicked outgoing node of the current node $\bar{u}=-1\in L(4)$, and go to step 4.

step 4 Since $\bar{u}=-1$, the outgoing node is the sink node and an MP has been found, output P, $P=\left\{1,2,3,4\right\}$. Backtrack to the previous node and go to step 3.

step3 Since all outgoing nodes have been selected for the current node $u=4$ and $u\neq 0$, backtrack to the previous node, $u=3$. After backtracking, the current MP is updated, $P=\left\{1,2,3\right\}$, and the current distance checking list is updated, $s_0=s_0-1=1$, and $S=\left\{1,2,0\right\}$.

step 3 Since there exist unselected outgoing nodes for the current visiting node $u=3$, select one unpicked outgoing node of the current node $\bar{u}=-1\in L(3)$, and go to step 4.

step 4 Since $\bar{u} = -1$, the outgoing node is the sink node and an MP has been found, output P, $P = \{1,2,3\}$. Backtrack to the previous node and go to step 3.

step 3 Since all the outgoing nodes have been selected for the current node $u = 3$ and $u \neq 0$, backtrack to the previous node, $u = 2$. After backtracking, the current MP is updated, $P = \{1,2\}$, and the current distance checking list is updated, $s_0 = s_0 - 1 = 0$, and $S = \{0,2,0\}$. Note that after backtracking, node 4 becomes unvisited under the new search branch.

Continue the procedure as illustrated above, as described in the flowchart in Fig. 4. In the end, all the MPs are found as follows,

$\{0,1,2,3,4,-1\}$, $\{0,1,2,3,-1\}$, $\{0,1,2,4,3,-1\}$, $\{0,1,2,4,-1\}$, $\{0,1,3,2,4,-1\}$, $\{0,1,3,4,-1\}$, $\{0,1,3,-1\}$,

$\{0,2,1,3,4,-1\}$, $\{0,2,3,4,-1\}$, $\{0,2,4,3,-1\}$, $\{0,2,4,-1\}$, $\{0,2,3,-1\}$, and $\{0,2,1,3,-1\}$.

The result agrees with the result in Chen and Lin [31], which partially verifies the proposed algorithm.

3.3.3 Complexity analysis

Let E denote the set of links, and V denote the set of nodes. The storage complexity of the algorithm is $O\left(4\left(|V|-1\right)\right)$, with respect to one input list L ($|V|-1$ in length), one path buffer P ($|L|$ in the worst case), one distance list Q ($|V|-1$ in the worst case), and one distance checking list S ($|V|-1$ in the worst case). The storage complexity of Chen and Lin's algorithm (algorithm 2) [31] is $O\left(3\left(|E|+1\right)+|V|\right)$, with respect to one input list W ($|E|+1$ in length), one path buffer P ($|W|$ in the worst case), one working list ($|E|+1$ in length), and one node path ($|V|$ in the worst case). It can be seen that the storage complexity of the proposed algorithm is less than Chen

and Lin's algorithm, because $|V| \leq |E| - 1$. For many real world networks and undirected networks, $|V| \ll |E|$.

Chen and Lin [31] claimed that the time complexity of their algorithm (algorithm 2), is $O(\lambda \cdot \pi)$, where λ denotes the average number of links for each MP, and π denotes the total number of MPs. Although Chen and Lin's algorithm does not search for cycles, it still visits all the cycles. Thus, the time complexity of Chen and Lin's algorithm is still affected by number of cycles and should be $O(\lambda \cdot (\pi + c))$, where c denotes the total number of cycles contained in the network. For the proposed algorithm, the time complexity of labeling the distance is bounded by polynomial time $O(|V|^2)$, where $|V|$ is the number of nodes [56]. As the process of searching for all MPs is NP-hard, the time complexity of labeling the distance is dominated. Let η denote the average number of nodes in a MP, and $\bar{c}$ denote the number of cycles visited by proposed algorithm, the time complexity of the algorithm is $O(\eta \cdot (\pi + \bar{c}))$. With the newly added backtracking condition. Many search branches that end in cycles are eliminated. It can be seen that our proposed algorithm is more efficient because $\bar{c} \ll c$. In addition, for undirected network, our improved algorithm can save extra computational efforts since it does not create additional links and no transformation is required when each MP is found.

3.4 Efficiency investigation

In this section, we investigate the efficiency of the proposed algorithm for finding all MPs in two-terminal networks. We compare the efficiency of the proposed algorithm with that of Chen and Lin's algorithm (algorithm 2) [31] for finding all the MPs in two-terminal networks. In terms of efficiency of the two algorithms, we are interested in the required CPU time with respect to different networks. Both algorithms are programmed in Matlab 2014a, and were implemented on

a personal computer with Intel Core i7 3.6GHz CPU, and 16 GB of RAM. We also record the ratio, which is defined as the CPU time of Chen and Lin's algorithm divided by the proposed algorithm. The ratio indicates the advantage of the proposed algorithm over Chen and Lin's algorithm.

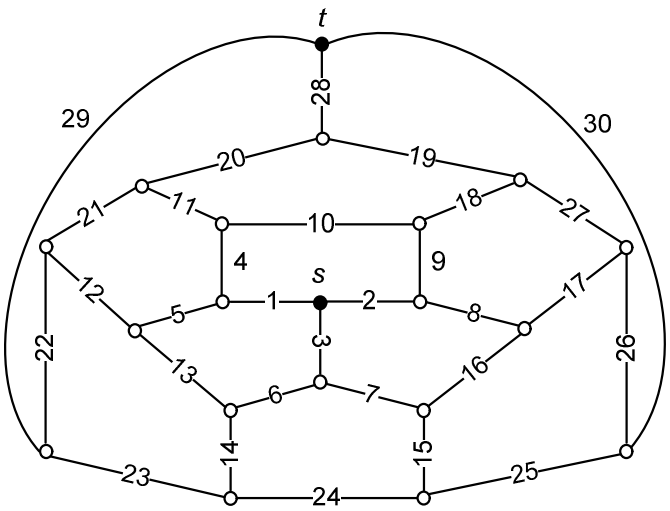

Figure 3.6: The benchmark network

Firstly, we consider a benchmark network given by Luo and Trivedi [57], shown in Figure 3.6. The number of MPs found by the proposed algorithm is 780, which agrees the result in Chen and Lin [31]. The CPU times for Chen and Lin's algorithm and the proposed algorithm are 2.7367 seconds and 1.511 seconds, respectively. The ratio is about 1.8111, indicating the proposed algorithm is 1.8 times faster than Chen and Lin's algorithm in finding all the MPs of the benchmark network.

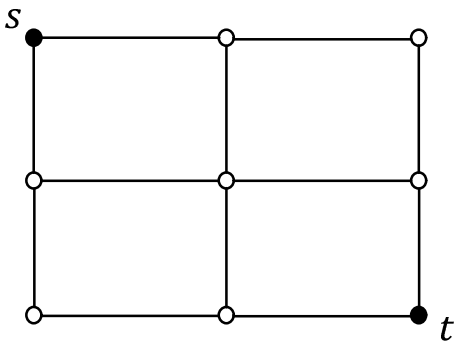

Figure 3.7: A typical undirected grid network with 9 nodes

Table 3.1: The efficiency comparison for two-terminal grid networks

Networks (Nodes)	T_1: CPU Time by Chen and Lin's algorithm [31] (sec.)	T_2: CPU Time by proposed algorithm (sec.)
2 × 2	0.0264	0.0275
3 × 3	0.0307	0.0294
4 × 4	0.1743	0.1339
5 × 5	13.7148	6.8459
6 × 6	3717.3675	1153.7572
7 × 7	2714114.1393	610070.9546

Secondly, we adopt the two terminal grid networks used in Chen and Lin [31] for efficiency investigation. Figure 3.7 shows atypical two terminal grid network with 9 nodes. Both the proposed algorithm, and Chen and Lin's algorithm [31] generate the same sets of MPs for all the grid networks considered**.** As can be seen from Table 3.1, the CPU time of the proposed algorithm is greater than that of Chen and Lin's algorithm for the two terminal grid network with 4 nodes. However, as the network size grows, the CPU time of the proposed algorithm is less

than Chen and Lin's algorithm for two terminal grid networks. In addition, as the network size grows, the ratio increases, indicating that the efficiency of the proposed algorithm over Chen and Lin's algorithm becomes more advantageous. Figure 3.8 shows the trend of ratio as the network size grows.

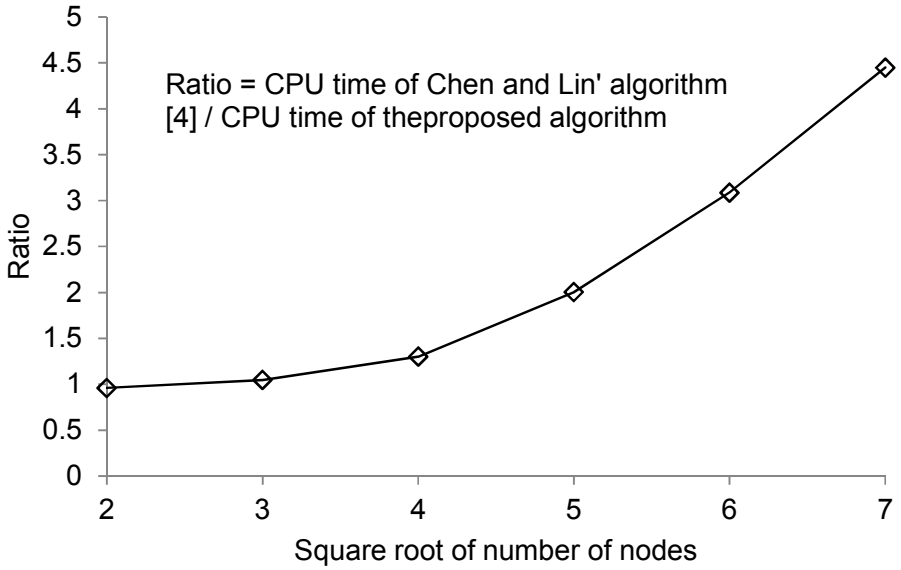

Figure 3.8: Ratio of CPU time with respect to different sizes of grid networks

3.5 Extension to networks with multiple source/sink nodes

The proposed algorithm in Section 3.3 is limited to two terminal networks, which consist of only one source node and one sink node. However, there are many real-world networks which consist of multiple source nodes and/or multiple sink nodes. For a binary network with one source node and multiple sink nodes, the network is said to be operative if there exist operating paths from the source node to each sink node [5]. For a network with multiple source nodes and multiple sink nodes, the network is said to be operative if there exist operating paths for all source-sink pairs, concurrently [31]. One way to evaluate the reliability of such networks is based on all the minimal paths for each source-sink pair. Note that a minimal path for each source-sink pair is not a MP for the network. Thus, given the source node, s , and the sink node,

t, a minimal path for this source-sink pair is denoted by MP_{st}. To find all MP_{st} for each source-sink node pair, we have to call the proposed algorithm in Section 3.3 multiple times because the distance for each non-sink node may be different for different source-sink node pairs. To address this limitation, we apply the classical network transformation technique in [5] to transform the networks with multiple source nodes and/or sink nodes to two-terminal networks and then apply the proposed algorithm to find all MPs for each source-sink pair. This is done as follows.

step 1 Add two artificial nodes, with one acting as a source node and the other as a sink node;

step 2 Add one artificial link between each source node and the artificial source node; and add one artificial link between each sink node and the artificial sink node. Thus, the network is converted to a two-terminal network.

step 3 Implement the proposed algorithm in Section 3.3 to find all MPs for the converted two-terminal network.

step 4 Remove the artificial source node and sink node in the MPs obtained in step 3. Obtain all the MP_{st} for each source-sink pair by collecting those MPs with the specific source node as the first node and the specific sink node as the last node.

Consider an undirected network shown on the left side of Figure 3.9. The corresponding directed form of the network is given in Figure 3.10 on the left hand side. The network has one source node and two sink nodes.

By adding an artificial sink node and 2 artificial links, the converted two-terminal network is shown on the right side of Figure 3.9. The corresponding directed network is shown on the right hand side of Figure 3.10. The corresponding linked path structure of the transformed network is given as follows:

$$L = \{(1,2),(2,4),(1,3,4),(2,4,-1),(1,2,3,-1)\}.$$

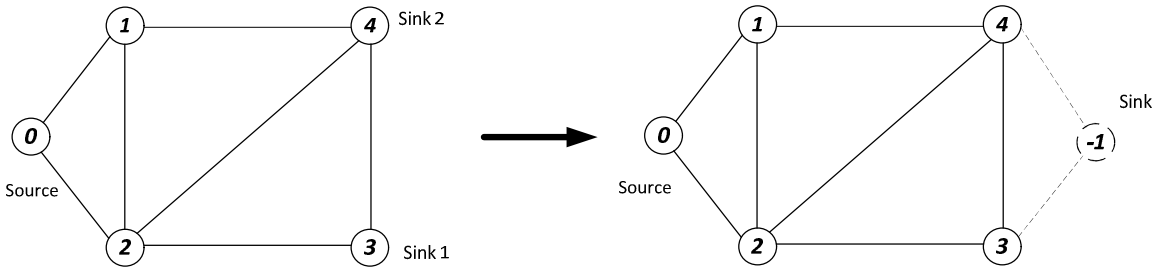

Figure 3.9: An example of undirected network with a source node and two sink nodes

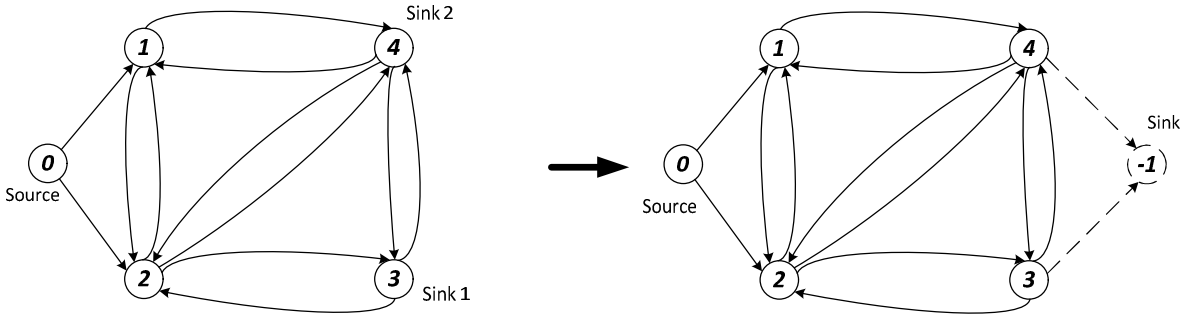

Figure 3.10: An example of directed network with a source node and two sink nodes

Note that the converted two-terminal network is the same as the network in Figure 3.2 in Section 3.2. Based on the numerical example in Section 3.3, all the MPs for the two-terminal networks are given as follows.

$\{0,1,2,3,4,-1\}$, $\{0,1,2,3,-1\}$, $\{0,1,2,4,3,-1\}$, $\{0,1,2,4,-1\}$, $\{0,1,3,2,4,-1\}$, $\{0,1,3,4,-1\}$, $\{0,1,3,-1\}$,

$\{0,2,1,3,4,-1\}$, $\{0,2,3,4,-1\}$, $\{0,2,4,3,-1\}$, $\{0,2,4,-1\}$, $\{0,2,3,-1\}$, and $\{0,2,1,3,-1\}$.

By removing the artificial sink node, -1, we can obtain all the MPs for each source-sink pair. All MPs from source node 0 to sink node 3 can be obtained by collecting MPs with node 3 as the last node, as follows.

$$\{0,1,2,3\},\ \{0,1,2,4,3\},\{0,1,3\},\{0,2,4,3\},\{0,2,3\}, \text{ and } \{0,2,1,3\}.$$

All MPs from source node 0 to sink node 4 can be obtained by collecting MPs with node 4 as the last node, as follows.

$$\{0,1,2,3,4\},\ \{0,1,2,4\},\ \{0,1,3,2,4\},\ \{0,1,3,4\},\{0,2,1,3,4\},\ \{0,2,3,4\}, \text{ and } \{0,2,4\}.$$

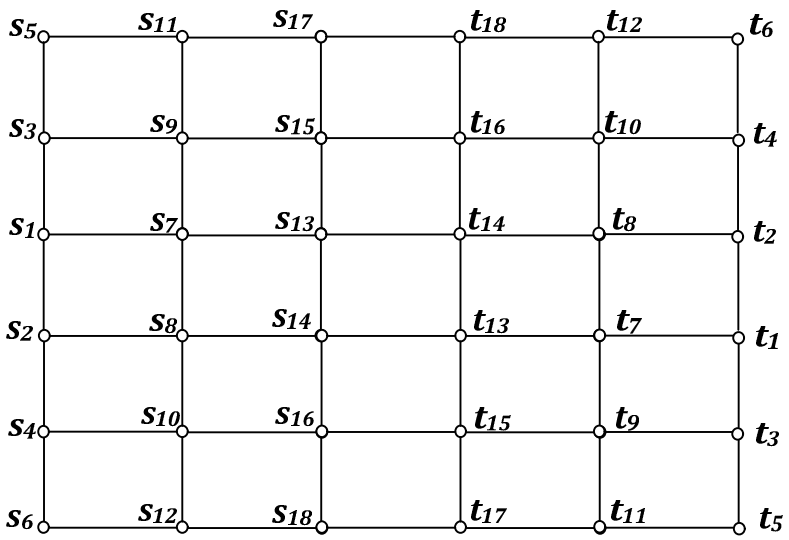

Figure 3.11: A 6 × 6 multi-terminal grid networks

We compare the efficiency of the proposed algorithm in searching for all MPs for networks with multiple source nodes and/or multiple sink nodes with that of Chen and Lin's algorithm [31]. We adopt the 6 × 6 grid network for the efficiency investigation, which is shown in Figure 3.11.

3.5.1 Networks with one source and multiple sinks

First, we consider a network with one source node and multiple sink nodes. We fix the source node s_1, and add two sink nodes t_1 and t_2. We add sink node one at a time until we have 6 sink nodes. We also consider the scenarios with 12 and 18 sinks nodes, respectively. Figure 3.12 shows the trend of ratio as the number of sink nodes grows. As can be seen from the results, the CPU time of the proposed algorithm is less than that of Chen and Lin's algorithm for all scenarios. In addition, as the number of sink nodes increases, the ratio decreases.

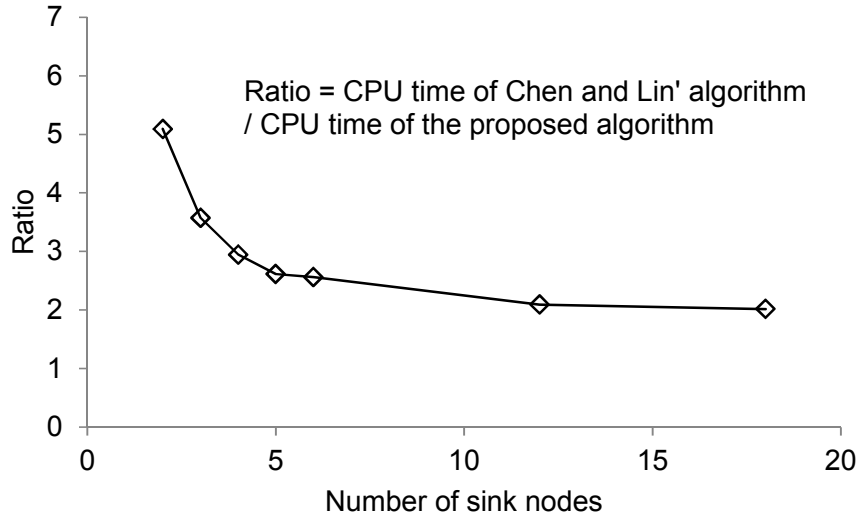

Figure 3.12: Ratio of CPU time for networks with one source node and multiple sink nodes

Table 3.2: The distances for 6 × 6 grid networks with one source node and multiple sink nodes

Number of sources and sinks	Number of nodes having same distance
1 sources & 2 sinks	$Q=\{2,4,6,6,6,6,5,2\}$
1 sources & 3 sinks	$Q=\{3,5,6,6,6,6,4,1\}$
1 sources & 4 sinks	$Q=\{4,6,6,6,6,6,3\}$
1 sources & 5 sinks	$Q=\{5,6,6,6,6,6,2\}$
1 sources & 6 sinks	$Q=\{6,6,6,6,6,6,1\}$
1 sources & 12 sinks	$Q=\{12,6,6,6,6,1\}$
1 sources & 18 sinks	$Q=\{18,6,6,6,1\}$

* $Q=\{q_0,q_1,q_2,\cdots q_{d^{\max}}\}$, where q_i is the number of nodes with distance i

This can be explained as follows. Recall that one advantage of the proposed algorithm is due to the inclusion of the additional backtracking condition based on *Property 1* in Section 3.2.3. The maximum distance decreases as the number of terminals increases, as shown in

Table 3.2. In addition, as the number of sinks increases, the number of nodes with shorter distance increases, while the number of nodes with longer distance decreases. Thus, as the number of sinks increases, the search branches reduced by additional backtracking condition decrease. As a result, the advantage of proposed algorithm over Chen and Lin's algorithm decreases as more nodes become sink nodes. However, another advantage of proposed algorithm, which uses linked path structure indexed by node to represent the network, is still in effect. Thus, despite the number of sink nodes, the proposed algorithm is still more efficient than Chen and Lin's algorithm [31].

3.5.2 Networks with one sink and multiple sources

Second, we consider a network with one sink node and multiple source nodes. We fix the sink node t_1, and add two source nodes s_1 and s_2. We add source nodes one at a time until we have 6 source nodes. We also consider the scenarios with 12 and 18 source nodes respectively. Figure 3.13 shows the trend of ratio as the number of sink nodes grows. As can be seen from the results, the CPU time of proposed algorithm is less than that of Chen and Lin's algorithm for all scenarios. In addition, as the number of sink nodes increases, the ratio remains almost the same.

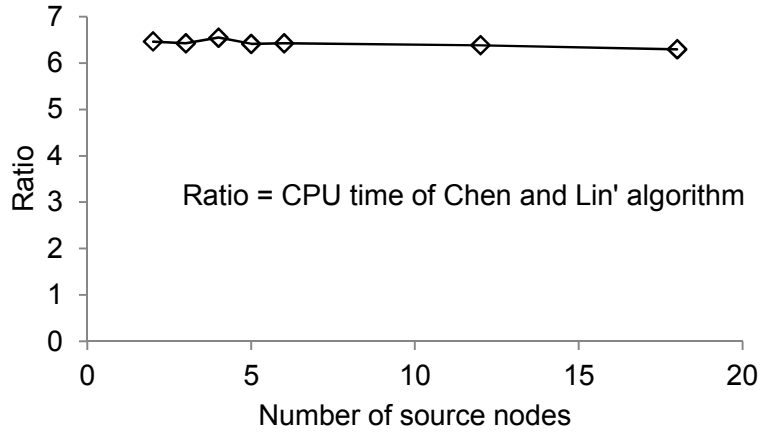

Figure 3.13: Ratio of CPU time for networks with one sink node and multiple source nodes

This can be explained as follows. The distance between each node and the sink node is the same for each scenario, as shown in Table 3.3. There is only a slight difference in scenarios with 12 and 18 source nodes. As a result, the advantage of the proposed algorithm over Chen and Lin's algorithm is the almost the same for network with one sink node and multiple source nodes.

Table 3.3: The distances for 6 × 6 grid networks with one sink node and multiple source nodes

Number of sources and sinks	Number of nodes having same distance
2 sources & 1 sinks	$Q=\{1,3,5,6,6,6,6,3,1\}$
3 sources & 1 sinks	$Q=\{1,3,5,6,6,6,6,3,1\}$
4 sources & 1 sinks	$Q=\{1,3,5,6,6,6,6,3,1\}$
5 sources & 1 sinks	$Q=\{1,3,5,6,6,6,6,3,1\}$
6 sources & 1 sinks	$Q=\{1,3,5,6,6,6,6,3,1\}$
12 sources & 1 sinks	$Q=\{1,3,5,6,6,7,5,3,1\}$
18 sources & 1 sinks	$Q=\{1,3,5,6,7,6,5,3,1\}$

* $Q=\{q_0,q_1,q_2,\cdots q_{d^{\max}}\}$, where q_i is the number of nodes with distance i

3.5.3 Networks with multiple sources and sinks

Finally, we consider a network with multiple source nodes and multiple sink nodes. We start with 2 source nodes, s_1 and s_2, and 2 sink nodes, t_1 and t_2. Then we consider a network with one additional source node and one additional sink node, s_3 and t_3. We keep adding one pair of source node and sink node at a time until we have 6 source nodes and 6 sink nodes. Finally, we consider a 6 × 6 grid network with 12 source nodes and 12 sink nodes, and a 6 × 6 grid network with 18 source nodes and 18 sink nodes respectively. Figure 3.14 shows the trend of the ratio as the number of nodes grows. As can be seen from the results, the CPU time of the proposed

algorithm is less than that of Chen and Lin's algorithm for all scenarios. In addition, as the number of sink nodes increases, the ratio decreases.

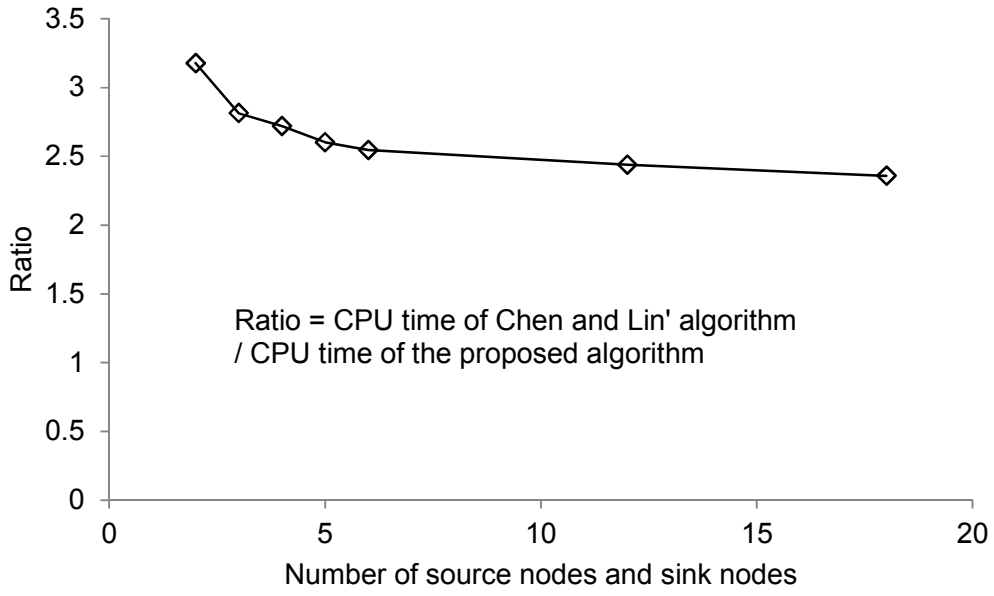

Figure 3.14: Ratio of CPU time for networks with multiple source/sink nodes

Table 3.4: The distance information for 6 × 6 multi-terminal grid networks

Number of sources and sinks	Number of nodes having same distance
2 sources & 2 sinks	$Q=\{2,4,6,6,6,6,5,2\}$
3 sources & 3 sinks	$Q=\{3,5,6,6,6,6,4,1\}$
4 sources & 4 sinks	$Q=\{4,6,6,6,6,6,3\}$
5 sources & 5 sinks	$Q=\{5,6,6,6,6,6,2\}$
6 sources & 6 sinks	$Q=\{6,6,6,6,6,6,1\}$
12 sources & 12 sinks	$Q=\{12,6,6,6,7\}$
18 sources & 18 sinks	$Q=\{18,6,7,6\}$

* $Q=\{q_0,q_1,q_2,\cdots q_{d^{\max}}\}$, where q_i is the number of nodes with distance i

The explanation is similar to networks with one source and multiple sinks. The maximum distance decreases as the number of terminals increases, as shown in Table 3.4. In addition, as the number of terminals increases, the number of nodes with shorter distance increases, while the number of nodes with longer distance decreases. Thus, as the number of terminals increases, the search branches reduced by additional backtracking condition decrease. As a result, the advantage of the proposed algorithm over Chen and Lin's algorithm decreases as more nodes become terminals. However, another advantage of the proposed algorithm, which uses linked path structure indexed by node to represent the network, is still in effect. Thus, despite the number of terminals, the proposed algorithm is still more efficient than Chen and Lin's algorithm [31]. Table 3.5 provides the CPU time required for each case.

Table 3.5: The efficiency comparison for 6 × 6 multi-terminal grid networks

Number of sources and sinks	T_1: CPU Time by Chen and Lin's algorithm (sec.)	T_2: CPU Time by proposed algorithm (sec.)
2 sources & 2 sinks	9281.3671	2921.0970
3 sources & 3 sinks	15809.0553	5619.2184
4 sources & 4 sinks	23134.0536	8506.3059
5 sources & 5 sinks	30336.4158	11659.4397
6 sources & 6 sinks	40590.2462	15943.8487
12 sources & 12 sinks	93229.71638	38224.48994
18 sources & 18 sinks	145299.5843	61605.53568

3.6 Summary

In this chapter, we develop an improved algorithm for searching for all MPs based on backtracking. By computational experiments, it is found that the proposed algorithm is more efficient than existing algorithms in finding all MPs when the network size is not too small, and the efficiency of the proposed algorithm over existing algorithms becomes more advantageous when the size of the network grows. Minimal paths play an important role in reliability evaluation for binary networks. The generated MPs in this chapter are used as building blocks for finding all the *d*-MPs in Chapter 4.

Chapter 4

Search for all *d*-MPs for all *d* levels in Multistate Networks

Recall the overall framework of the topics in this thesis, as shown in Figure 1.3. In this chapter, we focus on the second topic, which is finding all *d*-MPs for all possible demand levels. This is the second step of the indirect approaches for reliability evaluation of multistate networks. In Chapter 3, we propose an algorithm to search for all MPs in a network. After all the MPs have been found, we can use them as building blocks to generate all *d*-MPs. In Section 1.3, we point out that existing algorithms on generating *d*-MPs are for a particular *d* value. However, if all *d*-MPs for all possible integer d values are required, we need to call such methods multiple times with respect to all d values. A more efficient method is desirable to generate all *d*-MPs for all d levels.

In this chapter, we develop a recursive algorithm based on breadth-first search to search for all the *d*-MPs for all possible d values. The relationships among *d*-MPs for different *d* levels are revealed in Section 4.1. The proposed algorithm is given in Section 4.2, followed by its complexity analysis and an illustrative example. A heuristic for pre-processing the MPs is proposed in Section 4.3 to improve the efficiency of the algorithm. Section 4.4 investigates the efficiency of the algorithm. Section 4.5 presents the extension of using the proposed algorithm to search for subsets of *d*-MPs, which can be used for lower reliability bound evaluation. Section 4.6 summarizes the work. The materials in this chapter have been documented in paper [45] and the conference proceedings of *Reliability and Maintainability Symposium (RAMS)* [46].

4.1 Introduction

Given all the MPs have been found, Lin et al. [25] proposed a mathematical formulation with 3 constraints to obtain the *d*-MP candidates. The constraints are: 1) the summation of the flows on all the MPs equals to *d*; 2) the total flow on each MP is smaller than or equal to the maximum capacity on that MP, which is the minimum value of all the maximum capacities of components contained in that MP; 3) the total flow going through each component is less than or equal to its maximum capacity. After the *d*-MP candidates are generated, each *d*-MP candidate is verified through a recursive comparison and all *d*-MPs are obtained for a particular level d. Lin [32] further proved that constraint 2 was redundant and reduced the constraints proposed in [25] from 3 to 2. Another suggestion from Lin et al. [25] was that if the network is acyclic, each generated *d*-MP candidate is a real *d*-MP without verification through a recursive comparison. For cyclic networks, Yeh [33] further removed unsatisfied solutions from the generated *d*-MP candidates by detecting cycles if all cycles of the network are known. The approach proposed by Lin et al. [25] and modified by Lin [32] and Yeh [33] requires MPs as prior knowledge, and enumeration is used to generate the *d*-MP candidates. Chen [58] proposed an enumeration method to improve the efficiency of enumeration by rearranging the locations of constraints during the enumeration process.

From the approach proposed by Lin et al. [25] and modified by Lin [32] and Yeh [33], it can be observed that each *d*-MP candidate can be generated by a combination of MPs, given that the capacity of each component takes successive integer values from 0 to its maximum capacity. Consider the example in Figure 2.1. There are four MPs and their vector representations are given as follows:

$$\mathrm{MP}_1 = \{a_1, a_2\} \Rightarrow (1,1,0,0,0,0);\ \mathrm{MP}_2 = \{a_5, a_6\} \Rightarrow (0,0,0,0,1,1)$$
$$\mathrm{MP}_3 = \{a_1, a_3, a_6\} \Rightarrow (1,0,1,0,0,1);\ \mathrm{MP}_4 = \{a_2, a_4, a_5\} \Rightarrow (0,1,0,1,1,0).$$

The maximum capacity for each component is given by the maximum state vector

$\mathbf{X}^{\mathbf{Max}} = (3,2,1,1,1,2)$, that is, the maximum states for component 1 to 6 are 3, 2, 1, 1, 1 and 2, respectively. There is one cycle formed by component 3 and 4, denoted as $\mathbf{C} = (x_3, x_4)$.

Each 2-MP candidate can be generated by a combination of 2 MPs. For example,

$$\underbrace{(1,1,0,0,0,0)}_{MP_1} + \underbrace{(0,0,0,0,1,1)}_{MP_2} = \underbrace{(1,1,0,0,1,1)}_{2-MP\ candidate}$$

Furthermore, each *d*-MP candidate can be generated by a combination of one $(d-1)$-MP and one MP. For example, a 4-MP can be generated by a combination of a 3-MP and a MP as follows:

$$\underbrace{(3,2,1,0,0,1)}_{3-MP} + \underbrace{(0,0,0,0,1,1)}_{MP_2} = \underbrace{(3,2,1,0,1,2)}_{4-MP\ candidate}$$

Thus, instead of generating *d*-MP candidates for each *d* level by identifying combinations of MPs using enumerations, we can use MPs as building blocks to generate 2-MP candidates, and use 2-MPs and MPs as building blocks to generate 3-MP candidates, …, and so forth. During the process, each newly generated candidate will be verified using constraints derived from [25], [32], and [33] to obtain real *d*-MPs. Invalid candidates will be abandoned to avoid generating more invalid candidates. When the *d*-MPs with respect to the maximum d value have been found, all the *d*-MPs for all possible integer d are found as well.

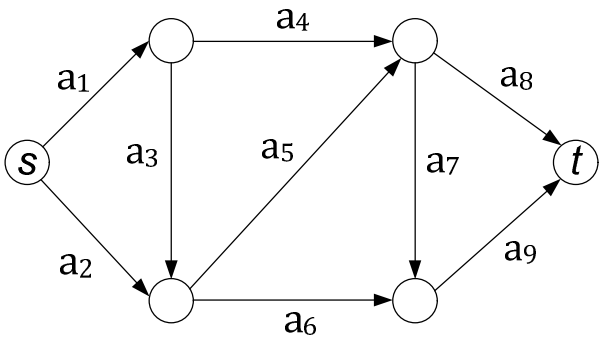

Figure 4.1: A network example from [34]

Although each generated candidate that passes the constraints derived from [25], [32], and

[33] is a real *d*-MP, there may exist redundant *d*-MPs. Consider the following network as shown in Figure 4.1, which is taken from [34].

When we build 2-MPs using 2 MPs, we will generate redundant solutions. For example, consider the following 4 MPs:

$$\mathrm{MP}_1 = \{a_1, a_3, a_5, a_8\} \Rightarrow (1,0,1,0,1,0,0,1,0); \mathrm{MP}_2 = \{a_2, a_6, a_9\} \Rightarrow (0,1,0,0,0,1,0,0,1);$$
$$\mathrm{MP}_3 = \{a_2, a_5, a_7, a_9\} \Rightarrow (0,1,0,0,1,0,1,0,1); \mathrm{MP}_4 = \{a_1, a_3, a_6, a_9\} \Rightarrow (1,0,1,0,0,1,0,0,1).$$

We can build one 2-MP using MP_1 and MP_2, and another 2-MP using MP_3 and MP_4 as follows:

$$\mathrm{MP}_1 + \mathrm{MP}_2 = (1,0,1,0,1,0,0,1,0) + (0,1,0,0,0,1,0,0,1) = (1,1,1,0,1,1,0,1,1)$$
$$\mathrm{MP}_3 + \mathrm{MP}_4 = (0,1,0,0,1,0,0,1,0) + (1,0,1,0,0,1,0,0,1) = (1,1,1,0,1,1,0,1,1)$$

As can be seen, different combinations of 2 MPs can result in the same 2-MP. Since redundant solutions generated at lower *d* levels will lead to even more redundant solutions as *d* increases, they need to be removed right away to avoid consuming additional computational effort.

The next question is how to implement our incremental building and validating process for finding all non-redundant *d*-MPs. There are mainly two search strategies reported in the literature, namely depth-first search and breadth-first search. The depth-first search starts at the root node and explores as deep as possible before backtracking. The breadth-first search starts at the root node and explores all the neighboring nodes. Given those neighboring nodes, it further explores their neighbor nodes and so on. Although both searching strategies can be used to implement our incremental building and validating process for finding all the *d*-MPs, breadth-first search is more efficient in dealing with redundant *d*-MPs. Because redundant *d*-MPs for each *d* level can only be identified given all *d*-MPs for that *d* level are obtained. Based on the observations and discussions above, we report a recursive breadth-first search algorithm to search for all the *d*-MPs for all possible integer d values.

Notation list

MP_i : the vector form of i th MP, $i = 1,2,\ldots,\pi$. $\overline{\mathbf{X}}_{ij}^{d}$: the j th *d*-MP candidate that is built by $\mathrm{MP}_u, \mathrm{MP}_v ..., \mathrm{MP}_w$, with maximum cardinality, $\max(u,v,...,w)$ equal to i for level *d*. $\mathrm{MP}_u, \mathrm{MP}_v ..., \mathrm{MP}_w$ are all binary MPs with cardinality $u, v, ..., w$.

L_d : denote the number of *d*-MPs for a particular *d* level.

ω : the total number of *d*-MPs generated for all d level.

$\mathbf{C}_v$: the v th cycle in the network, $\mathbf{C}_v = (x_i, x_j, \ldots, x_k)$, $v = 1,2,\ldots,c$.

4.2 Algorithm development

In this section, we report the recursive search algorithm, based on the mechanism of breadth-first search. Recall that the vector representation of a MP is indeed an 1-MP for a multistate network. Given all vector forms of MPs, it picks the first MP and adds each MP one at a time to build 2-MP candidates. If the generated 2-MP candidates pass the constraints derived from [25], [32], and [33], they become real 2-MPs. It then picks the second MP, and adds each MP except the first MP one at a time to build 2-MP candidates. If the generated 2-MP candidates pass the constraints derived from [25], [32], and [33], they become real 2-MPs. It then picks the third MP, and adds each MP except the first and second MP one at a time to build 2-MP candidates and so on. This process is implemented recursively until it picks the last MP and adds the last MP to build a 2-MP candidate. After removing the redundant 2-MPs, all the 2-MPs are found. It then proceeds to the next *d* level. This process is implemented recursively until the maximum d level has been reached.

4.2.1 The proposed algorithm

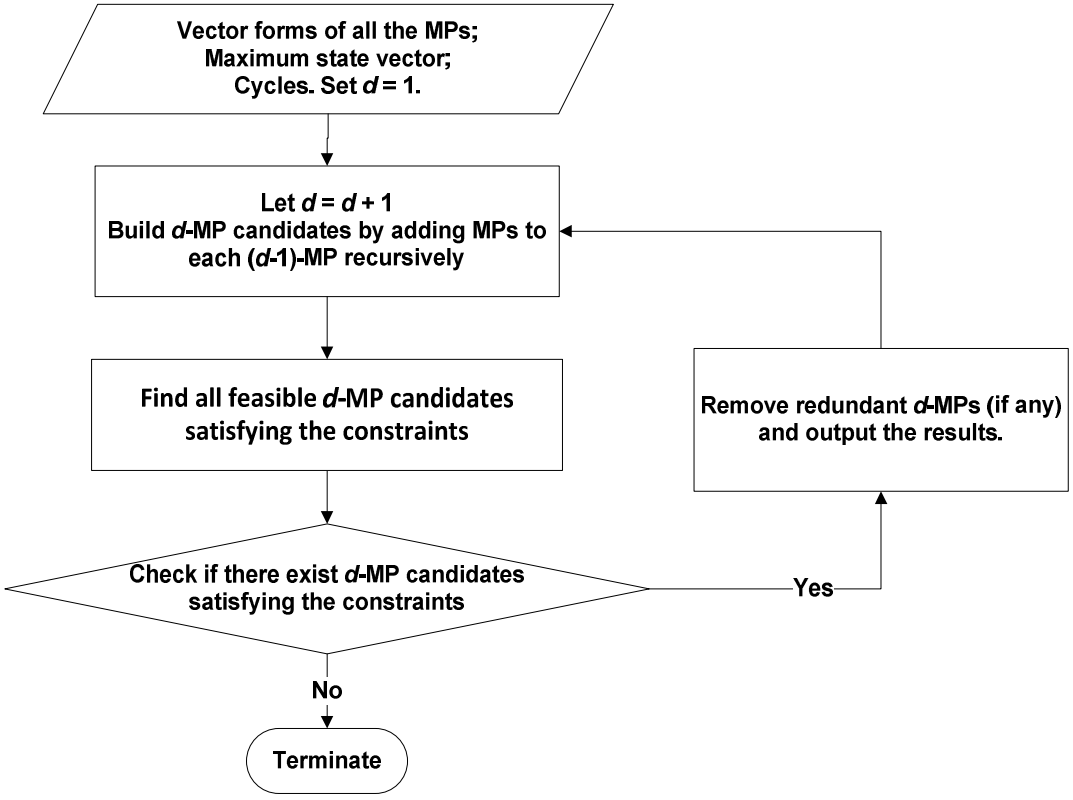

Figure 4.2: Flow chart of algorithm

Figure 4.2 shows a flow chart of the algorithm. The detailed procedure of algorithm is given in the following steps.

Step 1 Input the vector forms of all the MPs, MP_i, $i = 1,2,\ldots,\pi$; the maximum state vector, $\mathbf{X}^{\mathbf{Max}}$ all the cycles, $\mathbf{C}_v$, $v = 1,2,\ldots,c$. Set demand level *d* equal to 1.

Step 2 Let $d = d + 1$. Build *d*-MP candidates $\overline{\mathbf{X}}_{ij}^{d}$ by adding MPs to each $(d-1)$-MP recursively as follows:

- For the $(d-1)$-MPs built by MP_1, add each MP one at a time to build *d*-MP candidates;
- For the $(d-1)$-MPs built by $\mathrm{MP}_u, \mathrm{MP}_v \ldots, \mathrm{MP}_w$, with maximum cardinality, $\max(u,v,\ldots,w)$ equal to 2, add each MP except MP_2 one at a time to build *d*-MP candidates;

- For the $(d-1)$-MPs built by $\mathrm{MP}_u, \mathrm{MP}_v..., \mathrm{MP}_w$, with maximum cardinality, $\max(u,v,...,w)$ equal to 3, add each MP except MP_1 and MP_2 one at a time to build *d*-MP candidates;

  ……

- For the $(d-1)$-MPs built by $\mathrm{MP}_u, \mathrm{MP}_v..., \mathrm{MP}_w$, with maximum cardinality, $\max(u,v,...,w)$ equal to π, add MP_π to build *d*-MP candidates.

Step 3 Find all feasible *d*-MP candidates $\overline{\mathbf{X}}_{ij}^d$ satisfying the following constraints:

- The capacity of each component in $\overline{\mathbf{X}}_{ij}^d$ is smaller than or equal to the maximum capacity of its corresponding component [25], that is:

Let $\overline{\mathbf{X}}_{ij}^d = (x_1, x_2, \ldots, x_n)$, $\overline{\mathbf{X}}_{ij}^d \le \mathbf{X}^{\mathbf{Max}}$, where $\overline{\mathbf{X}}_{ij}^d \le \mathbf{X}^{\mathbf{Max}}$ means $x_i \le M^i$, for $i = 1,2,\ldots,n.$

- For cyclic network, $\overline{\mathbf{X}}_{ij}^d$ contains no cycles [33], i.e. there exist at least one component for $\overline{\mathbf{X}}_{ij}^d$ that has capacity equal to 0 for each cycle:

Let $\overline{\mathbf{X}}_{ij}^d = (x_1, x_2, \ldots, x_n)$, $\min(\mathbf{C}_v) = \min(x_i, x_j, \ldots, x_k) = 0,$ for $v = 1,2,\ldots,c$.

Step 4 If there is no *d*-MP candidate that can pass all the constraints, terminate the algorithm; otherwise, for all *d*-MP candidates satisfying all the constraints, remove the redundant ones and output them. They are all the real *d*-MPs for level d. Go to step 2 for the next d level.

4.2.2 An illustrative example

Consider the earlier example in Section 4.1, shown in Figure 2.1, with the information on MPs, Maximum state vector and cycles.

1. Input all the vector forms of MPs as follows:

$$\mathrm{MP}_1 = (1,1,0,0,0,0); \mathrm{MP}_2 = (0,0,0,0,1,1); \mathrm{MP}_3 = (1,0,1,0,0,1); \mathrm{MP}_4 = (0,1,0,1,1,0).$$

Input maximum state vector $\mathbf{X}^{\mathbf{Max}} = (3,2,1,1,1,2)$, and cycle $\mathbf{C} = (x_3, x_4)$. Set *d* = 1;

2. Let $d = d + 1 = 2$. Build 2-MP candidates by adding MPs to each MP (1-MP) recursively as follows:

- For MP_1 with maximum cardinality 1, add MP_1, MP_2, MP_3 and MP_4 one at a time and obtain the following 2-MP candidates:

$$\overline{\mathbf{X}}_{11}^{2} = \mathrm{MP}_1 + \mathrm{MP}_1 = (2,2,0,0,0,0);\ \overline{\mathbf{X}}_{21}^{2} = \mathrm{MP}_1 + \mathrm{MP}_2 = (1,1,0,0,1,1);$$
$$\overline{\mathbf{X}}_{31}^{2} = \mathrm{MP}_1 + \mathrm{MP}_3 = (2,1,1,0,0,1);\ \overline{\mathbf{X}}_{41}^{2} = \mathrm{MP}_1 + \mathrm{MP}_4 = (1,2,0,1,1,0).$$

  $\overline{\mathbf{X}}_{11}^{2}$ is the 1st *d*-MP candidate built by MPs with the maximum cardinality 1 for level 2;

  $\overline{\mathbf{X}}_{21}^{2}$ is the 1st *d*-MP candidate built by MPs with the maximum cardinality 2 for level 2;

  $\overline{\mathbf{X}}_{31}^{2}$ is the 1st *d*-MP candidate built by MPs with the maximum cardinality 3 for level 2;

  $\overline{\mathbf{X}}_{41}^{2}$ is the 1st *d*-MP candidate built by MPs with the maximum cardinality 4 for level 2.

- For MP_2 with maximum cardinality 2, add MP_2, MP_3 and MP_4 one at a time and obtain the following 2-MP candidates:

$$\overline{\mathbf{X}}_{22}^{2} = \mathrm{MP}_2 + \mathrm{MP}_2 = (0,0,0,0,2,2);\ \overline{\mathbf{X}}_{32}^{2} = \mathrm{MP}_2 + \mathrm{MP}_3 = (1,0,1,0,1,2);$$
$$\overline{\mathbf{X}}_{42}^{2} = \mathrm{MP}_2 + \mathrm{MP}_4 = (0,1,0,1,2,1).$$

  $\overline{\mathbf{X}}_{22}^{2}$ is the 2nd *d*-MP candidate built by MPs with the maximum cardinality 2 for level 2;

  $\overline{\mathbf{X}}_{32}^{2}$ is the 2nd *d*-MP candidate built by MPs with the maximum cardinality 3 for level 2;

  $\overline{\mathbf{X}}_{42}^{2}$ is the 2nd *d*-MP candidate built by MPs with the maximum cardinality 4 for level 2.

- For MP_3 with maximum cardinality 3, add MP_3 and MP_4 one at a time and obtain the following 2-MP candidates.

$$\overline{\mathbf{X}}_{33}^{2} = \mathrm{MP}_3 + \mathrm{MP}_3 = (2,0,2,0,0,2);\ \overline{\mathbf{X}}_{43}^{2} = \mathrm{MP}_3 + \mathrm{MP}_4 = (1,1,1,1,1,1).$$

  $\overline{\mathbf{X}}_{33}^{2}$ is the 3rd *d*-MP candidate built by MPs with the maximum cardinality 3 for level 2;

  $\overline{\mathbf{X}}_{43}^{2}$ is the 3rd *d*-MP candidate built by MPs with the maximum cardinality 4 for level 2.

- For MP_4 with maximum cardinality 4, add MP_4 one at a time and obtain the following 2-MP candidate.

$$\overline{\mathbf{X}}_{44}^{2} = \mathrm{MP}_4 + \mathrm{MP}_4 = (0,2,0,2,2,0).$$

$\overline{\mathbf{X}}_{44}^{2}$ is the 4th *d*-MP candidate built by MPs with the maximum cardinality 4 for level 2.

3. Find all feasible 2-MP candidates satisfying all the constraints as follows:

For $\overline{\mathbf{X}}_{11}^{2}$, since $\overline{\mathbf{X}}_{11}^{2} \le \mathbf{X}^{\mathbf{Max}}$ and $\min(x_3, x_4) = \min(0,0) = 0$, $\overline{\mathbf{X}}_{11}^{2}$ is a 2-MP.

For $\overline{\mathbf{X}}_{21}^{2}$, since $\overline{\mathbf{X}}_{21}^{2} \le \mathbf{X}^{\mathbf{Max}}$ and $\min(x_3, x_4) = \min(0,0) = 0$, $\overline{\mathbf{X}}_{21}^{2}$ is a 2-MP.

For $\overline{\mathbf{X}}_{31}^{2}$, since $\overline{\mathbf{X}}_{31}^{2} \le \mathbf{X}^{\mathbf{Max}}$ and $\min(x_3, x_4) = \min(1,0) = 0$, $\overline{\mathbf{X}}_{31}^{2}$ is a 2-MP.

For $\overline{\mathbf{X}}_{41}^{2}$, since $\overline{\mathbf{X}}_{41}^{2} \le \mathbf{X}^{\mathbf{Max}}$ and $\min(x_3, x_4) = \min(0,1) = 0$, $\overline{\mathbf{X}}_{41}^{2}$ is a 2-MP.

For $\overline{\mathbf{X}}_{22}^{2}$, since $\overline{\mathbf{X}}_{22}^{2} \ge \mathbf{X}^{\mathbf{Max}}$, $\overline{\mathbf{X}}_{22}^{2}$ is not a 2-MP.

For $\overline{\mathbf{X}}_{32}^{2}$, since $\overline{\mathbf{X}}_{32}^{2} \le \mathbf{X}^{\mathbf{Max}}$ and $\min(x_3, x_4) = \min(1,0) = 0$, $\overline{\mathbf{X}}_{32}^{2}$ is a 2-MP.

For $\overline{\mathbf{X}}_{42}^{2}$, since $\overline{\mathbf{X}}_{42}^{2} \ge \mathbf{X}^{\mathbf{Max}}$, $\overline{\mathbf{X}}_{42}^{2}$ is not a 2-MP.

For $\overline{\mathbf{X}}_{33}^{2}$, since $\overline{\mathbf{X}}_{33}^{2} \ge \mathbf{X}^{\mathbf{Max}}$, $\overline{\mathbf{X}}_{33}^{2}$ is not a 2-MP.

For $\overline{\mathbf{X}}_{43}^{2}$, since $\min(x_3, x_4) = \min(1,1) = 1$, $\overline{\mathbf{X}}_{43}^{2}$ is not a 2-MP.

For $\overline{\mathbf{X}}_{44}^{2}$, since $\overline{\mathbf{X}}_{44}^{2} \ge \mathbf{X}^{\mathbf{Max}}$, $\overline{\mathbf{X}}_{44}^{2}$ is not a 2-MP.

4. There are 5 2-MP candidates satisfying the constraints, and there are no redundant solutions. Output these 2-MPs as follows:

$$\overline{\mathbf{X}}_{11}^{2} = (2,2,0,0,0,0),\ \overline{\mathbf{X}}_{21}^{2} = (1,1,0,0,1,1),\ \overline{\mathbf{X}}_{31}^{2} = (2,1,1,0,0,1),$$

$$\overline{\mathbf{X}}_{41}^{2} = (1,2,0,1,1,0),\ \overline{\mathbf{X}}_{32}^{2} = (1,0,1,0,1,2).$$

Go to step 2 for next *d* level.

5. Let $d = d + 1 = 3$. Build 3-MP candidates by adding MPs to each 2-MP recursively as follows:
   - For the 2-MP that is built by MPs with maximum cardinality 1, $\overline{\mathbf{X}}_{11}^{2}$, add MP_1, MP_2, MP_3 and MP_4 one at a time and obtain 4 3-MP candidates.
   - For the 2-MP that is built by the MPs with maximum cardinality 2, $\overline{\mathbf{X}}_{21}^{2}$, add MP_2, MP_3, and MP_4 one at a time and obtain 4 3-MP candidates.
   - For the 2-MPs that are built only by the MPs with maximum cardinality 3, $\overline{\mathbf{X}}_{31}^{2}$ and $\overline{\mathbf{X}}_{32}^{2}$, add MP_3 and MP_4 one at a time and obtain 4 3-MP candidates.
   - For the 2-MP that is first built by the MPs with maximum cardinality 4, $\overline{\mathbf{X}}_{41}^{2}$, add MP_4 one at a time and obtain one 3-MP candidate.
6. Find all feasible 3-MP candidates satisfying all the constraints. We obtain 3 non-redundant 3-MPs. Output these 3-MPs as follows:

   $$(2,2,0,0,1,1),(3,2,1,0,0,1),(2,1,1,0,1,2).$$

   Go to step 2 for the next *d* level.
7. Let $d = d + 1 = 4$. Build 4-MP candidates by adding MPs to each 3-MP recursively. Find all feasible 4-MP candidates satisfying the constraints. We obtain one 4-MP, $(3,2,1,0,1,2)$. Go to step 2 for the next d level.
8. Let $d = d + 1 = 5$. Build 5-MP candidates by adding MPs to 4-MP recursively. Since there is no feasible 5-MP candidate satisfying all the constraints, terminate the search and all *d*-MPs for all d values have been found.

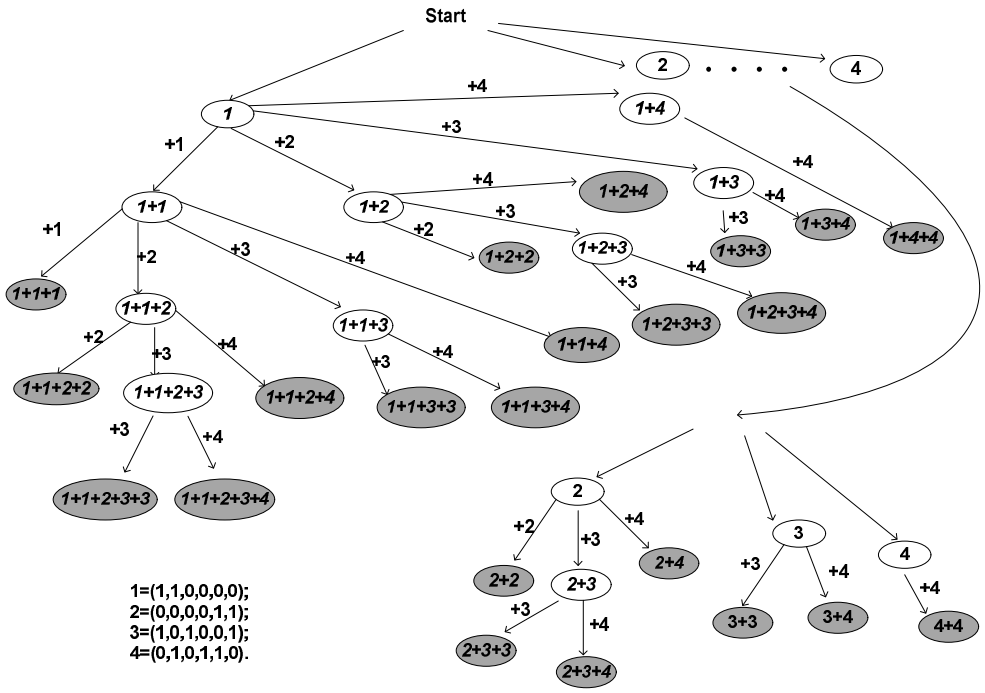

Figure 4.3: Implementation of algorithm

Figure 4.3 shows the whole procedure of the implementation for the illustrative example. In the end, we obtain the following *d*-MPs:

MPs: $(1,1,0,0,0,0)$, $(0,0,0,0,1,1)$, $(1,0,1,0,0,1)$, $(0,1,0,1,1,0)$;

2-MPs: $(2,2,0,0,0,0)$, $(1,1,0,0,1,1)$, $(2,1,1,0,0,1)$, $(1,2,0,1,1,0)$, $(1,0,1,0,1,2)$;

3-MPs: $(2,2,0,0,1,1)$, $(3,2,1,0,0,1)$, $(2,1,1,0,1,2)$;

4-MPs: $(3,2,1,0,1,2)$.

By setting each component to its maximum capacity and implementing the maximum flow algorithm, the maximum flow we obtain for this network example is indeed 4, which verifies the proposed algorithm's correctness on the highest level of d. The three 3-MP*s* found are the same as those in Lin et al. [25]. By implementing the approach proposed by Lin et al. [25] for each level d from 1 to 4, we obtain the same sets of *d*-MPs for all d level, which verifies the proposed algorithm's correctness on finding all the *d*-MPs for all d levels.

4.2.3 Complexity analysis

The complexity analysis is conducted as follows. Let L_d denote the number of *d*-MPs for a particular *d* level, and M denote a big number that is greater than the maximum L_d among all d levels. M is used to pre-allocate the space for storing *d*-MPs for one *d* level. Recall that n is the number of components, π is the number of MPs, i.e. 1-MPs, and c is the number of cycles. The storage complexity of the proposed algorithm is $O\left[\left(\pi+1+c+\mathrm{M}\right)\cdot n\right]$ with respect to all 1-MPs buffer (equal to $\pi\cdot n$), one maximum state vector $\mathbf{X}^{\mathbf{Max}}$ (equal to n), all the cycles (equal to $c\cdot n$ in the worst case), and the number of *d*-MPs for one *d* level (equal to $\mathrm{M}\cdot n$ in the worst case).

For the time complexity, recall that L_d is the number of *d*-MPs for a particular d level and π is the number of MPs, i.e., the number of 1-MPs. When we implement the proposed algorithm to generate the 2-MPs, the first 1-MP is used π times to build *d*-MP candidates, the second 1-MP is used $\pi-1$ times to build *d*-MP candidates, …, and the last 1-MP is used 1 time to build *d*-MP candidates. Then each *d*-MP candidate is verified once. Thus, it takes $O\left(\sum_{i=1}^{\pi} i\right)=O\left(\frac{\pi\cdot\left(\pi+1\right)}{2}\right)$ time to generate the *d*-MPs for level 2. When we implement the proposed algorithm to generate the *d*-MPs for level *d*+1, the number of times each *d*-MP is used depends on which (*d*-1)-MP and MP it was built from. Thus, by considering the worst case scenario, each *d*-MP in level *d* is used at most π times to build *d*-MP candidates. Then each *d*-MP candidate is verified once. Thus, it takes $O\left(\pi\cdot L_d\right)$ time to generate the *d*-MPs for level *d*+1 in the worst case. By aggregating the results, let ω denote the total number of *d*-MPs generated for all d level, where $d\geq 2$. It requires $O\left(\frac{\pi\cdot\left(\pi+1\right)}{2}+\pi\cdot\omega\right)$ time for the proposed algorithm to generate all the *d*-MPs for all d level.

4.3 Pre-processing of MPs

As can be seen from Section 4.2, the computational complexity of the propose algorithm is affected by the number of *d*-MP candidates generated. Some *d*-MP candidates prove to be real *d*-MPs after validation, and others fail to satisfy the constraints. Since our target is to find all real *d*-MPs for all d levels, the efficiency could be improved if we can generate less number of *d*-MP candidates that are not real *d*-MPs.

During the implementation of the algorithm, the MPs are given in an arbitrary order. It has been suggested that by arranging minimal cuts in a certain order, we can improve the efficiency of finding minimal cut vectors (*d*-MCs) [59]. Motivated by [59], we believe that different sequences of MPs may influence the efficiency of the proposed algorithm. By analyzing the algorithm, we obtain the following properties.

Firstly, we define the minimum state of a MP as follows. Given $\mathrm{MP}_i = \{a_1, a_2, \cdots, a_n\}$ and the maximum state of components contained in MP_i, $M^1, M^2, \cdots, M^n$, the minimum state of MP_i is

$$\mathrm{MS}_i = \min(M^1, M^2, \cdots, M^n)$$

We also define the length of MP as the number of components contained in the MP.

Property 1 *Adding MPs with lower minimum state are likely to produce non-real d-MPs.*

During the process of the algorithm, many candidates fail to pass the constraint that the capacity of each component is less or equal to its maximum capacity. Since the algorithm builds candidates by incrementally adding MPs, it is more likely for MP with lower minimum state to fail this constraint.

Property 2 For *MPs with the same minimum state, adding MPs* with higher number of components are *likely to produce non-real d-MPs.*

For MPs with the same minimum state, it is more likely for MPs with higher number of components to fail the constraint that the state of each component is less than or equal to its maximum state.

Property 3 *Ordering MPs that are more likely to produce non-real d-MPs in earlier positions is more likely to reduce the total number of d-MP candidates generated.*

Since the algorithm is based on a recursive searching mechanism and the number of MPs is π, the number of *d*-MP candidates built for the 1st branch of the searching process is π, $\pi-1$ for the 2nd branch, …, $\pi-k$ for the *(k+1)th* branch, …, and 1 for the last branch, for each *d* level. Thus, during the incremental building process of the proposed algorithm, for each *d* level, the number of *d*-MP candidates generated in an earlier branch is larger than those generated in a late branch. Having a failed *d*-MP candidate can terminate the searching branch earlier.

With the observation above, it can be seen that ordering MPs that are more likely to produce failed *d*-MP candidates in earlier positions is more likely to generate failed *d*-MP candidates in lower d levels, which avoid generating more failed *d*-MP candidates.

Based on the three properties, we propose the following pre-processing rules, called ORDER, for MPs:

1. Obtain the minimum state, MS_i, and the length for each MP_i.
2. Order the MPs in ascending order according to their minimum state.
3. For MPs with the same minimum state, order them in descending order according to their length.

During the process, any tie will be broken by random choice.

Consider the same illustrative example used in Sections 3 and 5. We have the MPs given in the following arbitrary order:

Arbitrary order	a_1	a_2	a_3	a_4	a_5	a_6
1	1	1	0	0	0	0
2	0	0	0	0	1	1
3	0	1	0	1	1	0
4	1	0	1	0	0	1

We apply the pre-processing rule ORDER, and obtain the minimum state and length for each MP as follows:

Current order	a_1	a_2	a_3	a_4	a_5	a_6	MS_i	Length
1	1	1	0	0	0	0	2	2
2	0	0	0	0	1	1	1	2
3	0	1	0	1	1	0	1	3
4	1	0	1	0	0	1	1	3

Order the MPs in ascending order according to their minimum states as follows:

Current order	a_1	a_2	a_3	a_4	a_5	a_6	MS_i	Length
1	0	0	0	0	1	1	1	2
2	0	1	0	1	1	0	1	3
3	1	0	1	0	0	1	1	3
4	1	1	0	0	0	0	2	2

For MPs with the same minimum state, order them in descending order according to their length, and obtain the final order as follows:

Current order	a_1	a_2	a_3	a_4	a_5	a_6	MS_i	Length
1	0	1	0	1	1	0	1	3
2	1	0	1	0	0	1	1	3
3	0	0	0	0	1	1	1	2
4	1	1	0	0	0	0	2	2

We apply the proposed algorithm in Section 4.2 to search for all the *d*-MPs, and the implementation process is given below.

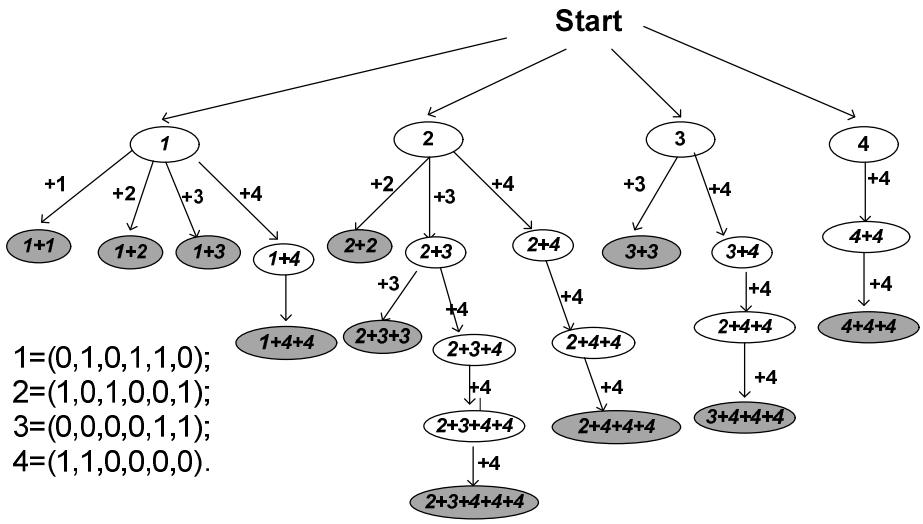

Figure 4.4: Implementation of the proposed algorithm with pre-processed MPs

As can be seen from Figure 4.4, we still generate the exactly same 9 real *d*-MPs, but reduce the search space by only generating 20 candidates. The number of candidates generated in the previous illustrative example is 31. Thus, the pre-processing rule for MPs, ORDER, is incorporated as the first step into the algorithm.

4.4 Efficiency investigation

In this section, we investigate the efficiency of the proposed algorithm. Note that the proposed algorithm may not be as efficient for searching *d*-MPs for one specific level d, comparing to the existing methods dedicated to this purpose, due to its recursive nature. However, recall the objective of this study is to search for *d*-MPs for all possible d levels, which is also significant. It can be used when we are interested in the system reliability with respect to each of the system performance levels, in order to obtain a complete picture of the system capability during the

design phase or operation phase. Thus, we compare the efficiency of our reported algorithm with reported works for finding all *d*-MPs for all possible integers d.

There are two types of approaches for generating *d*-MPs for a particular level d in the reported works. We compare the proposed algorithm with both of them. For approaches requiring MPs as prior knowledge, we compare the efficiency of our algorithm with the approach proposed by Lin et al. [25], modified by Lin [32] and Yeh [33]. We also incorporate the improved enumeration method proposed by Chen [58]. We call the approach by [25], [32], [33], and [58], A1 for short.

For approaches without requiring MPs as prior knowledge, given that the algorithm by Ramirez-Marquez et al. [35] is recognized as an efficient algorithm, we compare the efficiency of our reported algorithm with that by Ramirez-Marquez et al. [35] for finding all *d*-MPs for all possible *d* levels. The algorithm by Ramirez-Marquez et al. [35] requires successor matrix and cycles as prior information. The proposed algorithm requires all the MPs and cycles. Thus, we need to generate MPs given the successor matrix, and we include this procedure when we compare the efficiency. We adopt the method described in [31] to generate all MPs given the successor matrix. We call the approach by [35] A2 for short.

All algorithms were coded in MATLAB 2011, and were implemented on a server with 2 AMD 2.3 GHz CPU (12% CPU was allocated for each user) and 16 GB of RAM. In general, the maximum value of d increases with the maximum capacity of the component. We assume all the component are i.i.d and we are interested in the required computation time for finding all *d*-MPs for all d levels with respect to different maximum state of component.

4.4.1 Example 1

First, we consider the network used in the illustrative example in Section 4.2.2, as shown in Figure 2.1. The successor matrix in this example is shown in Table 4.. U_{ij} = 1 means that

component *i* is a successor of component *j*. The highest states of the components are the same. We consider different scenarios, where the highest states of the components are equal to 1, 2, …, 20, respectively. We use ratios, which are defined as the CPU time consumed by A1 and A2, divided by CPU time consumed by the proposed method, respectively. The ratios represent the advantage of proposed algorithm over the reported works for finding all *d*-MPs for all *d* levels.

Table 4.1: Successor Matrix of the bridge network

U_{ij}	1	2	3	4	5	6
1	*0*	*1*	*1*	*0*	*0*	*0*
2	*0*	*0*	*0*	*0*	*0*	*0*
3	*0*	*0*	*0*	*0*	*0*	*1*
4	*0*	*1*	*0*	*0*	*0*	*0*
5	*0*	*0*	*0*	*1*	*0*	*1*
6	*0*	*0*	*0*	*0*	*0*	*0*

Firstly, we compare the efficiency of our reported algorithm with the approach A1. As can be seen from Figure 4.5, when the maximum state of component is less than 2, the proposed approach is less efficient. This is because the number of calls to the reported approach is small when the maximum state of component is small, i.e., the level of demand is small. However, when the maximum state of component is more than 3, the proposed approach is more efficient. As the maximum state increases, the ratio increases, indicating the advantage becomes more significant. This can be explained as follows. As the maximum state of the component increases, the number of levels of demand increases. This increases the number of calls to the reported approach, which will consume lots of computational efforts. For the proposed method though, the *d*-MPs are obtained recursively, i.e., those for level *d* are obtained based on the *d*-MPs obtained for level *d*-1 and earlier, and the incremental computation efforts for level *d* alone is much less comparing to the existing methods.

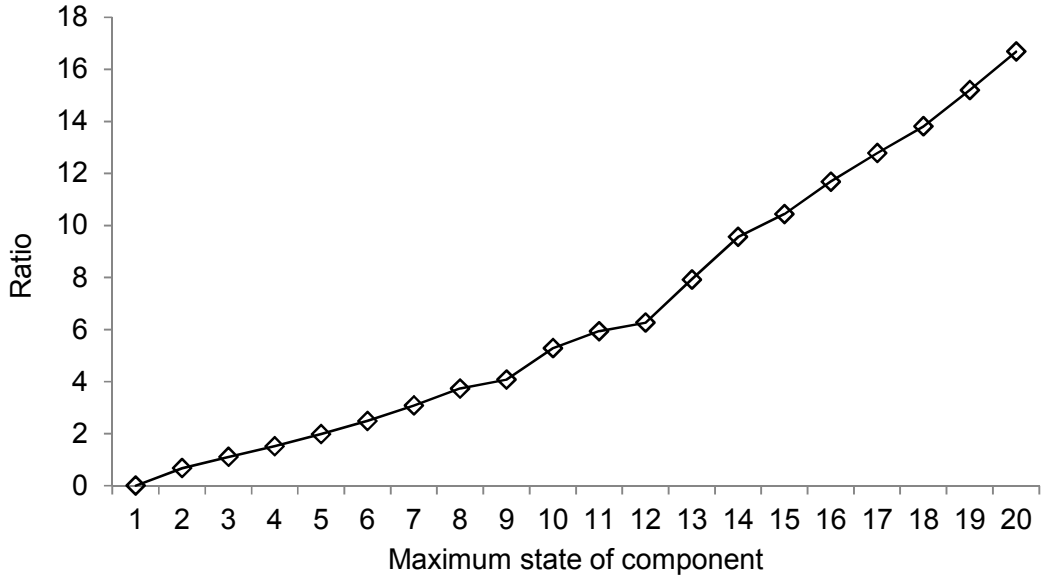

Figure 4.5: Comparison with the approach A1 for the bridge network

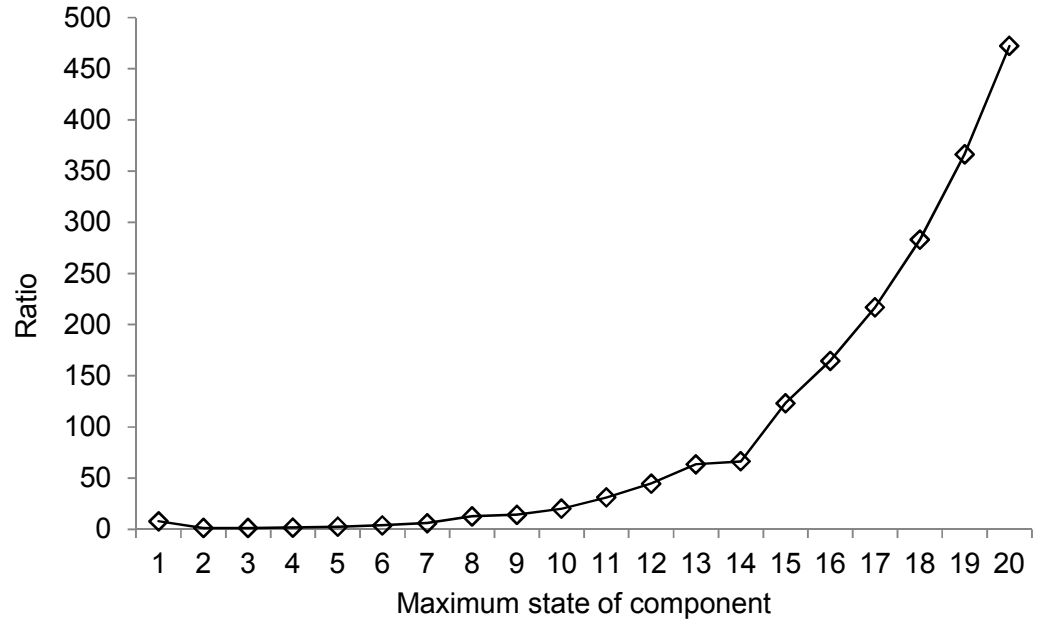

Figure 4.6: Comparison with the approach A2 for the bridge network

Secondly, we compare the efficiency of our proposed approach with A2 by Ramirez-Marquez et al. [35]. As can be seen from Figure 4.6, all ratios for different maximum state are greater than 1, indicating the proposed algorithm is more efficient than A2 in terms of searching

for all *d*-MPs for all *d* values given different maximum states of components. As the maximum state increases, the advantage becomes more significant. Note that the ratio for the first maximum state is relatively high. This is because the proposed algorithm uses the algorithm reported in [31] to search for 1-MPs, which is more efficient than A2 to search for 1-MPs. When the maximum state of each component is 1, the demand level *d* can only take 1 and 2 for the testing network examples, i.e. there are only 1-MPs and 2-MPs generated. The computational saving of generating 1-MPs makes the ratio of CPU time very high when the maximum state of each component is 1. When the maximum state of each component is 2, the demand level *d* can take 1, 2, 3 and 4 for the testing network examples, i.e., there are 1-MPs, 2-MPs, 3-MPs and 4-MPs generated. The computational saving on generating all 1-MPs becomes relatively small, which makes the ratio smaller. In addition, the average CPU time for the algorithm with pre-processing of MPs, ORDER, is about 1.4 times faster than the average CPU time for the algorithm without pre-processing of MPs for each *d* level.

4.4.2 Example 2

In this section, we consider the network shown in Figure 4.1. It has 9 components, and is more complex than the network considered in the previous example. The successor matrix of this example is shown in Table 4.2. We consider different scenarios, where the highest states of the components are equal to 1, 2, …, 10, respectively. We use ratios, which are defined as the CPU times consumed by the reported works divided by the CPU time consumed by the proposed algorithm, respectively. The ratios represent the advantage of proposed algorithm over the reported works for finding all *d*-MPs for all *d* levels.

Table 4.2: Successor Matrix of network in [34]

U_{ij}	1	2	3	4	5	6	7	8	9
1	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*0*
2	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*
3	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*
4	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*0*
5	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*0*
6	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
7	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*
8	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
9	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Similarly, we first compare the efficiency of our proposed algorithm with the approach A1. As can be seen from Figure 4.7, when the maximum state of the component is less than 3, the proposed approach is less efficient. However, when the maximum state of component is 4 or more in this example, the proposed approach is more efficient. As the maximum state increases, the advantage becomes more significant.

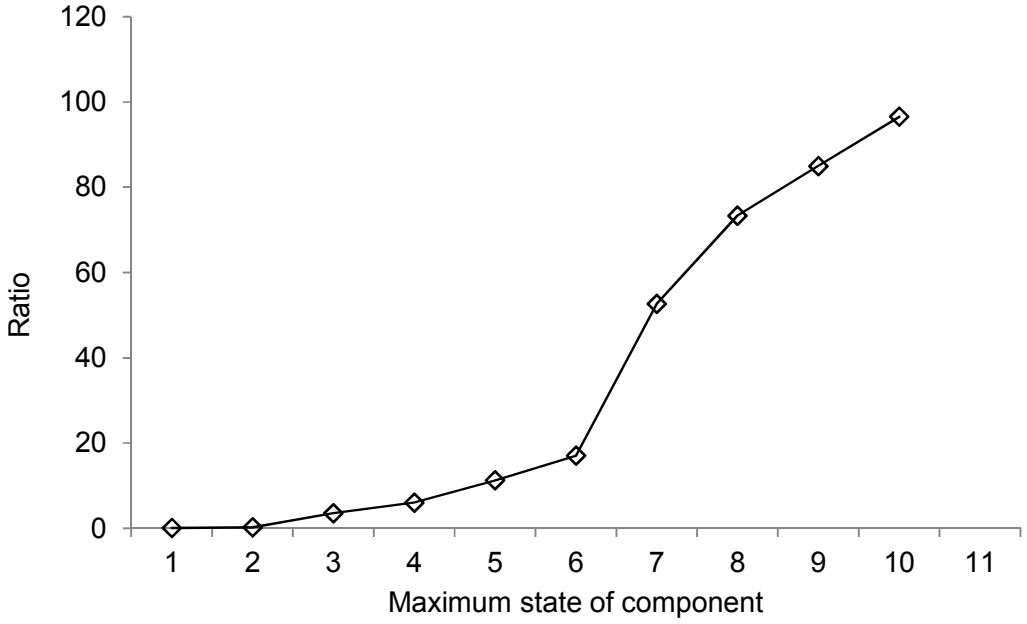

Figure 4.7: Comparison with the approach A1 for the network in [34]

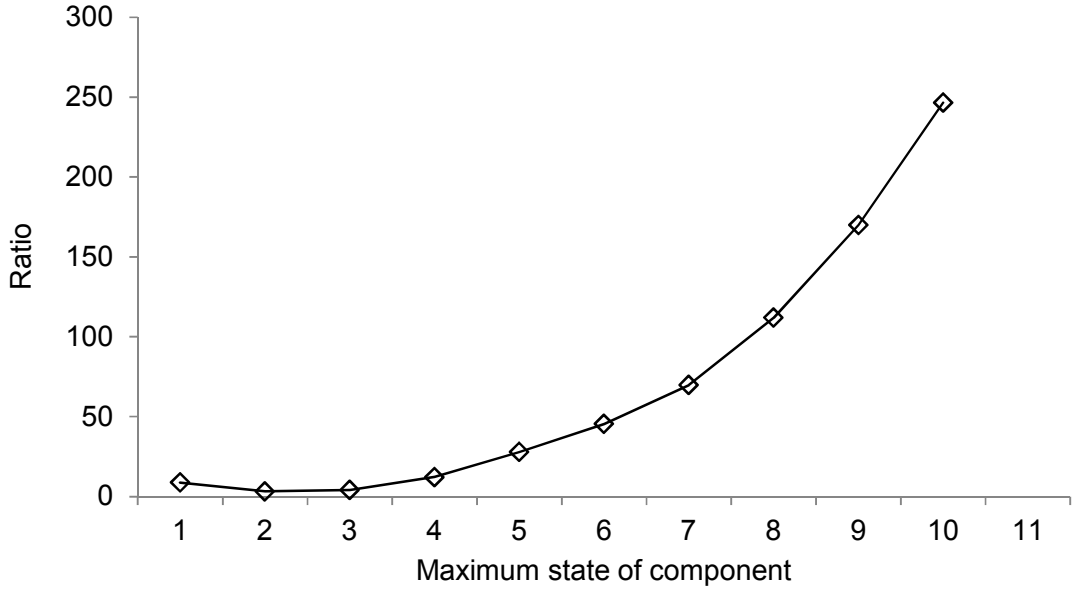

Figure 4.8: Comparison with the approach A2 for the network in [34]

Secondly, we compare the efficiency of our proposed approach with A2 by Ramirez-Marquez et al. [35]. As can be seen from Figure 4.8, all ratios for different maximum state are greater than 1, indicating the proposed algorithm is more efficient than A2 in terms of finding all *d*-MPs for all *d* values given different maximum states of the components. As the maximum state increases, the advantage becomes more significant. In addition, the average CPU time of the algorithm with pre-processing of MPs, ORDER, is about 1.3 times faster than the average CPU time of the algorithm without pre-processing of MPs for each *d* level.

4.5 Reliability bounding by searching for subsets of *d*-MPs

Given the need to find more efficient algorithm to search for all the *d*-MPs, it may still be cumbersome when the size of the network and the number of component states are relatively large. Since the multistate network reliability is calculated as the probability of the union of events, with each event involving a *d*-MP, if only a subset of *d*-MPs is available, the lower

reliability bounds (LRB) can be obtained. LRB can provide approximate reliability values with less computational effort.

As can be seen from the proposed algorithm, if only a subset of MPs is given, we can use the proposed algorithm to search for a subset of *d*-MPs for each *d* value. Thus, we can generate a subset of *d*-MPs for each *d* value as follows. The first subset of *d*-MPs for each *d* value is obtained by using one MP. Then additional MPs are included one at a time, and all the *d*-MPs for all *d* values are found when all MPs are included. By doing so, we only require a subset of MPs and only generate a subset of *d*-MPs for each *d* value, which relaxes the computational burden of the two NP-hard problems. Besides, the LRBs obtained using the generated subsets of *d*-MPs are always between 0 and 1, and these LRBs are always strictly monotonically increasing [41].

Consider Example 1 in Section 4.4.1. We have 4 MPs as follows:

$$\mathrm{MP}_1=(1,1,0,0,0,0);\mathrm{MP}_2=(0,0,0,0,1,1);\mathrm{MP}_3=(1,0,1,0,0,1);\mathrm{MP}_4=(0,1,0,1,1,0).$$

Table 4.3 shows the generated subset of *d*-MPs for each *d* value given subset of MPs.

Table 4.3: Subset of *d*-MPs for bridge network

MPs used	MP_1	MP_1, MP_2	MP_1, MP_2, MP_3	MP_1, MP_2, MP_3, MP_4
2-MPs	$(2,2,0,0,0,0)$	$(2,2,0,0,0,0)$, $(1,1,0,0,1,1)$	$(2,2,0,0,0,0)$, $(1,1,0,0,1,1)$, $(2,1,1,0,0,1)$, $(1,0,1,0,1,2)$	$(2,2,0,0,0,0)$, $(1,1,0,0,1,1)$, $(2,1,1,0,0,1)$, $(1,0,1,0,1,2)$, $(1,2,0,1,1,0)$
3-MPs		$(2,2,0,0,1,1)$	$(2,2,0,0,1,1)$, $(3,2,1,0,0,1)$, $(2,1,1,0,1,2)$	$(2,2,0,0,1,1)$, $(3,2,1,0,0,1)$, $(2,1,1,0,1,2)$
4-MPs			$(3,2,1,0,1,2)$	$(3,2,1,0,1,2)$

Consider Example 2 in Section 4.4.2. It has 9 components and 8 MPs. Suppose each component has 13 states, from 0 to 12, and the state distributions of all the components are identical, denoted as

$$p = (0.05, 0.1, 0.05, 0.05, 0.05, 0.15, 0.05, 0.15, 0.05, 0.1, 0.05, 0.1, 0.05).$$

We first use the proposed algorithm to generate a subset of *d*-MPs for several *d* values using one MP. Then additional MPs are included one at a time until all 8 MPs are included. Given the current efficient method for reliability evaluation of multistate networks is the recursive sum of disjoint product method, namely RSDP [39], we apply RSDP method to calculate the LRB for each generated subset of *d*-MPs for these *d* values. We record the CPU time it takes to obtain the subset of *d*-MPs using the proposed algorithm.

Table 4.4 shows the LRBs with respect to different numbers of MPs used. When only one or two MPs are included, the LRBs with *d* equal to 20 and 24 are zero, indicating no *d*-MPs can be found in these situations when using only the first two MPs. As more MPs are included, subsets of *d*-MPs for each *d* value can be obtained and the LRBs are closer to the exact reliability values. When all 8 MPs are given as input, the exact reliability for these *d* values can be found. Besides, the CPU time increases as the number of MPs used increases. One can choose a proper tradeoff between the computational effort and the evaluation precision. In this example, when 6 MPs are used, the LRBs are almost the same as the exact reliability values, but the required CPU time is much less comparing to the case when using all 8 MPs.

4.6 Summary

In this chapter, we develop a recursive algorithm based on breadth-first search to search for all the *d*-MPs for all possible integer *d* values. A heuristic for pre-processing the MPs is proposed to improve the efficiency of the algorithm. Through computational experiments, it is found that the proposed algorithm is more efficient than existing algorithms for finding all *d*-MPs for all

possible integer *d* values. The generated *d*-MPs can be used for reliability evaluation and system state distribution evaluation of multistate networks. In addition, we show that the proposed algorithm can also be used to generate a subset of *d*-MPs for all *d* values given a subset of MPs. The generated subset of *d*-MPs can be used for lower reliability bound evaluation.

The generated *d*-MPs in this chapter will be used for RSDP type method in Chapter 5 and SSD type method in Chapter 6 respectively for obtaining the reliability of multistate networks.

Table 4.4: LRBs with respect to different number of MPs used and several *d* values

Number of MPs	1	2	3	4	5	6	7	8
d = 1	8.574E-01	9.410E-01	9.878E-01	9.898E-01	9.920E-01	9.941E-01	9.943E-01	9.943E-01
d = 5	3.430E-01	5.693E-01	7.614E-01	7.862E-01	8.118E-01	8.418E-01	8.483E-01	8.486E-01
d = 10	8.000E-03	7.660E-02	2.818E-01	3.185E-01	3.447E-01	3.968E-01	4.060E-01	4.063E-01
d = 20	0	0	3.436E-04	7.522E-04	8.273E-04	1.707E-03	1.723E-03	1.723E-03
d = 24	0	0	1.563E-08	1.280E-07	1.280E-07	8.117E-07	8.117E-07	8.117E-07
CPU Time (s)	0.0074	0.0394	0.0424	0.2258	0.9256	9.472	98.9457	880.4098

Chapter 5

Ordering Heuristics for Reliability Evaluation of Multistate Networks Using Recursive Sum of Disjoint Product Method

Recall the overall framework of the topics in this thesis, as shown in Figure 1.3. In this chapter, we focus on the third topic, which is evaluating the probability of union of all *d*-MPs. This is the third step of the indirect approaches for reliability evaluation of multistate networks. In Chapter 4, we propose an algorithm to search for all the *d*-MPs for all possible integer *d* values. Given all the *d*-MPs, the reliability can be obtained by calculating the probability of the union of the *d*-MPs. There are two popular methods in existing literature, namely RSDP method and SSD method. In this chapter, we focus on RSDP method. The limitation of the RSDP method is introduced in Section 5.1. In Section 5.2, the concept of length is generalized for *d*-MPs in multistate networks. Two definitions of *length* are given, and four heuristic ordering methods, namely O1, O2, O3, and O4, are developed to improve the efficiency of the RSDP method reported in [39], by ordering the *d*-MPs before feeding them to the RSDP method. Section 5.3 investigates the efficiency of the heuristics by comparing the original RSDP method with the RSDP method incorporating heuristics O1, O2, O3, and O4 respectively. Section 5.4 extends the ordering heuristic methods to the situations when *d*-MCs instead of *d*-MPs are given for evaluations of the exact reliability. Section 5.5 summarizes the work. The materials in this chapter have been

documented in paper [47] and presented in the *6th Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modelling (APARM)* [48].

5.1 Introduction

For the evaluation of the exact reliability of multistate networks, the Recursive Sum of Disjoint Product (RSDP) method [39] is shown to be efficient. The basic idea of the RSDP method is based on the Sum of Disjoint Products (SDP) principle, and a specially defined maximum operator, denoted by $\oplus$ [39]. The probability of the union of *L* *d*-MPs can be evaluated via the probabilities of several unions each involving *L* -1 vectors or less. The main reason for the RSDP method to be efficient is that, even though in the starting minimal path (or minimal cut) vectors each vector does not dominate one another, the vectors generated from operator $\oplus$ may not have this property, in which the simplifying procedure can be applied by removing the vectors that are greater than others.

However, the reported RSDP method does not consider the ordering of the given *d*-MPs. Consider the example used in [39]. There are three 3-MPs given in the following order:

$$\mathbf{z}^1=(3,2,1,0,0,1),\mathbf{z}^2=(2,2,0,0,1,1),\mathbf{z}^3=(2,1,1,0,1,2).$$

Let $\mathbf{Y}^{i,j}=\mathbf{z}^i\oplus\mathbf{z}^j$, where $\mathbf{z}^i\oplus\mathbf{z}^j\equiv\left(\max\left(z_k^i,z_k^j\right)\right),1\le k\le n$. When implementing the RSDP method, the reliability of the network with respect to a demand *d* = 3 can be calculated as

$$\begin{aligned}&\Pr(\phi(\mathbf{x})\ge 3)=\Pr\left(\left\{\mathbf{x}\ge\mathbf{z}^1\right\}\cup\left\{\mathbf{x}\ge\mathbf{z}^2\right\}\cup\left\{\mathbf{x}\ge\mathbf{z}^3\right\}\right)\\&=\Pr\left(\mathbf{x}\ge\mathbf{z}^1\right)+\Pr\left(\mathbf{x}\ge\mathbf{z}^2\right)-\Pr\left(\mathbf{x}\ge\mathbf{Y}^{1,2}\right)+\Pr\left(\mathbf{x}\ge\mathbf{z}^3\right)-\Pr\left(\left\{\mathbf{x}\ge\mathbf{Y}^{1,3}\right\}\cup\left\{\mathbf{x}\ge\mathbf{Y}^{2,3}\right\}\right),\end{aligned}$$

where

$$\mathbf{Y}^{1,2}=\mathbf{z}^1\oplus\mathbf{z}^2=(3,2,1,0,1,1),\ \mathbf{Y}^{1,3}=\mathbf{z}^1\oplus\mathbf{z}^3=(3,2,1,0,1,2),\ \mathbf{Y}^{2,3}=\mathbf{z}^2\oplus\mathbf{z}^3=(2,2,1,0,1,2).$$

Because $\mathbf{Y}^{1,3}\ge\mathbf{Y}^{2,3}$, $\mathbf{Y}^{1,3}$ can be removed. Thus, we have

$$\Pr(\phi(\mathbf{x})\ge 3)=\Pr\left(\mathbf{x}\ge\mathbf{z}^1\right)+\Pr\left(\mathbf{x}\ge\mathbf{z}^2\right)-\Pr\left(\mathbf{x}\ge\mathbf{Y}^{1,2}\right)+\Pr\left(\mathbf{x}\ge\mathbf{z}^3\right)-\Pr\left(\mathbf{x}\ge\mathbf{Y}^{2,3}\right).$$

The advantage of the RSDP method is shown in this step because the following simplifying procedure has been implemented:

$$\Pr\left(\left\{\mathbf{x}\geq\mathbf{Y}^{1,3}\right\}\cup\left\{\mathbf{x}\geq\mathbf{Y}^{2,3}\right\}\right)=\Pr\left(\mathbf{x}\geq\mathbf{Y}^{2,3}\right).$$

However, if the order of the same three 3-MPs is given as

$$\mathbf{z}^1=(3,2,1,0,0,1),\mathbf{z}^3=(2,1,1,0,1,2),\mathbf{z}^2=(2,2,0,0,1,1),$$

then, we have

$$\begin{aligned}&\Pr\left(\phi(\mathbf{x})\geq 3\right)=\Pr\left(\left\{\mathbf{x}\geq\mathbf{z}^1\right\}\cup\left\{\mathbf{x}\geq\mathbf{z}^3\right\}\cup\left\{\mathbf{x}\geq\mathbf{z}^2\right\}\right)\\&=\Pr\left(\mathbf{x}\geq\mathbf{z}^1\right)+\Pr\left(\mathbf{x}\geq\mathbf{z}^3\right)-\Pr\left(\mathbf{x}\geq\mathbf{Y}^{1,3}\right)+\Pr\left(\mathbf{x}\geq\mathbf{z}^2\right)-\Pr\left(\left\{\mathbf{x}\geq\mathbf{Y}^{1,2}\right\}\cup\left\{\mathbf{x}\geq\mathbf{Y}^{2,3}\right\}\right).\end{aligned}$$

Because $\mathbf{Y}^{1,2}$ and $\mathbf{Y}^{2,3}$ cannot be merged using any simplifying procedure, the advantage of the RSDP method cannot be realized. Based on the observation above, we believe that a proper ordering of *d*-MPs is critical to the efficiency of the RSDP method.

In the literature, many reported works are dedicated to the SDP methods for evaluating the reliability of networks with binary components. Traldi [60] classified these SDP algorithms for binary systems into three stages. 1) Choose the order of the binary minimal paths. 2) Analyze the sum of disjoint events. 3) Analyze each individual disjoint event as a sum of disjoint products. In this paper, we focus on stage 1, which is the order of the *d*-MPs for the RSDP method to evaluate the reliability of networks with multistate state components. In terms of efficiency, Yeh [61] gave three criteria for the efficiency evaluation of SDP methods for the reliability evaluation of binary systems: the size of the results, i.e. the number of terms in disjoint sums; the computation time; and the form of the resulting formulae. Computation time is the direct measure of computational efficiency, and thus the most important criterion in comparing the efficiency of different reliability evaluation methods. Given that his first criterion was also used in many other studies, we investigate both of these criteria, the number of terms in disjoint sums and the computation time, for the RSDP method to evaluate the reliability of networks with multistate components.

Abraham [62] suggested an ordering heuristic based on increasing length (cardinality), where smaller minimal paths, that is, those with fewer components, should be ordered before larger ones. Based on the ordering suggestion in [62], Locks [63] suggested lexicographic ordering, where for a group of terms with the same length, the ordering is lexicographic, following the order of the symbols of the alphabet. Based on the ordering suggestion in [62], [63], Wilson [64] further suggested ordering based on hamming distance in such a way that two terms, which have the largest number of variables in common, are normally placed next to each other. A summary and comparison of different heuristic ordering methods for binary systems are given in Soh and Rai [65], [66], which concluded that ordering based on cardinality (length) or its combinations with lexicographic, or hamming distance-ordering, or both performs better than lexicographic ordering, and hamming distance-ordering in terms of computation time. Balan and Traldi [67] proposed another ordering strategy for binary MPs, claiming to be more efficient in terms of the number of terms in disjoint sums. The idea behind Balan and Traldi [67] was to allow violation of the increasing length by Abraham [62] on occasion. Based on the length of the MP and the relative complements of two MPs (the number of components that are contained in the first MP but not in the second MP), they used a score function given by a multiplication of a factor and sum. The factor gives an advantage to the MP with a shorter length, and the sum gives a weighted average of relative complements, favoring those, which come from shorter already listed minimal paths.

Given all the reported ordering heuristics for binary systems, we intend to adopt these ideas to order the *d*-MPs for our multistate network reliability evaluation for the RSDP method. However, the concept of length for MP, used by [62] and [67], is not suitable for *d*-MP in a multistate system. Thus, it is necessary to generalize the concept of length for *d*-MP in a multistate system first, and then adopt the idea to order the *d*-MPs for the multistate network reliability evaluation using the RSDP method.

5.2 Development of ordering heuristics

5.2.1 The definition of length for a *d*-MP

The length defined in binary systems is the number of components contained in each minimal path. Given the vector representation of a minimal path defined by assigning 1 for the components that are contained in the minimal path and 0 otherwise, the length of each minimal path is given by counting the 1 entries. Because there are other integers that are higher than 1 contained in the *d*-MP, one way of defining the length for a *d*-MP is given as follows.

First, we define the highest state vector. Given all *L* *d*-MPs, denoted as $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^L$, we have the highest state vector defined as

$$\mathbf{ZH}^1 = \mathbf{z}^1 \oplus \mathbf{z}^2 \oplus, \ldots, \oplus \mathbf{z}^L,$$

where $\oplus$ is the same maximum operator defined in [39]. For example, given the *d*-MPs

$$\mathbf{z}^1 = (3,2,1,0,0,1), \mathbf{z}^2 = (2,2,0,0,1,1), \mathbf{z}^3 = (2,1,1,0,1,2),$$

we have $\mathbf{ZH}^1 = (3,2,1,0,1,2)$.This indicates that the highest state is 3 for component 1, 2 for component 2, 1 for component 3, 0 for component 4, 1 for component 5, and 2 for component 6.

Given the definition of the highest state vector, we can also define the second highest state vector, $\mathbf{ZH}^2$, which contains the second largest state among all given *d*-MPs for each component. For example, the second component contains states 2, 2, and 1 for $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$ respectively. Given the highest state is 2, the second highest is then 1. If there is no such a state for a component, we use 0 for convenience. For the same example, we have $\mathbf{ZH}^2 = (2,1,0,0,0,1)$. We can also find the third highest state for each component, and so forth. If there is no such a state for a component, we again use 0 for convenience. The process stops when all the maximum values for the kth highest state for all the components are equal to 0. Then the lowest state vector is $\mathbf{ZH}^{k-1}$.

Given the definition of the highest state vector, the second highest state vector, etc., we give the following definition of the length for a *d*-MP.

Definition 1 *Let* a^k *denote the number of elements that a d-MP has in common with the* kth *highest state vector* $\mathbf{ZH}^k$. a^k *is defined as the length of this d-MP with respect to the* kth *highest state.*

For example, given the following *d*-MPs

$$\mathbf{z}^1=(3,2,1,0,0,1),\mathbf{z}^2=(2,2,0,0,1,1),\mathbf{z}^3=(2,1,1,0,1,2),$$

the highest state vector, and the second highest state vector are respectively

$$\mathbf{ZH}^1=(3,2,1,0,1,2), \text{ and } \mathbf{ZH}^2=(2,1,0,0,0,1).$$

There is no 3rd highest state vector because the third highest state for each component is 0. Then the length of $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$ with respect to the highest state is 3, 2, and 3, respectively; the length of $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$ with respect to the second highest state is 1, 2, and 2, respectively. Note that if each component can only take two possible states, i.e. the binary case, this length definition reduces to the length definition for a binary MP [62].

Length Definition 1 of the *d*-MP generates a sequence of lengths. An alternative way of defining the length of *d*-MP is to give a single length, given in length Definition 2 in the remainder of this subsection.

First, we obtain the logically equivalent vector for each *d*-MP by eliminating all state values that appeared in no *d*-MPs. For all the state values of each component in the given *d*-MPs, the lowest nonzero state value is replaced by 1 in the logically equivalent vector; the second lowest nonzero state value is replaced by 2, and so forth. For example, given the *d*-MPs $\mathbf{z}^1=(3,2,1,0,0,1)$, $\mathbf{z}^2=(2,2,0,0,1,1)$, and $\mathbf{z}^3=(2,1,1,0,1,2)$, the first component contains states 3, 2, and 2 for $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$, respectively. Given that state 2 is the lowest nonzero state, state 2 is replaced by 1 in the logically equivalent vector. Similarly, state 3, which is the second lowest

nonzero state, is replaced by 2 in the corresponding logically equivalent vector. The logically equivalent vectors for $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$ are given as follows:

$$\overline{\mathbf{z}}^1=(2,2,1,0,0,1),\overline{\mathbf{z}}^2=(1,2,0,0,1,1),\overline{\mathbf{z}}^3=(1,1,1,0,1,2).$$

Given the logically equivalent vector for each *d*-MP, we give the following length definition for *d*-MP.

Definition 2 *The length of a d-MP is the sum of the values in its logically equivalent vector.*

For example, given the *d*-MPs

$$\mathbf{z}^1=(3,2,1,0,0,1),\mathbf{z}^2=(2,2,0,0,1,1),\mathbf{z}^3=(2,1,1,0,1,2),,$$

and logically equivalent vectors

$$\overline{\mathbf{z}}^1=(2,2,1,0,0,1),\overline{\mathbf{z}}^2=(1,2,0,0,1,1),\overline{\mathbf{z}}^3=(1,1,1,0,1,2),$$

the length of $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$ is 6, 5, and 6, respectively.

Note that, if each component can only take two possible states, i.e. the binary case, this length definition reduces to the length definition for a binary MP [62].

5.2.2 Analysis of the RSDP method

We summarized the RSDP method as follows [39]. Given all *L d*-MPs, denoted as $\mathbf{z}^1,\mathbf{z}^2,\ldots,\mathbf{z}^L$, the reliability can be calculated via the following recursive function.

$$\Pr(\phi(\mathbf{x})\geq d)\equiv\Pr\left(\{\mathbf{x}\geq\mathbf{z}^1\}\cup\{\mathbf{x}\geq\mathbf{z}^2\}\cup\cdots\cup\{\mathbf{x}\geq\mathbf{z}^L\}\right)\equiv\Pr\left(\bigcup_{i=1}^{L}\mathbf{x}\geq\mathbf{z}^i\right) \tag{5.1}$$

Following the SDP principle, the following recursive algorithm is used.

$$\Pr U(\mathbf{z}^1,\mathbf{z}^2,\ldots,\mathbf{z}^L)\equiv\sum_{i=1}^{L}\mathrm{TM}_i, \tag{5.2}$$

where TM_i is the ith term in the SDP calculation [6] given by

$\mathrm{TM}_1=\Pr(\mathbf{x}\geq\mathbf{z}^1)$.

$$\begin{aligned}\mathrm{TM_i} &= \Pr\left(\mathbf{x} \geq \mathbf{z}^{\mathrm{i}}\right) - \Pr\left(\bigcup_{\mathrm{j}=1}^{\mathrm{i}-1} \mathbf{x} \geq \mathbf{Y}^{\mathrm{i,j}}\right) \\ &= \Pr\left(\mathbf{x} \geq \mathbf{z}^{\mathrm{i}}\right) - \Pr U\left(\mathbf{Y}^{1,\mathrm{i}}, \ldots, \mathbf{Y}^{\mathrm{i}-1,\mathrm{i}}\right), \text{ for } \mathrm{i} \geq 2.\end{aligned}$$

The simplifying procedure is used whenever in each function $\Pr U(\bullet)$ there exists an input vector $\mathbf{y}^{\mathrm{i}}$ satisfying $\mathbf{y}^{\mathrm{i}} \geq \mathbf{y}^{\mathrm{j}} (\mathrm{i} \neq \mathrm{j})$, in which case $\mathbf{y}^{\mathrm{i}}$ can be deleted from the set of input vectors. The simplifying procedure is used during the process of comparing each pair of *d*-MPs. Although the number of comparisons involved for each *d*-MP is the same among all the $\Pr U(\bullet)$, the number of occurrences in all the comparisons for each *d*-MP is different. The first heuristic is given below.

Property 1 *– During the process of the RSDP method, a d-MP that is ordered in a later position is expected to have more occurrences in all the comparisons, when the total number of d-MPs is greater than or equal to 3.*

The comparison takes place in $\Pr U\left(\mathbf{Y}^{1,i}, \ldots, \mathbf{Y}^{i-1,i}\right)$ when $i \geq 3$, where the number of comparisons is 1 for $i = 3$ ($\mathbf{Y}^{1,3}$ and $\mathbf{Y}^{2,3}$). The number of comparisons increases to 3 when $i = 4$ ($\mathbf{Y}^{1,4}$ and $\mathbf{Y}^{2,4}$, $\mathbf{Y}^{1,4}$ and $\mathbf{Y}^{3,4}$, and $\mathbf{Y}^{2,4}$ and $\mathbf{Y}^{3,4}$), and increases to $\binom{L-1}{2}$ when $i = L$. Thus the number of occurrences for each *d*-MP in all the comparisons is influenced by its order. That is, $\mathbf{z}^1$ has $\sum_{i=1}^{L-2} i = \left(L^2 - 3L + 2\right)/2$ occurrences in all comparisons because it will appear once in each $\Pr U\left(\mathbf{Y}^{1,i}, \ldots, \mathbf{Y}^{i-1,i}\right), i > 2$; $\mathbf{z}^2$ also has $\left(L^2 - 3L + 2\right)/2$ occurrences in all comparisons, because it also appears once in each $\Pr U\left(\mathbf{Y}^{1,i}, \ldots, \mathbf{Y}^{i-1,i}\right), i > 2$. For each $\mathbf{z}^{\mathrm{j}}, j \geq 3$, it will appear once in each $\Pr U\left(\mathbf{Y}^{1,\mathrm{i}}, \ldots, \mathbf{Y}^{\mathrm{i}-1,\mathrm{i}}\right), i > j$, which is $\sum_{i=1}^{L-j} j + i - 2 = \left(L - j\right) \times \left(L + j - 3\right)/2$ occurrences, and $2 \times \binom{j-1}{2}$ occurrences in $\Pr U\left(\mathbf{Y}^{1,\mathrm{i}}, \ldots, \mathbf{Y}^{\mathrm{i}-1,\mathrm{i}}\right), i = j$. Thus, $\mathbf{z}^3$ has a total of $\left(L^2 - 3L + 4\right)/2$

occurrences in all comparisons, and so on. Generally $\mathbf{z}^j$ has a total of $(L-j)\times(L+j-3)/2+2\times\binom{j-1}{2}$ occurrences in all comparisons, and $\mathbf{z}^L$ has a total of $2\times\binom{L-1}{2}$ occurrences in all comparisons. Except $\mathbf{z}^1$ and $\mathbf{z}^2$, a *d*-MP ordered in a later position has a larger number of occurrences in all the comparisons when the total number of *d*-MPs is greater than or equal to 3. For example, given $L=4$, for the third *d*-MP, there are 4 comparisons: $\mathbf{Y}^{1,3}$ and $\mathbf{Y}^{2,3}$, $\mathbf{Y}^{1,4}$ and $\mathbf{Y}^{2,4}$, $\mathbf{Y}^{1,4}$ and $\mathbf{Y}^{3,4}$, and $\mathbf{Y}^{2,4}$ and $\mathbf{Y}^{3,4}$. Thus, for $L=4$, $\mathbf{z}^1$ and $\mathbf{z}^2$ has 3 occurrences, $\mathbf{z}^3$ has 4 occurrences, and $\mathbf{z}^4$ has 6 occurrences.

Because a *d*-MP, when ordered in a later position, has a larger number of occurrences in all the comparisons, the next question becomes which *d*-MPs should be put into later positions, i.e., which *d*-MP is more likely to evoke the simplifying procedure. For example, given the *d*-MPs $\mathbf{z}^1=(3,2,1,0,0,1)$, $\mathbf{z}^2=(2,2,0,0,1,1)$, and $\mathbf{z}^3=(2,1,1,0,1,2)$, the length of $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$ with respect to the highest state is 3, 2, and 3 respectively under Definition 1. The length of $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$ is 6, 5, and 6 respectively, under Definition 2. Then, given $\mathbf{z}^1$, $\mathbf{z}^2$, and $\mathbf{z}^3$, we have

$$\mathbf{z}^1\oplus\mathbf{z}^3=(3,2,1,0,1,2),\ \mathbf{z}^2\oplus\mathbf{z}^3=(2,2,1,0,1,2),\ \mathbf{z}^1\oplus\mathbf{z}^2=(3,2,1,0,1,1),$$

where $\mathbf{z}^1\oplus\mathbf{z}^3\geq\mathbf{z}^2\oplus\mathbf{z}^3$ and $\mathbf{z}^1\oplus\mathbf{z}^3\geq\mathbf{z}^1\oplus\mathbf{z}^2$. Thus, placing either $\mathbf{z}^1$ or $\mathbf{z}^3$ to the last place in order can evoke the simplifying procedure. Based on the observation above, we summarize the second heuristic.

Property 2 *– A d-MP with a longer length is expected to evoke more simplifying procedures, and thus should be placed in a later position.*

5.2.3 The ordering heuristic O1

Given the length for a *d*-MP under Definition 1 in Section 5.2.1, and the two properties for the RSDP method in Section 5.2.2, we propose the following ordering heuristic O1.

1) *Given all the d-MPs, obtain the highest state vector, the second highest state vector, …, and the n_{th} highest state vector.*
2) *For each d-MP, obtain the length with respect to the highest state, the second highest state, …, and the n_{th} highest state using Definition 1.*
3) *Order the d-MPs in ascending order according to their lengths with respect to the highest state.*
4) *For those d-MPs with the same length with respect to the highest state, order those d-MPs in ascending order according to their lengths with respect to the second highest state.*
5) *For those d-MPs with the same length with respect to the second highest state, order those d-MPs in ascending order according to their lengths with respect to the third highest state.*

This process continues in a similar manner up to the $(n-1)_{th}$ highest state. Whenever there is a tie, the tie is broken arbitrarily.

As can be seen from the procedure above, the ordering heuristic O1 is a kind of combination of cardinality ordering and lexicographic ordering, in which the cardinality refers to the length for *d*-MP under Definition 1*,* and lexicographic refers to the level of state. We illustrate the ordering heuristic O1 through the illustrative example used in [39]. There are five 2-MPs given in an arbitrary order:

$$\mathbf{z}^1=(1,2,0,1,1,0),\mathbf{z}^2=(2,2,0,0,0,0),\mathbf{z}^3=(1,0,1,0,1,2),\mathbf{z}^4=(1,1,0,0,1,1),\mathbf{z}^5=(2,1,1,0,0,1).$$

1) The highest state vector, and the second highest state vector are respectively

$$\mathbf{ZH}^1=(2,2,1,1,1,2), \text{ and } \mathbf{ZH}^2=(1,1,0,0,0,1).$$

2) The length of each 2-MP under Definition 1 can be calculated.
3) Order the *d*-MPs in ascending order according to their lengths with respect to the highest state.

4) For those *d*-MPs with the same length with respect to the highest state, order those *d*-MPs in ascending order according to their lengths with respect to the second highest state, and obtain the results in Table 5.1.

Table 5.1: Ordering result based on O1

Final Order	*d*-MP	Component State Vector						Length with respect to $\mathbf{ZH}^1$	Length with respect to $\mathbf{ZH}^2$
1	$\mathbf{z}^4$	1	1	0	0	1	1	1	3
2	$\mathbf{z}^2$	2	2	0	0	0	0	2	0
3	$\mathbf{z}^5$	2	1	1	0	0	1	2	2
4	$\mathbf{z}^1$	1	2	0	1	1	0	3	1
5	$\mathbf{z}^3$	1	0	1	0	1	2	3	1

Because there is no third highest state vector, we generate the final order. Note that if each component can only take two possible states, i.e. the binary case, O1 reduces to the ordering heuristic in [62].

5.2.4 The ordering heuristic O2

Given Definition 2 of the length for *d*-MP in Section 5.2.1, and the two properties for the RSDP method in Section 5.2.2, we give the following ordering heuristic O2.

1) For each d-MP, obtain its logically equivalent vector.

2) Obtain the length of each d-MP using Definition 2.

3) Order the d-MPs in ascending order with respect to their lengths.

During the process, whenever there is a tie, the tie is broken arbitrarily.

We illustrate the ordering heuristic O2 through the same illustrative example used in [39]. There are five 2-MPs given in an arbitrary order:

$$\mathbf{z}^1=(1,2,0,1,1,0),\mathbf{z}^2=(2,2,0,0,0,0),\mathbf{z}^3=(1,0,1,0,1,2),\mathbf{z}^4=(1,1,0,0,1,1),\mathbf{z}^5=(2,1,1,0,0,1).$$

1) Obtain the logically equivalent vectors as

$$\overline{\mathbf{z}}^1=(1,2,0,1,1,0),\overline{\mathbf{z}}^2=(2,2,0,0,0,0),\overline{\mathbf{z}}^3=(1,0,1,0,1,2),\overline{\mathbf{z}}^4=(1,1,0,0,1,1),\overline{\mathbf{z}}^5=(2,1,1,0,0,1).$$

2) The length of the $\mathbf{z}^1$, $\mathbf{z}^2$, $\mathbf{z}^3$, $\mathbf{z}^4$, and $\mathbf{z}^5$ are 5, 4, 5, 4, and 5 respectively under Definition 2.

3) Order the *d*-MPs in ascending order according to the length. After ordering, the final order can be generated as in Table 5.2.

Table 5.2: Ordering result based on O2

Final Order	*d*-MP	Component State Vector						Length by Definition 2
1	$\mathbf{z}^4$	1	1	0	0	1	1	4
2	$\mathbf{z}^2$	2	2	0	0	0	0	4
3	$\mathbf{z}^5$	2	1	1	0	0	1	5
4	$\mathbf{z}^1$	1	2	0	1	1	0	5
5	$\mathbf{z}^3$	1	0	1	0	1	2	5

Note that, if each component can only take two possible states, i.e. the binary case, then O2 reduces to the ordering heuristic in [62].

5.2.5 The ordering heuristic O3

O1 provides an order heuristic based on Definition 1 for the length of a *d*-MP, by considering the length with respect to different states one after another. Another way to utilize these length values is to combine them into a single length measure by assigning weights to different states. Motivated by this idea, based on length Definition 1, first we will define a score function.

Let $a_i^1, a_i^1, \ldots, a_i^n$ denote the length values with respect to the highest state, the second highest state, and on to the n_{th} highest state, respectively, for a *d*-MP, $\mathbf{z}^i$. The score function is defined as:

$$F(\mathbf{z}^i)=\sum_{k=1}^{n} a_i^k \times 2^{(n-k)}. \tag{5.3}$$

Given the length of a *d*-MP by Definition 1, and the score function, we give the ordering heuristic O3 as follows.

1) *For d-MPs,* $\mathbf{z}^i, i=1,\cdots,L$*, obtain their highest state vector, second highest state vector, …, and the* n_{th} *highest state vector.*
2) *For each d-MP* $\mathbf{z}^i, i=1,\cdots,L$*, obtain its length with respect to the highest state, the second highest state, …, and the* n_{th} *highest state using Definition 1.*
3) *Obtain its score function value using equation (5.3).*
4) *Order the d-MPs in ascending order according to their score function values.*

   During the process, whenever there is a tie, the tie is broken arbitrarily.

We illustrate the ordering heuristic O3 through the illustrative example used before. There are five 2-MPs given in an arbitrary order:

$$\mathbf{z}^1=(1,2,0,1,1,0),\mathbf{z}^2=(2,2,0,0,0,0),\mathbf{z}^3=(1,0,1,0,1,2),\mathbf{z}^4=(1,1,0,0,1,1),\mathbf{z}^5=(2,1,1,0,0,1).$$

Table 5.3: Ordering result based on O3

Final Order	*d*-MP	Component State Vector						Length with respect to $\mathbf{ZH}^1$	Length with respect to $\mathbf{ZH}^2$	Score
1	$\mathbf{z}^2$	2	2	0	0	0	0	2	0	7
2	$\mathbf{z}^4$	1	1	0	0	1	1	1	3	4
3	$\mathbf{z}^5$	2	1	1	0	0	1	2	2	7
4	$\mathbf{z}^1$	1	2	0	1	1	0	3	1	5
5	$\mathbf{z}^3$	1	0	1	0	1	2	3	1	6

1) The highest state vector, and the second highest state vector are

   $\mathbf{ZH}^1=(2,2,1,1,1,2)$, and $\mathbf{ZH}^2=(1,1,0,0,0,1)$.
2) The length of each 2-MP under Definition 1 can be calculated.
3) Given the length, obtain the score of each *d*-MP using the score function.

4) Order the *d*-MPs in ascending order according to their score function values. The final order of all *d*-MPs is shown in Table 5.3.

5.2.6 The ordering heuristic O4

In this ordering heuristic O4, in addition to ordering *d*-MPs based on their lengths, the relationships and similarities among different *d*-MPs are also considered. This consideration makes similar *d*-MPs placed close to one another, and increases the likelihood of evoking more simplifying procedures. The Ordering heuristic O4 is based on the concept of relative complements between two binary MPs and the ordering method reported in [67]. By extending the relative complements concept in [67] to multistate networks, we define the relative difference for two *d*-MPs, $\mathbf{z}^i$ and $\mathbf{z}^j$, as follows.

Given the logically equivalent vectors of $\mathbf{z}^i$ and $\mathbf{z}^j$, *and* $\overline{\mathbf{z}}^i$ and $\overline{\mathbf{z}}^j$ respectively, the relative difference for two *d*-MPs $\mathbf{z}^i$ and $\mathbf{z}^j$ is

$$\left|\overline{\mathbf{z}}^i - \overline{\mathbf{z}}^j\right| = \sum_{k=1}^{\mathrm{n}} \left\{ \overline{\mathrm{x}}_i^k - \overline{\mathrm{x}}_j^k \mid \overline{\mathrm{x}}_i^k > \overline{\mathrm{x}}_j^k \right\}, \tag{5.4}$$

which is the summation of the differences between the values of each component for $\overline{\mathbf{z}}^i$ and $\overline{\mathbf{z}}^j$ respectively, given the value of the component from $\overline{\mathbf{z}}^i$ is bigger than the value of the same component from $\overline{\mathbf{z}}^j$. Given the length of the *d*-MP under Definition 2, and the relative difference for two *d*-MPs, we give the ordering heuristic O4 described as follows.

1) *For each d-MP, obtain its logically equivalent vector and its length using Definition 2.*
2) *Order the d-MPs in ascending order according to their length.*
3) *Given* $\mathbf{z}^L$ *has been placed in order, the next* $\mathbf{z}^{L\text{-}1}$ *is chosen so that* $\left|\overline{\mathbf{z}}^{L\text{-}1} - \overline{\mathbf{z}}^L\right| \Big/ \left|\mathbf{z}^L\right|$ *is the largest, where* $\left|\mathbf{z}^L\right|$ *is the length of d-MP*$_L$ *according to Definition 2, and* $\left|\overline{\mathbf{z}}^{L\text{-}1} - \overline{\mathbf{z}}^L\right|$ *is the relative difference between the logically equivalent vectors of d-MP*$_{L\text{-}1}$ *and d-MP*$_L$.

4) *Given* $\mathbf{z}^{L}$ *and* $\mathbf{z}^{L\text{-}1}$ *have been placed in order,* $\mathbf{z}^{L\text{-}2}$ *is chosen so that* $\sum_{k=0}^{j\text{-}1}\left|\overline{\mathbf{z}}^{L-j}-\overline{\mathbf{z}}^{L-k}\right| \Big/ \left|\mathbf{z}^{L-k}\right|$ *is the largest, where* $j=2$.

5) *Given* $\mathbf{z}^{L}$, $\mathbf{z}^{L\text{-}1}$, *and* $\mathbf{z}^{L\text{-}2}$ *have been placed in order,* $\mathbf{z}^{L\text{-}3}$ *is chosen so that* $\sum_{k=0}^{j\text{-}1}\left|\overline{\mathbf{z}}^{L-j}-\overline{\mathbf{z}}^{L-k}\right| \Big/ \left|\mathbf{z}^{L-k}\right|$ *is the largest, where* $j=3$.

This process continues until $\mathbf{z}^{2}$ *is chosen. Whenever there is a tie, the tie is broken arbitrarily.*

We illustrate the ordering heuristic O4 through the same illustrative example used in [39]. There are five 2-MPs given in an arbitrary order:

$$\mathbf{z}^{1}=(1,2,0,1,1,0),\mathbf{z}^{2}=(2,2,0,0,0,0),\mathbf{z}^{3}=(1,0,1,0,1,2),\mathbf{z}^{4}=(1,1,0,0,1,1),\mathbf{z}^{5}=(2,1,1,0,0,1).$$

1) Their logically equivalent vectors are given as

$$\overline{\mathbf{z}}^{1}=(1,2,0,1,1,0),\overline{\mathbf{z}}^{2}=(2,2,0,0,0,0),\overline{\mathbf{z}}^{3}=(1,0,1,0,1,2),\overline{\mathbf{z}}^{4}=(1,1,0,0,1,1),\overline{\mathbf{z}}^{5}=(2,1,1,0,0,1).$$

The lengths of $\mathbf{z}^{1}$, $\mathbf{z}^{2}$, $\mathbf{z}^{3}$, $\mathbf{z}^{4}$, and $\mathbf{z}^{5}$ are 5, 4, 5, 4, and 5 respectively under Definition 2. $L=5$ in this example.

2) Order the *d*-MPs in ascending order according to their length. After ordering, we obtain the results in Table 5.4.

Table 5.4: Ordering result after step 2 of O4

Current Order	*d*-MP	Component State Vector						Length by Definition 2
1	$\mathbf{z}^{4}$	1	1	0	0	1	1	4
2	$\mathbf{z}^{2}$	2	2	0	0	0	0	4
3	$\mathbf{z}^{5}$	2	1	1	0	0	1	5
4	$\mathbf{z}^{1}$	1	2	0	1	1	0	5
5	$\mathbf{z}^{3}$	1	0	1	0	1	2	5

3) Given that *d*-MP$_5$ is $\mathbf{z}^3$, *d*-MP$_4$ is chosen so that $\left|\overline{\mathbf{z}}^{L-1} - \overline{\mathbf{z}}^{L}\right| \Big/ \left|\mathbf{z}^{L}\right|$ is the largest, where $\mathbf{z}^L$ is $\mathbf{z}^3$.

We obtain the values given in Table 5.5, and z^1 is chosen as the vector *d*-MP$_4$ before $\mathbf{z}^3$.

Table 5.5: Ordering result after *d*-MP$_4$ is chosen

Current Order	*d*-MP	Component State Vector	$\left\|\overline{\mathbf{z}}^{L-1} - \overline{\mathbf{z}}^{L}\right\| / \left\|\mathbf{z}^{L}\right\|$
1	$\mathbf{z}^4$	1 1 0 0 1 1	1/5
2	$\mathbf{z}^2$	2 2 0 0 0 0	3/5
3	$\mathbf{z}^5$	2 1 1 0 0 1	2/5
4	$\mathbf{z}^1$	1 2 0 1 1 0	3/5
5	$\mathbf{z}^3$	1 0 1 0 1 2	

4) Given *d*-MP$_5$ and *d*-MP$_4$, *d*-MP$_3$ is chosen so that $\sum_{k=0}^{j-1} \left|\overline{\mathbf{z}}^{L-j} - \overline{\mathbf{z}}^{L-k}\right| \Big/ \left|\mathbf{z}^{L-k}\right|$ is the largest, where $j = 2$. We obtain the results in Table 5.6, and $\mathbf{z}^5$ is chosen as *d*-MP$_3$ to be placed before $\mathbf{z}^1$.

Table 5.6: Ordering result after *d*-MP$_3$ is chosen

Current Order	*d*-MP	Component State Vector	$\sum_{k=0}^{j-1} \left\|\overline{\mathbf{z}}^{L-j} - \overline{\mathbf{z}}^{L-k}\right\| / \left\|\mathbf{z}^{L-k}\right\|$
1	$\mathbf{z}^4$	1 1 0 0 1 1	2/5
2	$\mathbf{z}^2$	2 2 0 0 0 0	4/5
3	$\mathbf{z}^5$	2 1 1 0 0 1	5/5
4	$\mathbf{z}^1$	1 2 0 1 1 0	
5	$\mathbf{z}^3$	1 0 1 0 1 2	

5) Given *d*-MP$_5$, *d*-MP$_4$, and *d*-MP$_3$, *d*-MP$_2$ is chosen so that $\sum_{k=0}^{j-1} \left|\overline{\mathbf{z}}^{L-j} - \overline{\mathbf{z}}^{L-k}\right| \Big/ \left|\mathbf{z}^{L-k}\right|$ is the largest, where $j = 3$. The ordering heuristic is finished with the ordering in Table 5.7.

Table 5.7: Final ordering result based on O4

Final Order	*d*-MP	Component State Vector						$\sum_{k=0}^{j-1}\left\|\overline{\mathbf{z}}^{L-j}-\overline{\mathbf{z}}^{L-k}\right\| \Big/ \left\|\mathbf{z}^{L-k}\right\|$
1	$\mathbf{z}^4$	1	1	0	0	1	1	3/5
2	$\mathbf{z}^2$	2	2	0	0	0	0	5/5
3	$\mathbf{z}^5$	2	1	1	0	0	1	
4	$\mathbf{z}^1$	1	2	0	1	1	0	
5	$\mathbf{z}^3$	1	0	1	0	1	2	

5.3 Efficiency investigation of the four ordering heuristics

In this section, we compare the efficiency of the RSDP method [39] with the approach of applying the ordering heuristics O1, O2, O3, and O4 to the RSDP method. The programs were developed using MATLAB 2011, and were implemented on a server with 2 AMD 2.3 GHz CPU (12% CPU was allocated for each user), and 16 GB of RAM. In terms of efficiency, we are interested in the required computation time, and the number of terms in disjoint sums [61], with respect to the number of components *n* and the number of *d*-MPs. We adopt the same efficiency investigation setting in [39]. We consider a hypothetical multistate network system with *n* components. Each component has ten states, from 0 to 9; and the state distributions of all the components are identical, denoted as

$$p=\left(0.05,0.15,0.1,0.05,0.15,0.05,0.15,0.1,0.05,0.15\right).$$

We randomly generate *L* vectors, and ensure that no vector is dominated. As a result, these *L* vectors can be treated as *L* *d*-MPs of some hypothetical network. We investigate 10 groups, with *L* randomly generated *d*-MPs in each group. Each group is processed with the original RSDP method, and the RSDP method incorporating one of the four heuristics, namely, O1, O2,

O3, and O3, proposed in this paper. The CPU time needed to process each group and the number of terms in the resulting disjoint sums for each method are recorded. Considering all 10 groups, we then calculate the average CPU time needed to process a group of such *d*-MPs using the original RSDP method, the resulting average number of terms in the resulting disjoint sums, and the average CPU time needed to process the same group of such *d*-MPs using the RSDP method with O1, O2, O3, and O4 ordering heuristics, respectively, and the resulting average number of terms in the resulting disjoint sums. A CPU time ratio is calculated by dividing the average CPU time with the original RSDP method by the average CPU time with the RSDP method incorporating one of the four heuristics. A ratio of the number of terms is calculated by dividing the average number of terms obtained with the original RSDP method by the average number of terms obtained with the updated RSDP method incorporating one of the four heuristics. These ratios represent the advantage of these four ordering heuristics over the original RSDP method which has arbitrary ordering of the *L d*-MPs.

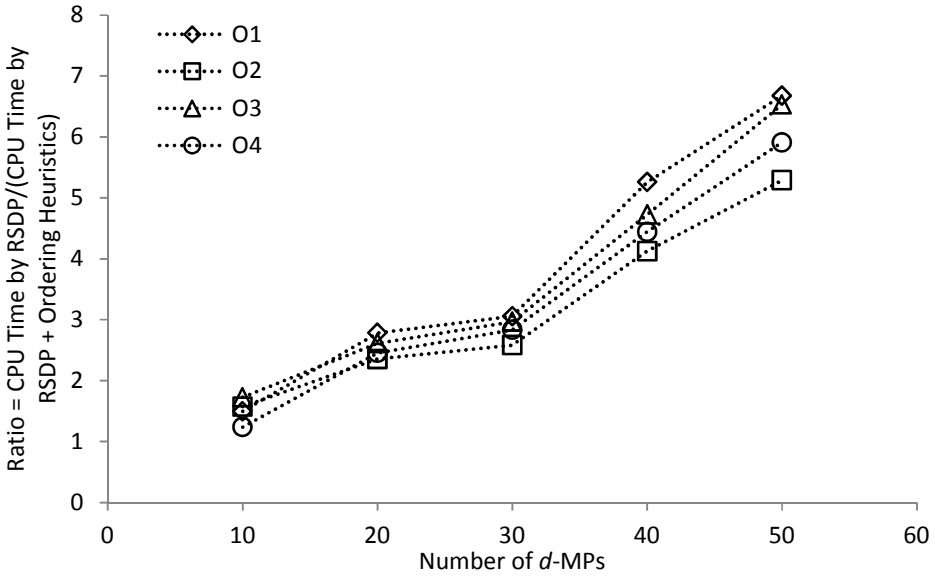

Figure 5.1: Ratio of the mean computation times for 10 components.

First, we investigate multistate networks with 10 and 20 components. For each case, we consider different scenarios with 10, 20, 30, 40, and 50 *d*-MPs. The mean CPU time ratio, and

the mean number of terms ratio for 10 components are shown in Figure 5.1 and Figure 5.2 respectively. The mean CPU time ratio, and the mean number of terms ratio for 20 components are shown in Figure 5.3, and Figure 5.4 respectively.

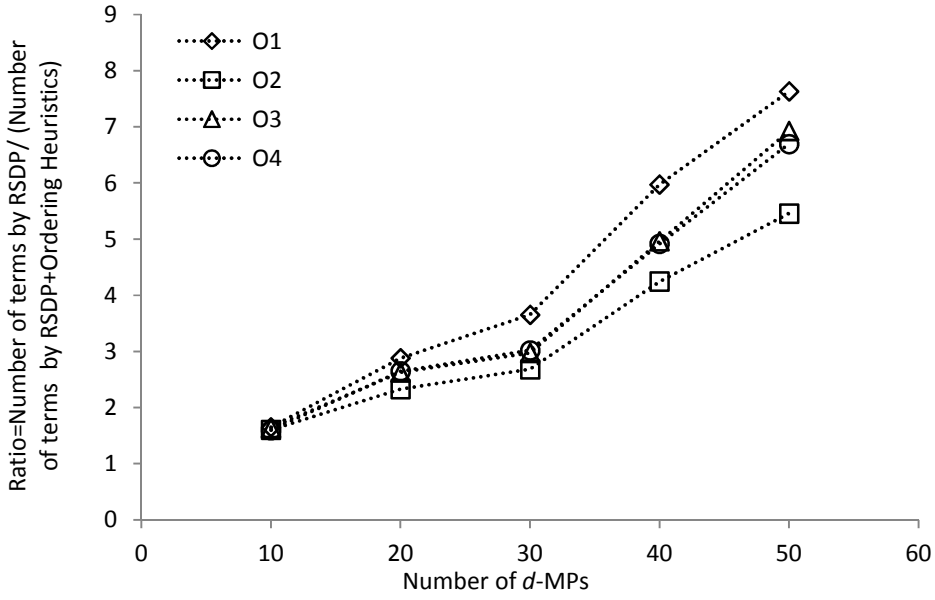

Figure 5.2: Ratio of the mean number of terms for 10 components.

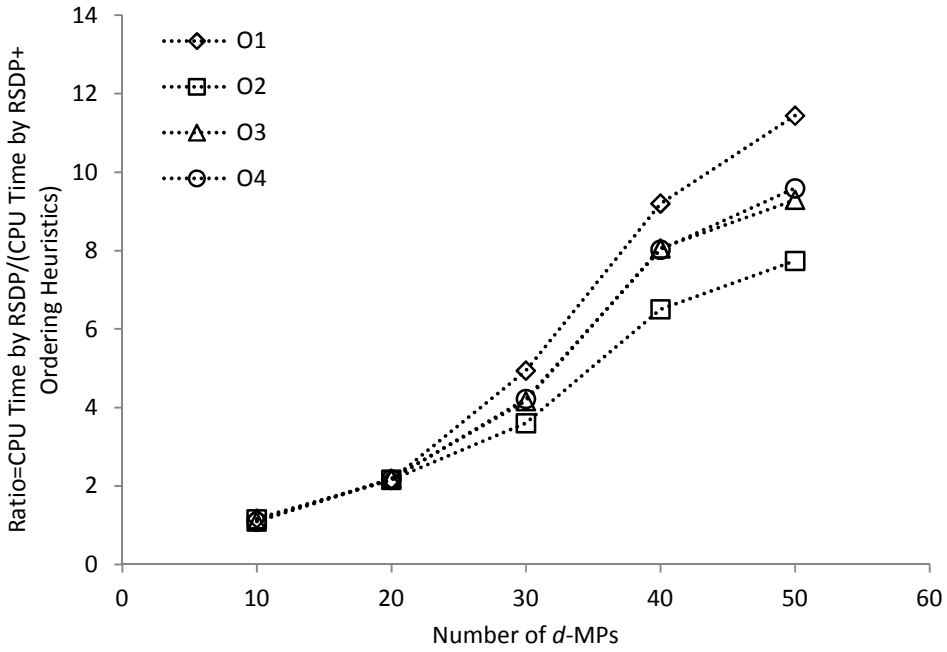

Figure 5.3: Ratio of the mean computation times for 20 components.

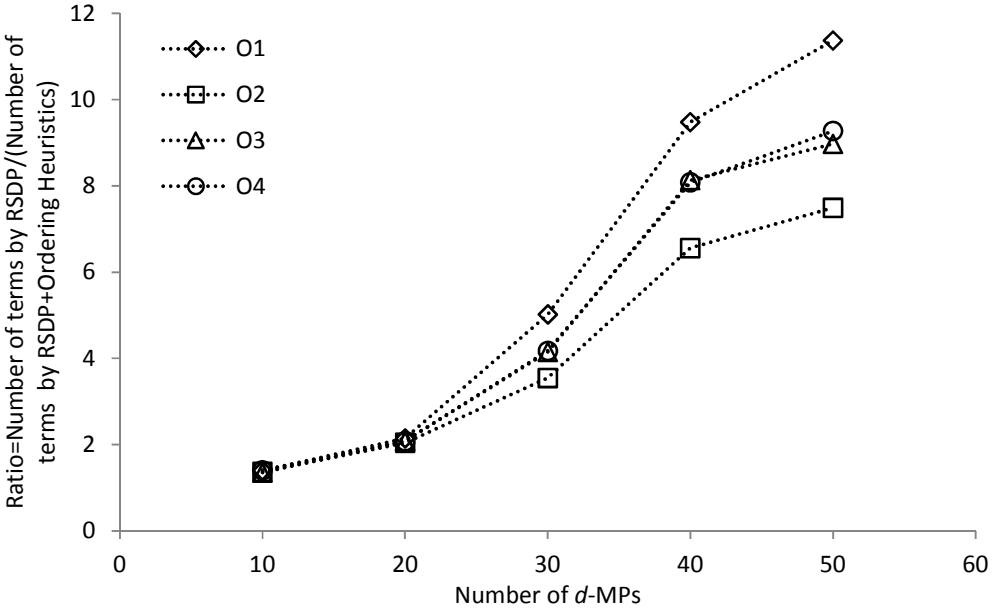

Figure 5.4: Ratio of the mean number of terms for 20 components.

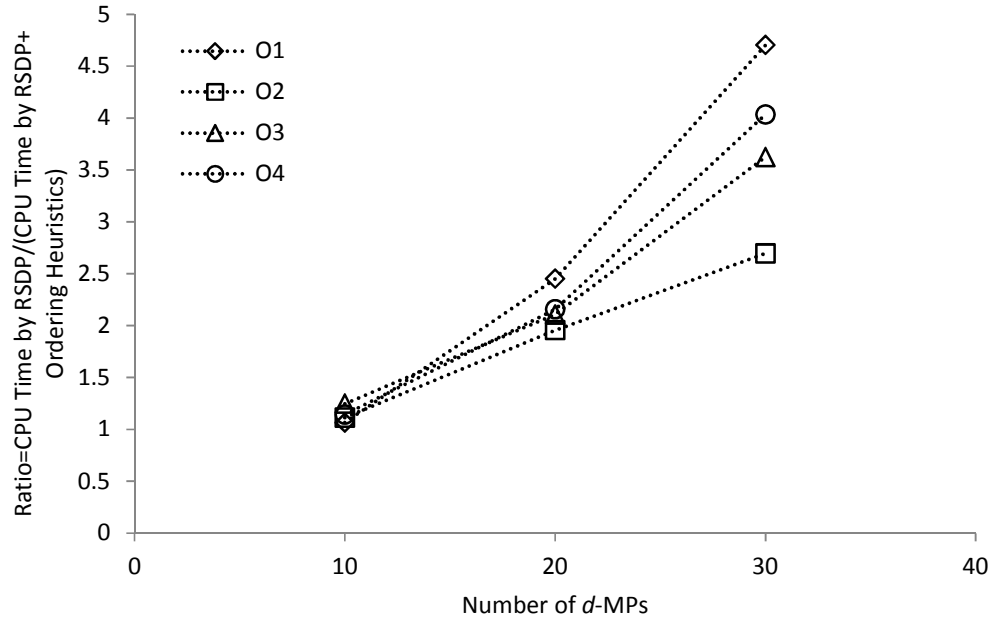

Figure 5.5: Ratio of the mean computation times for 30 components.

Next, we investigate multistate networks with 30 and 40 components. Because more CPU time is consumed, and more terms in the disjoint sums are generated with a larger number of components [39], for each case, we consider different scenarios with 10, 20, and 30 *d*-MPs. The

mean CPU time ratio, and the mean number of terms ratio for 30 components are shown in Figure 5.5 and Figure 5.6 respectively. The mean CPU time ratio, and the mean number of terms ratio for 40 components are shown in Figure 5.7, and Figure 5.8 respectively.

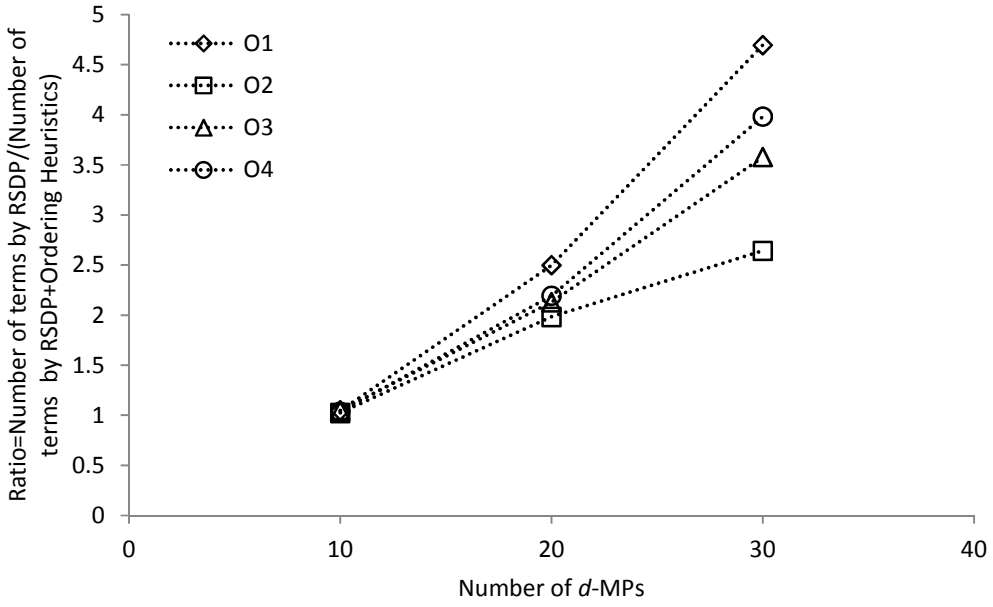

Figure 5.6: Ratio of the mean number of terms for 30 components.

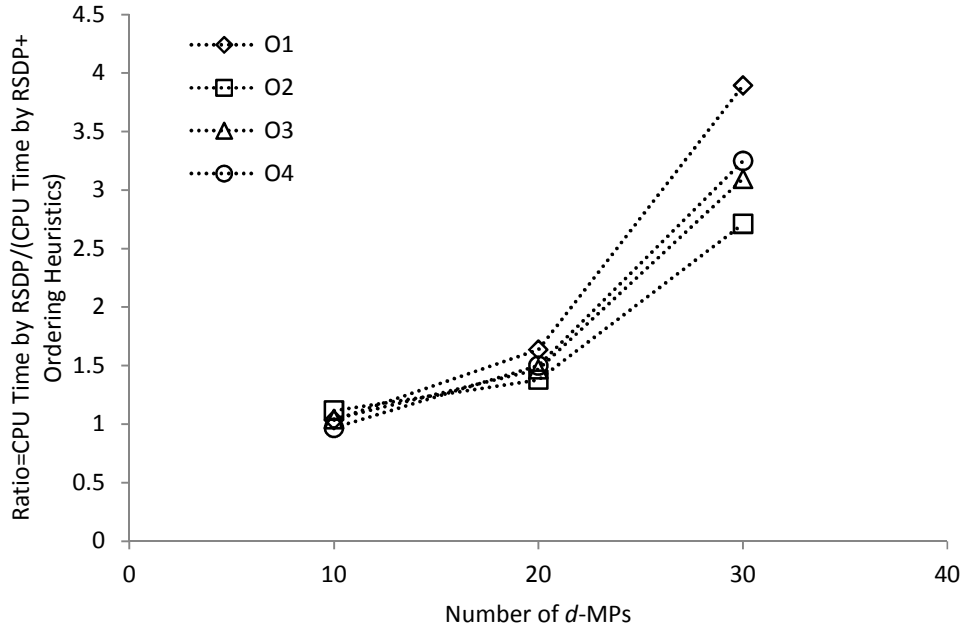

Figure 5.7: Ratio of the mean computation times for 40 components.

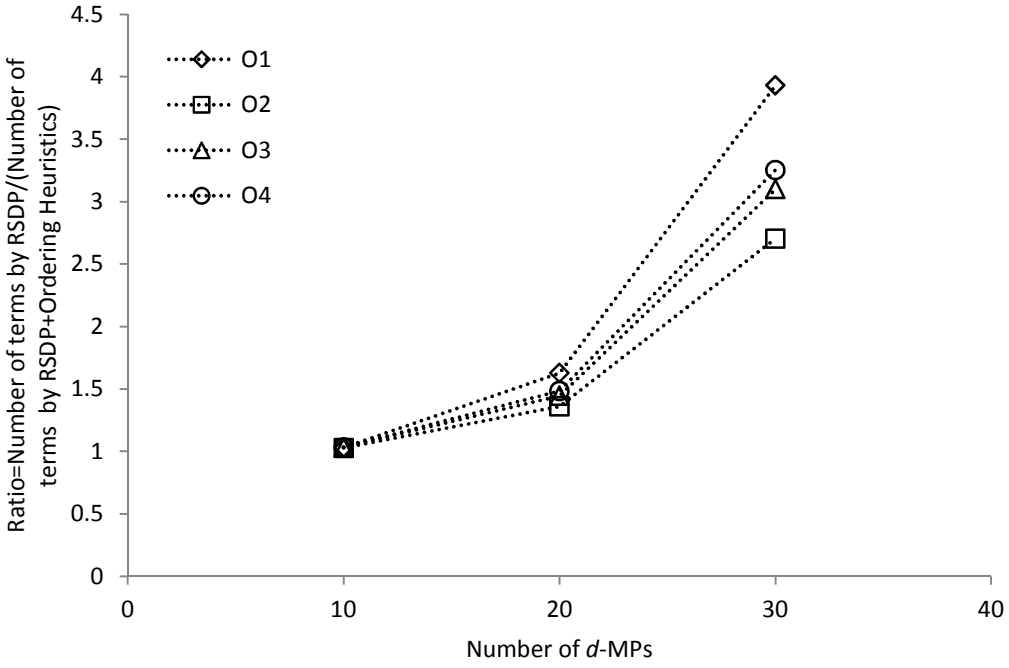

Figure 5.8: Ratio of the mean number of terms for 40 components.

The ratio between the mean CPU times and the ratio between the mean number of terms in disjoint sum increase as the number of *d*-MPs increases, as shown from Figure 5.1 through Figure 5.8. As can be seen from these figures, when the number of *d*-MPs is larger than 10, as the number of *d*-MPs increases, the mean CPU time, and the mean number of terms in the disjoint sums for the RSDP method with O1, O2, O3, and O4 become more advantageous. The efficiency of the RSDP method with O4 performs better than the RSDP method with O2, as the number of *d*-MPs increases. This result indicates that, under length Definition 2, the ordering heuristic O4, incorporating the idea from [67], is better than the ordering heuristic O2 based on length Definition 2 alone. The efficiencies of the RSDP method with O3 and O4 are very close, with O3 performing slightly better under 10 components, and O4 performing slightly better under 20, 30, and 40 components. O1 gives the best performance among the 4 ordering heuristics. The results indicate that the performance of different ordering heuristics is consistent under both efficiency criteria.

5.4 Ordering heuristic methods given minimal cut vectors

The ordering heuristic methods for multistate network reliability evaluations covered earlier in this paper are based on *d*-MPs. However, sometimes only the minimal cut vectors, or *d*-MCs, are available. Given all *d*-MCs, denoted as $\mathbf{c}^1, \mathbf{c}^2, \ldots, \mathbf{c}^L$, the RSDP for *d*-MCs is the same as that for *d*-MPs except that: 1) the "maximum" operator, "⊕", becomes "minimum" operator, "$\otimes$", defined as $\mathbf{c}^i \otimes \mathbf{c}^j \equiv \left(\min\left(c_k^i, c_k^j\right)\right), 1 \le k \le n$; 2) equation (5.1) becomes equation (5.5) as follows:

$$\begin{aligned} &\Pr\left(\phi(\mathbf{x}) \ge d+1\right) = 1 - \Pr\left(\phi(\mathbf{x}) \le d\right) = 1 - \Pr\left(\left\{\mathbf{x} \le \mathbf{c}^1\right\} \cup \left\{\mathbf{x} \le \mathbf{c}^2\right\} \cup \cdots \cup \left\{\mathbf{x} \le \mathbf{c}^L\right\}\right) \\ &\equiv 1 - \Pr\left(\bigcup_{i=1}^{L} \mathbf{x} \le \mathbf{c}^i\right) \equiv 1 - \Pr U\left(\mathbf{c}^1, \mathbf{c}^2 \cdots \mathbf{c}^L\right). \end{aligned} \tag{5.5}$$

In contrast to RSDP given all *d*-MPs [33], for RSDP given *d*-MCs, the vector that is smaller than or equal to others can be removed. Thus, given that O1 gives the best performance among ordering heuristics given all *d*-MPs and it is based on length *Definition 1*, its corresponding ordering heuristic given all *d*-MCs, namely O5, can be developed accordingly. First, we can define the so-called "lowest state vectors" in a way similar to the "highest state vectors" defined in Section 5.2.1. The procedure of O5 is given as follows.

1) *Given all the d-MCs, obtain the lowest state vector, the second lowest state vector, …, and the (n-1)$_{th}$ lowest state vector and the n$_{th}$ lowest state vector;*
2) *For each d-MC, obtain the length with respect to the lowest state, the second lowest state, …, and the n$_{th}$ lowest state using Definition 1;*
3) *Order the d-MCs in ascending order according to their lengths with respect to the lowest state;*
4) *For those d-MCs with the same length with respect to the lowest state, order those d-MPs in ascending order according to their lengths with respect to the second lowest state;*

5) *For those d-MCs with the same length of the second lowest state, order those d-MCs in ascending order according to their lengths with respect to the third lowest state;*

……

This process continues in a similar manner up to the $(n\text{-}1)_{th}$ *lowest state. A tie is broken by random choice.*

The RSDP method and the ordering heuristic O5 proposed in this section are useful for multistate network reliability evaluation if only *d*-MCs are available. The properties of these methods are similar to the RSDP method given all *d*-MPs developed in [39] and the ordering heuristic O1 for *d*-MPs presented earlier in this paper, and thus illustrations and efficiency investigations of these two methods will not be presented in this paper.

5.6 Summary

In this chapter, we develop ordering heuristics to improve the efficiency of reliability evaluation for multistate two-terminal networks given all *d*-MPs. The ordering heuristics are based on the observation that the importance of each *d*-MP is different, and different orderings affect the efficiency of reliability evaluation. We introduce the length definitions for *d*-MPs in a multistate two-terminal network, and develop four ordering heuristics, called O1, O2, O3 and O4, to improve the efficiency of the RSDP method for evaluating the network reliability. The results show that the proposed ordering heuristics can significantly improve the reliability evaluation efficiency, and O1 performs the best among the four methods. In addition, a RSDP method and an ordering heuristic are developed for reliability evaluation of multistate two-terminal networks given all *d*-MCs.

Chapter 6

An Improved Algorithm for Reliability Evaluation of Multistate Networks Using State Space Decomposition Method

Recall the overall framework of the topics in this thesis, as shown in Figure 1.3. In this chapter, we focus on the third and fourth topics. In Chapter 5, given all the *d*-MPs found in Chapter 4, we evaluate the reliability of multistate networks based on the RSDP method. We proposed ordering heuristics for RSDP to improve the reliability evaluation efficiency. Another popular method for exact reliability evaluation of multistate networks is the SSD method. In this chapter, we focus on SSD methods. During each recursive call to decomposition, the existing methods select qualified *d*-MPs by comparing all the *d*-MPs with the upper limiting point [38]. However, we find that the set of *d*-MPs can also be decomposed recursively, and only those qualified *d*-MPs from previous set of unspecified states need to be compared with the upper limiting point. Based on the above observation, we propose an algorithm to improve the efficiency of SSD methods for evaluation of multistate network reliability. An improved heuristic rule is also proposed for choosing a proper *d*-MP to decompose each set of unspecified states.

The limitation of existing algorithms using SSD is introduced in Section 6.1. In section 6.2, a property is proposed to decompose the *d*-MPs, together with a new heuristic rule to select a proper *d*-MP. An improved algorithm is presented, followed by an illustrative example. Section 6.3 investigates the efficiency of proposed algorithm, the existing SSD method, and the RSDP with ordering heuristic O1 [47], using hypothetical networks. Section 6.4 investigates the

efficiency of the proposed algorithm, the existing SSD method, and the RSDP with ordering heuristic O1 [47], using networks with known structures. The existing SSD method, the RSDP with ordering heuristic O1, and direct approaches are also added in the efficiency comparisons. Section 6.5 extends the algorithm to the situations when *d*-MCs instead of *d*-MPs are given for evaluations of the exact reliability. Section 6.6 summarizes the findings. The results of this chapter have been documented in [49].

6.1 Introduction

For the evaluation of the exact reliability of multistate networks, Aven [38] proposed an algorithm based on State Space Decomposition (SSD) method. The state space of a multistate network, S is a Cartesian product of the states of each component, S_i, $i = 1,2,..,n$. Aven's algorithm [38] starts by using one *d*-MP to decompose the state space, S into a set of acceptable states, a set of unacceptable states, and sets of disjoint unspecified states. Then each set of unspecified states is recursively decomposed using a *d*-MP until no set of unspecified states is left. Finally, the reliability is the sum of probabilities of all the disjoint sets of acceptable states. Associated with each set of unspecified states, C, there are two limiting state points $\boldsymbol{b}^0$ and $\boldsymbol{b}$ such that:

$$C = \{\boldsymbol{x} \in S;\ \boldsymbol{b} \leq \boldsymbol{x} \leq \boldsymbol{b}^0\}.$$

Initially, $\boldsymbol{b}^0 = \boldsymbol{X^{Max}}$, where $\boldsymbol{X^{Max}}$ is the maximum state vector, and $\boldsymbol{b} = \boldsymbol{0}$.

Associated with each set of acceptable states, A, there is a critical value $\boldsymbol{v}^0 \in C$ (C is the set of unspecified states which is decomposed) such that:

$$A = \{\boldsymbol{x} \in C;\ \boldsymbol{v}^0 \leq \boldsymbol{x}\}.$$

To each set of unacceptable states, U, there is an associated critical value $\boldsymbol{v}$ such that:

$$U = \{\boldsymbol{x} \in C;\ x_i < v_i,\ \text{for at least one } i\}.$$

As can be seen from Aven's algorithm [38], during each recursive call of decomposition, a filtering process is carried out first to select qualified *d*-MPs that are smaller than or equal to $\boldsymbol{b}^0$. Currently, this is done by comparing all the *d*-MPs with $\boldsymbol{b}^0$. This may not be a computational issue if the total number of *d*-MPs is relatively small. However, for networks with large number of components, the number of *d*-MPs can be very large. Therefore, the filtering process may consume quite a lot of computational effort.

Virtually, if a *d*-MP has been filtered out for a set of unspecified states, C, it is not qualified for the later sets of unspecified states under that C either. Thus, when each set of unspecified states is decomposed, the set of *d*-MPs can also be decomposed into a set of qualified *d*-MPs, and a set of unqualified *d*-MPs. As each set of unspecified states is decomposed recursively, the set of qualified *d*-MPs can also be decomposed recursively. During each decomposition process, a proper *d*-MP is chosen only from the corresponding qualified set of *d*-MPs. By doing so, the number of *d*-MPs that need to be scanned decreases as the decomposition deepens.

Another limitation of Aven's algorithm is the heuristic rule for choosing a proper *d*-MP to decompose each set of unspecified states. During each recursive call of the decomposition, any *d*-MP, $\boldsymbol{z}^l$ that is smaller than or equal to $\boldsymbol{b}^0$ can be used [38]. To make the number of sets of unspecified states as small as possible, Aven [38] suggests that a large $\boldsymbol{v}$ and a small $\boldsymbol{v}^0$ should be sought for. Currently, the heuristic is to choose a *d*-MP, $\boldsymbol{z}^l$, such that $\boldsymbol{z}^l$ maximizes the following equation [38]:

$$H\left(\mathbf{z}^l\right)=\sum_{i=1}^{n}\left(b_i^0-\max\left\{z_i^l,b_i\right\}\right),\ \mathbf{z}^l\le\mathbf{b}^0. \tag{6.1}$$

The idea behind this heuristic rule is given as follows. Given that $\boldsymbol{b}^0$ is the upper limiting point for the current set of unspecified states, C, the *d*-MP is chosen so that the difference between $\boldsymbol{b}^0$ and the *d*-MP is as large as possible. Then it is more likely that a smaller $\boldsymbol{v}^0$ is obtained, resulting in a smaller number of sets of unspecified states. However, during each decomposition process, given $\boldsymbol{v}^0$ and $\boldsymbol{v}$, the number of sets of unspecified states is determined

by the number of i , satisfying $v_i < v_i^0$, $i = 1,2,\dots,n$. [38] Thus, a heuristic rule with $\boldsymbol{v}$ directly involved may provide better performance.

6.2 The development of the algorithm

In this section, a property named *Property 1*, is proposed to decompose the set of *d*-MPs when decomposing each unspecified set. Then an improved heuristic is proposed to select a proper *d*-MP for decomposing each set of unspecified states. Based on *Property 1* and the improved heuristic, we present the improved algorithm. An example is given for illustration.

6.2.1 Decomposition of *d*-MPs

Virtually, the SSD procedure is a type of branch-and-bound procedure, which produces a tree structure, with each node representing a set of states [5]. There are three types of nodes in the tree, one corresponding to a set of acceptable states, one corresponding to a set of unacceptable states, and one corresponding to a set of unspecified states. The node corresponding to a set of unspecified states is further branched into several additional nodes. All the leaf nodes are either corresponding to set of acceptable states, or a set of unacceptable states. The set of states corresponding to a node must be included in its parental node. Thus, if a *d*-MP is not qualified for a parental node, it is not qualified for all the sibling nodes either. Thus, we have the following property.

Property 1 *during the decomposition process, if a d-MP is not qualified for a set of unspecified states,* C*, it is not qualified for all the sets of unspecified states decomposed from that* C *either.*

Proof: Assume there exists an *d*-MP, $\boldsymbol{z}^l$ that is not qualified for a set of unspecified states, C, but qualified for a set of unspecified states, $\hat{C}$, decomposed from C. Thus, we have $\boldsymbol{z}^l \nleq \boldsymbol{b}^0$,

where $\boldsymbol{b}^0$ is the upper limiting point of set C, and $\boldsymbol{z}^l \leq \widehat{\boldsymbol{b}^0}$, where $\widehat{\boldsymbol{b}^0}$ is the upper limiting point of set $\hat{C}$. However, given $\hat{C}$ is decomposed from C, we have $\widehat{\boldsymbol{b}^0} \leq \boldsymbol{b}^0$. Contradiction.

Given *Property 1*, when a set corresponding to a parental node is decomposed, its corresponding qualified *d*-MPs are classified into a set of qualified *d*-MPs, and a set of unqualified *d*-MPs, by comparing with the current upper limiting point. Only those qualified *d*-MPs are considered when decomposing its sibling nodes. When one of these sibling nodes is decomposed (now it becomes parental node for its sibling nodes), its corresponding qualified *d*-MPs are again classified into a set of qualified *d*-MPs, and a set of unqualified *d*-MPs. Only those qualified *d*-MPs are considered for decomposing its sibling nodes. Thus, instead of comparing all the *d*-MPs with the current upper limiting point to find the qualified *d*-MPs for each decomposition procedure, only those qualified *d*-MPs from the parental node is considered.

Let τ denote the total number of unspecified sets generated, *L* denote the total number of *d*-MPs, n denote the number of components, and θ denote the average number of qualified *d*-MPs (*d*-MPs that are smaller than or equal to the current upper limiting point). During each decomposition procedure of Aven's algorithm, it requires $O(L)$ time to select the qualified *d*-MPs, $O(\theta)$ time for the heuristic to select a proper *d*-MP, and $O(n)$ time in the worst case to perform pivot decomposition. Thus, the time complexity of Aven's algorithm is $O(\tau \cdot [L + \theta + n])$. By incorporating the decomposition of *d*-MPs procedure, the decomposition of *d*-MPs implements recursively as the decomposition of state space is implemented recursively. The number of qualified *d*-MPs for the first node, i.e. the root node is m. The number of qualified *d*-MPs for all the sibling nodes (corresponding to a set of unspecified states) of the root node is less than m. The number of qualified *d*-MPs for all the nodes (corresponding to a set of unspecified states) becomes smaller as the decomposition deepens. The number of qualified *d*-MPs for the last node (corresponding to a set of unspecified states) is 1. Let ρ denote the average number of *d*-MPs from the parental node of each unspecified set. By incorporating the decomposition of *d*-

MPs, the time complexity of the Aven's algorithm becomes $O(\tau \cdot [\rho + \theta + n])$. The advantage can be seen given $\rho \ll L$.

6.2.2 An improved heuristic

Because the evaluation of multistate network reliability is an NP-hard problem, the total number of unspecified sets, π grows exponentially with the size of the network. In order to make the number of unspecified sets as small as possible, Aven [38] suggests finding a large $\boldsymbol{v}$ and a small $\boldsymbol{v}^0$ during each decomposition procedure as follows.

After initialization, first choose a *d*-MP, $\boldsymbol{z}^l$, such that $\boldsymbol{z}^l$ maximizes the following equation [38]:

$$H\left(\mathbf{z}^l\right) = \sum_{i=1}^{n}\left(b_i^0 - \max\left\{z_i^l, b_i\right\}\right),\ \mathbf{z}^l \le \mathbf{b}^0,$$

and generate a system state vector, $\hat{\boldsymbol{z}}$, using the following equation [38]:

$$\hat{z}_\iota = \min\{z_i^l, \boldsymbol{z}^l \le \boldsymbol{b}^0\}, i = 1,2,\dots,n.$$

Second, set $v_i^0 = \max\{z_i^l, b_i\}, i = 1,2,\dots,n$. $\boldsymbol{v}^0$ is selected. Set $v_i = \max\{\hat{z}_i, b_i\}, i = 1,2,\dots,n$. $\boldsymbol{v}$ is selected.

After $\boldsymbol{v}$ and $\boldsymbol{v}^0$ are selected, the set of acceptable states is determined as

$$A = \{\boldsymbol{x} \in C;\ \boldsymbol{v}^0 \le \boldsymbol{x}\},$$

and the set of unacceptable states is determined as

$$B = \{\boldsymbol{x} \in C;\ x_i < v_i, \text{for at least one } i\}.$$

In order to make the sets of unspecified states disjoint from each other, the decomposition is implemented by pivoting on each component. Because pivoting on component i, which satisfies $v_i = v_i^0$, is unnecessary, the number of unspecified sets is determined by the number of i, satisfying $v_i < v_i^0, i = 1,2,\dots,n$.

As can be seen, the number of unspecified sets is determined only by $\boldsymbol{v}$ and $\boldsymbol{v}^0$. However, the current heuristic for choosing $\boldsymbol{v}^0$ involves $\boldsymbol{b}^0$. A heuristic that directly involve $\boldsymbol{v}$ may provide better performance. Thus, we propose the following steps.

First, select $\boldsymbol{v}$ using the same equation in [38] as follows:

$$\hat{z_i} = \min\{z_i^l, \boldsymbol{z}^l \leq \boldsymbol{b}^0\}, i = 1,2, \dots, n.$$

Set $v_i = \max\{\hat{z_i}, b_i\}, i = 1,2, \dots, n.$

Second, given $\boldsymbol{v}$ is selected, choose a *d*-MP, $\boldsymbol{z}^l$, such that $\boldsymbol{z}^l$ minimizes the following equation:

$$H\left(\mathbf{z}^l\right) = \sum_{i=1}^{n}\left(\max\left\{z_i^l, b_i\right\} - v_i\right),\ \mathbf{z}^l \leq \mathbf{b}^0. \tag{6.2}$$

During the process, whenever there is a tie, the tie is broken arbitrarily.

Set $v_i^0 = \max\{z_i^l, b_i\}, i = 1,2, \dots, n.$

The idea behind this heuristic is given as follows. Because the number of unspecified sets is determined by the number of i, satisfying $v_i < v_i^0, i = 1,2, \dots, n$. Given $v_i^0 = \max\{z_i^l, b_i\}$, the *d*-MP, $\boldsymbol{z}^l$, is chosen so that the summation of the difference between each pair of v_i^0 and v_i is as small as possible. Therefore, it is more likely that the number of unspecified sets, determined by the number of i satisfying $v_i < v_i^0$, is smaller.

6.2.3 The algorithm

Given *Property 1* and the heuristic, we propose the following algorithm for exact reliability evaluation of multistate network given all *d*-MPs.

1. Set the following parameters: initial reliability value, $R = 0$; index for the current set of unspecified states, $k = 1$; index for the current set of qualified *d*-MPs, $w = L$, where L is the total number of *d*-MPs; initial limiting state points, $\boldsymbol{b}^0 = \boldsymbol{X^{Max}}$, and $\boldsymbol{b} = \boldsymbol{0}$. Store all the *d*-MPs

into a stack, Q. Create two matrices, B^0 and B to store the limiting state points of each set of unspecified states respectively. Create a vector $\boldsymbol{w}$, to store the index of qualified *d*-MPs for each set of unspecified states.

2. Based on *Property 1*, decompose the current set of *d*-MPs into qualified *d*-MPs, $\boldsymbol{z}^l \leq \boldsymbol{b}^0$, and nonqualified *d*-MPs, $\boldsymbol{z}^l \nleq \boldsymbol{b}^0$; put qualified *d*-MPs on the top of the stack Q. Update the index for the current set of qualified *d*-MPs, w, which equals to the number of qualified *d*-MPs.
3. Given all the qualified *d*-MPs for the current set, do the following:
   1) $z_i^* = \min\{z_i^l\}, i = 1,2, \dots, n.$
   2) $v_i = \max\{z_i^*, b_i\}, i = 1,2, \dots, n.$
   3) choose a *d*-MP, $\boldsymbol{z}^l$, such that $\boldsymbol{z}^l$ minimizes the following equation:

$$H\left(\mathbf{z}^l\right) = \sum_{i=1}^{n}\left(\max\left\{z_i^l, b_i\right\} - v_i\right),\ \mathbf{z}^l \leq \mathbf{b}^0.$$

   During the process, whenever there is a tie, the tie is broken arbitrarily.

   4) $v_i^0 = \max\{z_i^l, b_i\}, i = 1,2, \dots, n.$
4. Set $p_i = Pr\left(v_i^0 \leq x_i \leq b_i^0\right), i = 1,2, \dots, n.$

$$R \leftarrow R + \prod_{i=1}^{n} p_i.$$

5. Let a_d, $d = 1,2, \dots, s$ be the i, $i \in \{1,2, \dots, n\}$, satisfying $v_i < v_i^0$. If no such i exists, set $s = 0$. If $s \geq 1$, for $d = 1,2, \dots, s, i = 1,2, \dots, n$, do the following:

$$B^0(d + k - 1, i) = \begin{cases} v_i^0 - 1, & \text{for } i = a_d \\ b_i^0, & \text{otherwise} \end{cases}$$

$$B(d + k - 1, i) = \begin{cases} v_i^0, & \text{for } i < a_d \\ v_i, & \text{otherwise} \end{cases}$$

$$\boldsymbol{w}(d + k - 1) = w.$$

Set $k \leftarrow k - 1 + s$. If $k = 0$, stop and output R. Otherwise, set $\boldsymbol{b}^0 = B^0(k)$, $\boldsymbol{b} = B(k)$, $w = \boldsymbol{w}(k)$. Go to step 2.

6.2.4 An illustrative example

Consider the example shown in Figure 2.1. Table 2.1 lists the state distribution of each component in the network.

There are three 3-MPs given in the following arbitrary order [39].

$$z^1=(3,2,1,0,0,1), z^2=(2,1,1,0,1,2), z^3=(2,2,0,0,1,1).$$

step 1 $R=0, k=1, w=L=3, \boldsymbol{b}^0=\boldsymbol{X^{Max}}=(3,2,1,1,1,2), \boldsymbol{b}=(0,0,0,0,0,0)$.

step 2 All three 3-MPs are smaller or equal to $\boldsymbol{b}^0$, $w=3$.

step 3

1) $z_1^*=\min\{z_1^l, \boldsymbol{z}^l \le \boldsymbol{b}^0\}=\min\{3,2,2\}=2, z_2^*=1, z_3^*=0, z_4^*=0, z_5^*=0, z_6^*=1.$

2) $v_1=\max\{z_1^*, b_1\}=\max\{2,0\}=2, v_2=1, v_3=0, v_4=0, v_5=0, v_6=1.$

3) $H(\boldsymbol{z}^1)=\sum_{i=1}^{6}(\max\{z_i^l, b_i\}-v_i)=1+1+1=3,\ H(\boldsymbol{z}^2)=3, H(\boldsymbol{z}^3)=2.$

$H(\boldsymbol{z}^1)=H(\boldsymbol{z}^2)>\ H(\boldsymbol{z}^3)$. We select $\boldsymbol{z}^3$.

4) $v_1^0=\max\{z_1^3, b_1\}=2,\ v_2^0=2, v_3^0=0, v_4^0=0, v_5^0=1, v_6^0=1.$

step 4 $p_1=Pr(v_1^0 \le x_1 \le b_1^0)=0.85, p_2=0.6, p_3=1, p_4=1, p_5=0.9, p_6=0.95.$

$$R=R+\prod_{i=1}^{6} p_i=0.43605\ . \tag{18}$$

step 5 Given $v_i<v_i^0$ for $i=2,5$. $s=2$. We have $a_1=2, a_2=5$.

1) For $d=1, a_1=2$,

$B^0(1,1)=b_1^0=3\ ;\ B^0(1,2)=v_1^0-1=1;\ B^0(1,3)=1; B^0(1,4)=1; B^0(1,5)=1; B^0(1,6)=2.$

Thus $B^0(1,:)=(3,1,1,1,1,2)$.

$B(1,1)=2; B(1,2)=1;\ B(1,3)=0; B(1,4)=0; B(1,5)=0; B(1,6)=1;$

Thus, $B(1,:)=(2,1,0,0,0,1)$. $\boldsymbol{w}(1)=3$.

2) For $d=2, a_2=5$,

$B^0(2,1)=b_1^0=3; B^0(2,2)=2;\ B^0(2,3)=1; B^0(2,4)=1; B^0(2,5)=v_5^0-1=0;\ B^0(2,6)=2.$

Thus $B^0(2,:)=(3,1,1,1,1,2)$.

$B(2,1)=v_1^0=2; B(2,2)=2;\ B(2,3)=0; B(2,4)=0; B(2,5)=0; B(2,6)=1;$

Thus, $B(2,:)=(3,1,0,0,0,1)$. $\boldsymbol{w}(2)=3$.

step 6 $k = 2 \neq 0$, set $\boldsymbol{b}^0 = B^0(2) = (3,2,0,1,1,2)$, $\boldsymbol{b} = B(2) = (3,2,0,0,0,1)$. $w = \boldsymbol{w}(3) = 3$. Go to step 2.

step 2 only $\boldsymbol{z}^1 \leq \boldsymbol{b}^0$, $w = 1$.

step 3

1) $\boldsymbol{z}^* = \boldsymbol{z}^1 = (3,2,1,0,0,1)$.
2) $\boldsymbol{v} = \max(\boldsymbol{z}^*, \boldsymbol{b}) = (3,2,1,0,0,1)$.
3) $\boldsymbol{v}^0 = \max(\boldsymbol{z}^3, \boldsymbol{b}) = (3,2,1,0,0,1)$.

step 4 $p_1 = Pr(v_1^0 \leq x_1 \leq b_1^0) = 0.6, p_2 = 0.6, p_3 = 0.9, p_4 = 1, p_5 = 0.1, p_6 = 0.95$.

$$R = R + \prod_{i=1}^{6} p_i = 0.43605 + 0.03078 = 0.46683 . \quad (19)$$

Step 5 no such i exists satisfying $v_i < v_i^0$. Set $s = 0$.

Step 6 $k = 1 \neq 0$, set $\boldsymbol{b}^0 = B^0(1) = (3,1,1,1,1,2)$, $\boldsymbol{b} = B(1) = (2,1,0,0,0,1)$.

$w = \boldsymbol{w}(2) = 3$. Go to step 2.

Step2 only $\boldsymbol{z}^2 \leq \boldsymbol{b}^0$, $w = 1$.

step 3

1) $\boldsymbol{z}^* = \boldsymbol{z}^2 = (2,1,1,0,1,2)$.
2) $\boldsymbol{v} = \max(\boldsymbol{z}^*, \boldsymbol{b}) = (2,1,1,0,1,2)$.
3) $\boldsymbol{v}^0 = \max(\boldsymbol{z}^2, \boldsymbol{b}) = (2,1,1,0,1,2)$.

step 4 $p_1 = Pr(v_1^0 \leq x_1 \leq b_1^0) = 0.85, p_2 = 0.3, p_3 = 0.9, p_4 = 1, p_5 = 0.9, p_6 = 0.7$.

$$R = R + \prod_{i=1}^{6} p_i = 0.46683 + 0.144585 = 0.611415. \quad (20)$$

Step 5 no such i exists satisfying $v_i < v_i^0$, set $s = 0$.

Step 6 $k = 0$. Stop and output $R = 0.611415$.

This result agrees with the result in [39], which illustrates the correctness of the proposed algorithm.

6.3 Efficiency investigation on hypothetical networks

In this section, we investigate the efficiency of the proposed algorithm using hypothetical networks. We compare the efficiency of the proposed algorithm with Aven's algorithm [38] and RSDP with ordering heuristic O1 [47]. The computation time is affected by three parameters, namely the number of components (the size of the network), the number of *d*-MPs, and the number of states of each component. Thus, we consider hypothetical networks given different settings of the three parameters. In terms of performance, we are interested in the CPU time required for obtaining the exact reliability. Let T_1, T_2 and T_3 represent the CPU time (in seconds) by Aven's algorithm [38], RSDP + O1 [47], and the proposed algorithm respectively. In addition, let $r_1 = {}^{T_1}/_{T_3}$ denote the ratio between the CPU time by Aven's algorithm [38] and that of proposed algorithm; $r_2 = {}^{T_2}/_{T_3}$ denote the ratio between the CPU time by RSDP algorithm [47] and that of proposed algorithm. r_1 and r_2 present the advantage of proposed algorithm over Aven's algorithm and RSDP + O1 respectively. All algorithms are coded in Matlab 2015 and the experiments are carried on a personal computer with Windows 7, Intel i7 @ 3.6GHz, 16 GB RAM.

First, we consider hypothetical networks with fixed number of components and fixed number of states for each component. We consider 20 components and all components are i.i.d with 3 states. We randomly generate *L* vectors, and ensure that no vector is dominated. As a result, these *L* vectors can be treated as *L d*-MPs of some hypothetical networks. We consider different scenarios with 200, 400, 600, 800, and 1000 *d*-MPs. Results are shown in Table 6.1. RSDP with O1 is more efficient than Aven's algorithm for all scenarios, and the proposed algorithm is more efficient than Aven's algorithm and the RSDP with O1. Figure 6.1 shows the ratios with respect to different number of *d*-MPs.

Table 6.1: Efficiency comparison given different numbers of *d*-MPs

Number of *d*-MPs	200	400	600	800	1000
CPU time by Aven's algorithm (T_1)	91.94	554.72	973.51	2379.50	4395.06
CPU time by RSDP + O1 (T_2)	42.12	460.70	743.51	1564.88	3475.94
CPU time by the proposed algorithm (T_3)	12.63	46.17	54.23	104.99	159.12
Ratio ($r_1 = {}^{T_1}/_{T_3}$)	7.28	12.01	17.95	22.66	27.62
Ratio ($r_2 = {}^{T_2}/_{T_3}$)	3.33	9.98	13.71	14.90	21.85

The advantageous of the proposed algorithm over Aven's algorithm increases as the number of *d*-MPs increases. This can be explained given the time complexities of Aven's algorithm and the proposed algorithm, which are $O(\tau \cdot [L + \theta + n])$ and $O(\tau \cdot [\rho + \theta + n])$ respectively. The main advantage of the proposed algorithm over Aven's algorithm is due to $\rho \ll L$. As the number of *d*-MPs, L increases, the average number of qualified *d*-MPs, ρ increases. But ρ grows much slower than L. Thus, the ratio r_1 increases with number of *d*-MPs. The advantageous of the proposed algorithm over RSDP with O1 increases as the number of *d*-MPs increases. This can be explained as follows. The computational time of RSDP with O1 based on SDP method grows exponentially with the number of *d*-MPs. However, the computational time of the proposed algorithm does not increase as fast as RSDP with O1 with number of *d*-MPs. Thus, the ratio r_2 increases with the number of number of *d*-MPs.

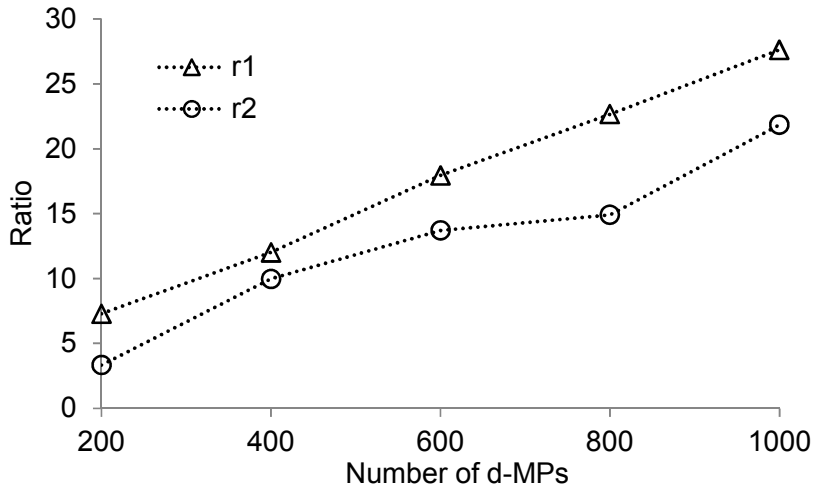

Figure 6.1: Ratio with respect to different numbers of *d*-MPs

Second, we consider hypothetical networks with fixed number of *d*-MPs and fix number of components. We consider 20 i.i.d components and 100 *d*-MPs. We consider different scenarios with 3, 4, 5, 6, and 7 states. Results are shown in Table 6.2, RSDP with O1 is more efficient than Aven's algorithm. The proposed algorithm is more efficient than RSDP with O1 when the number of states is small (3 and 4). However, RSDP with O1 is more efficient than the proposed algorithm when the number of states is greater than or equal to 5.

As can be seen from Figure 6.2, the advantageous of the proposed algorithm over Aven's algorithm stays about the same as the number of states increases. This can be explained given the time complexities of Aven's algorithm and the proposed algorithm, which are $O(\tau \cdot [L + \theta + n])$ and $O(\tau \cdot [\rho + \theta + n])$ respectively. The main advantage of the proposed algorithm over Aven's algorithm is due to $\rho \ll L$. Because the number of *d*-MPs is fixed for different scenarios, the ratio r_1 remain the same. The advantageous of RSDP with O1 over the proposed algorithm increases as the number of states increases for states greater than 5. This can be explained as follows. The computational time of the proposed algorithm based on SSD method grows exponentially with the number of sets of unspecified states. The number of sets of unspecified states increases with the state space of the network. Therefore, as the number of states for

each component increases, the state space of the network becomes larger, resulting in larger number of sets of unspecified states. However, the computational time of RSDP with O1 does not increase as fast as the proposed algorithm when the number of states increases. Thus, the ratio r_2 decreases with number of states.

Table 6.2: Efficiency comparison given different numbers of states

Number of states	3	4	5	6	7
CPU time by Aven's algorithm (T_1)	28.28	133.50	654.00	1320.00	3851.08
CPU time by RSDP + O1 (T_2)	15.11	47.60	145.40	257.53	634.87
CPU time by proposed algorithm (T_3)	7.00	32.86	167.54	320.55	956.89
Ratio ($r_1 = {}^{T_1}/_{T_3}$)	4.04	4.06	3.90	4.12	4.02
Ratio ($r_2 = {}^{T_2}/_{T_3}$)	2.16	1.45	0.8679	0.8034	0.6635

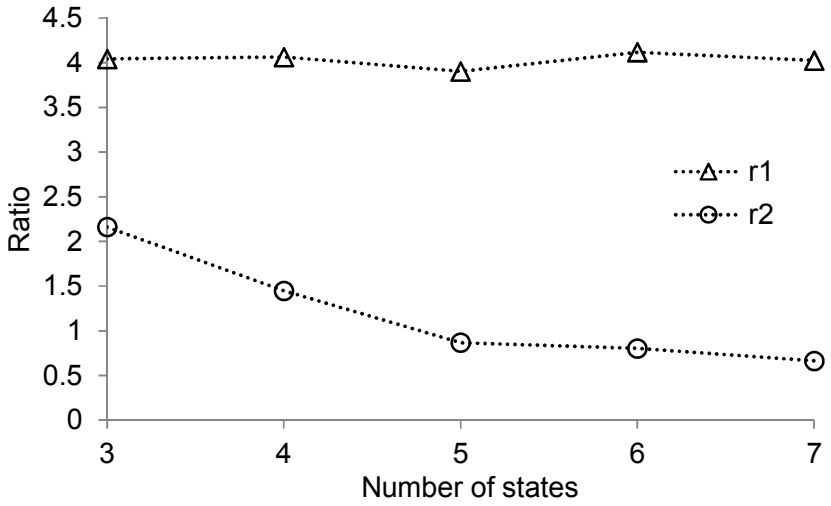

Figure 6.2: Ratio with respect to different numbers of states

Third, we consider hypothetical networks with fix number of *d*-MPs and fix number of states for each component. We considered 100 *d*-MPs and all components are i.i.d with 3 states. We consider different scenarios with 20, 25, 30, 35, and 40 components. Results are shown in

Table 6.3. The proposed algorithm is more efficient than RSDP with O1 when the number of components is less than 20. However, RSDP with O1 is more efficient than the proposed algorithm when the number of components is greater than or equal to 25.

Table 6.3: Efficiency comparison given different numbers of components

Number of components	20	25	30	35	40
CPU time by Aven's algorithm (T_1)	24.34	152.50	538.42	2781.25	10135.85
CPU time by RSDP + O1 (T_2)	10.28	36.58	54.73	238.11	737.63
CPU time by proposed algorithm (T_3)	5.89	42.22	148.67	836.06	3342.21
Ratio ($r_1 = {}^{T_1}/_{T_3}$)	4.13	3.61	3.622	3.33	3.03
Ratio ($r_2 = {}^{T_2}/_{T_3}$)	1.74	0.87	0.3681	0.2848	0.2207

As can be seen from Figure 6.3 , the advantageous of RSDP with O1 over the proposed algorithm increase as the number of states increases when the number of states is greater than 25. Algorithm based on SSD method is more sensitive to the number of components than that of algorithm based on SDP method. This can be explained as follows. The computational time of the proposed algorithm based on SSD method grows exponentially with the number of sets of unspecified states. The number of sets of unspecified states increases with the state space of the network. Therefore, as the number of components increases, the state space of the network becomes larger, resulting in larger number of sets of unspecified states. However, the computational time of RSDP with O1 does not increase as fast as the proposed algorithm when the number of components increases. Thus, the ratio r_2 decreases with number of states. The decreasing trend of ratio r_1 can be explained given the time complexities of Aven's algorithm and the proposed algorithm, which are $O(\tau \cdot [L + \theta + n])$ and $O(\tau \cdot [\rho + \theta + n])$ respectively. The main advantage of the proposed algorithm over Aven's algorithm is due to $\rho \ll L$. Thus, the ratio

r_1 decreases as the number of components, n increases. However, ratio r_1 will always be greater than one.

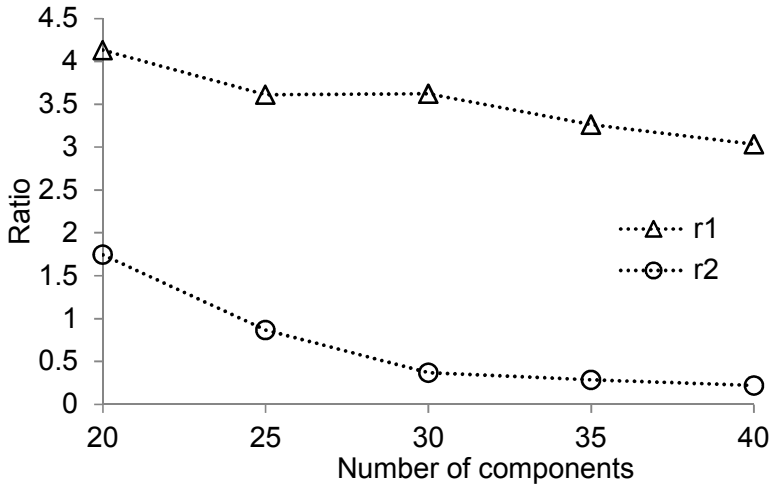

Figure 6.3: Ratio with respect to different numbers of components

6.4 Efficiency investigation on real networks

The efficiency investigation in Section 6.3 is based on hypothetical networks. Thus, it is necessary to evaluate the performance of the proposed algorithm using networks with known structures. In addition, we assume that all the *d*-MPs are not known in advance. We adopt the algorithm reported in [43] to search for all MPs, and the algorithm reported in [45] to search for all *d*-MPs of one *d* level given all MPs. Stop the search algorithm in [45] when certain demand *d* is reached. We incorporate these procedures into the proposed algorithm, Aven's algorithm [38] and RSDP with ordering heuristic O1 [47] respectively. We also compare the efficiency with the method based on direct approaches by Jane and Laih [19], which does not require *d*-MPs as prior information. Jane's algorithm requires a so called *d*-flow, to be derived first to decompose each unspecified state space. In order to derive a *d*-flow, the reported Maximum Flow algorithm is used in Jane's algorithm. In this work, we adopt the build-in Maximum Flow algorithm from Matlab by Edmond [68]. Let T_4 , T_5, T_6 and T_7 represent the CPU time (in seconds) by the

algorithm in [19], Aven's algorithm (incorporating the algorithm in [43] and the algorithm in [45]), RSDP with ordering heuristic O1 (incorporating the algorithm in [43] and the algorithm in [45]), and the proposed algorithm (incorporating the algorithm in [43] and the algorithm in [45]) respectively. Let $r_3 = {}^{T_4}/_{T_7}$, $r_4 = {}^{T_5}/_{T_7}$, and $r_5 = {}^{T_6}/_{T_7}$ denote the ratios. These ratios present the advantage of proposed algorithm over the other reported algorithms.

6.4.1 Example 1

We consider a network with 21 components as shown in Figure 6.4. We assume the 21 components are independent and identically distributed (i.i.d.), with 5 states (capacities) from 0 to 4. The state distribution of each component is $p = (0.05, 0.15, 0.2, 0.25, 0.35)$.

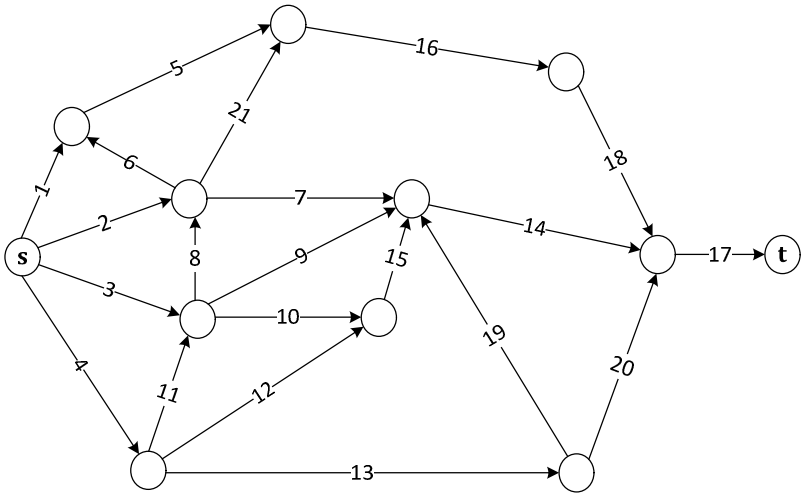

Figure 6.4: A moderate multistate network [42]

We consider the following different demands, which are *d*=1, *d*=2, *d*=3 and *d*=4. The results are shown in Table 6.4. The proposed algorithm is faster than Jans'algorithm, Aven's algorithm and RSDP+O1. Jane's algorithm is slower than Aven's algorithm when the demand is 1 and 2, and faser when the demand is 3 and 4. Jane's algorithm is slower than RSDP+O1 when the demand is 1, 2, and 3, and faser when the demand is 4. Aven's algorithm is faster than RSDP+O1 when the demand is 1 and 4, and slower when the demand is 2 and 3.

Table 6.4: Efficiency comparison for a network in [42]

Demand (*d*)	1	2	3	4
Number of *d*-MPs (L)	16	126	671	2761
Reliability (R)	0.9493	0.7944	0.5815	0.3182
CPU time by Jane's algorithm (T_4)	0.3100	7.91	92.46	703.23
CPU time by Aven's algorithm (T_5)	0.0276	1.82	94.30	2985.32
CPU time by RSDP + O1 (T_6)	0.0384	0.9198	78.11	5174.08
CPU time by proposed algorithm (T_7)	0.0195	0.4279	8.42	107.13
Ratio ($r_3 = {}^{T_4}/_{T_7}$)	15.88	18.48	10.98	6.56
Ratio ($r_4 = {}^{T_5}/_{T_7}$)	1.41	4.24	11.20	27.87
Ratio ($r_5 = {}^{T_6}/_{T_7}$)	1.97	2.15	9.28	48.30

Figure 6.5 presents the ratios in with respect to different demands. Compared with Aven's algorithm and RSDP+O1, the poroposed becomes more advantageous as the demand increases. This is because the number of *d*-MPs increases as the demand increases in this example. This agrees with the resulsts and discussiosn in the hypothetical networks with different *d*-MPs in Section 3. The ratio r_3 decreases as the demand increases when the demand is greater than 2. This can be explained as follows. The *d*-flow in Jane's algorithm is a component state vector for decomposing the set of unspecified states, which has similar function of the *d*-MP in the porposed algorithm. Jane's algorithm uses maximum flow algorithm to derive a *d*-flow for each set of unspecified states. Thus, there is no procedure of comparing and selecting qualified *d*-MPs. Thus, the computational time of Jane's algorithm is not affected by the number of *d*-MPs. As the demand increases, the number of *d*-MPs grows very fast. As can be seen by the time complexity of the proposed algorithm, $O(\pi \cdot [\rho + \mu + n])$, ρ also increases as the number of *d*-MPs grows. Thus, the advantage of the poroposed algorihtm over Jane's algorithm become less as the demand increases when the demand is greater than 2.

Another reason is that the direct approach tends to be more efficient when the demand appraoches the maximum demand [18].

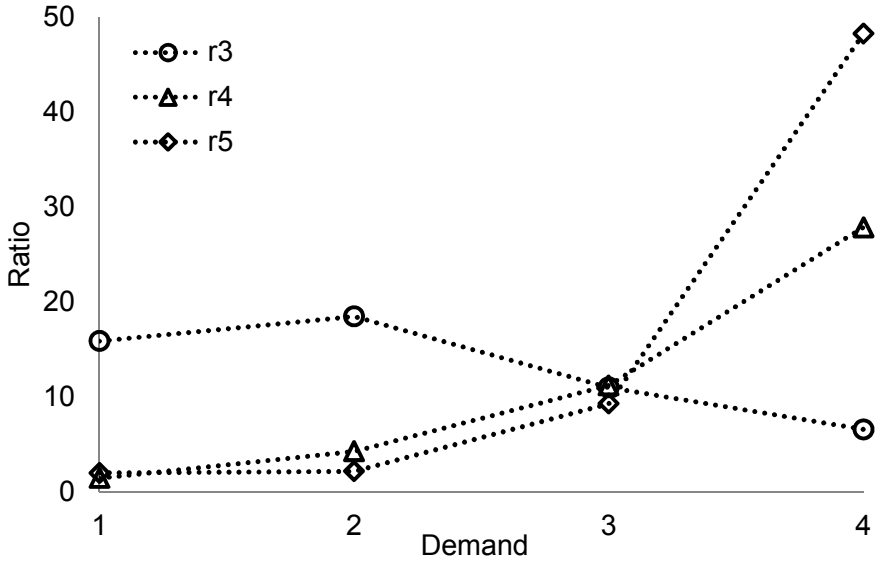

Figure 6.5: Ratio with respect to different demands for a network in [42]

6.4.2 Example 2

We consider a bridge network with 31 components as shown in Figure 6.6. We assume the 31 components are independent and identically distributed (i.i.d.), with 6 states (capacities) from 0 to 4. The state distribution of each component is given as follows.

$$p=\left(0.05,0.1,0.15,0.1,0.2,0.4\right).$$

We consider the following different demands, which are *d*=1*, d*=2, *d*=3, *d*=4 and *d*=5.

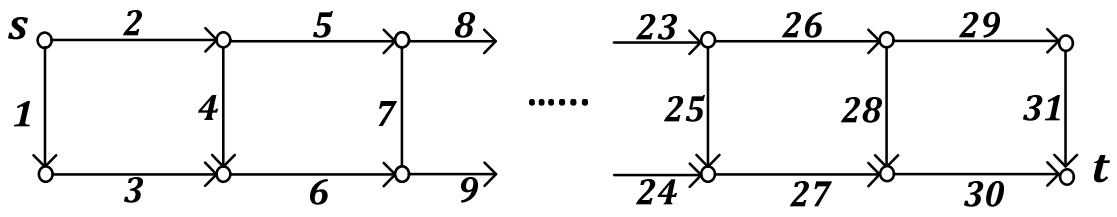

Figure 6.6: A multistate bridge network

Table 6.5: Efficiency comparison for bridge network

Demand (d)	1	2	3	4	5
Number of d-MPs (L)	11	66	286	1001	3003
Reliability (R)	0.8943	0.6356	0.3279	0.1274	0.0362
CPU time by Jane's algorithm (T_4)	13.35	10903.65	——	——	——
CPU time by Aven's algorithm (T_5)	0.0133	0.3974	16.44	499.67	11353.92
CPU time by RSDP + O1 (T_6)	0.0295	0.3157	7.53	272.68	7077.37
CPU time by proposed algorithm (T_7)	0.0084	0.5226	23.45	343.49	3890.97
Ratio ($r_4 = {}^{T_4}/_{T_7}$)	1517.05	20967.87	——	——	——
Ratio ($r_5 = {}^{T_5}/_{T_7}$)	1.5812	0.7605	0.7011	1.4547	2.9180
Ratio ($r_6 = {}^{T_6}/_{T_7}$)	3.5174	0.6042	0.3213	0.7939	1.8189

The results are shown in Table 6.5. We did not record the CPU time for Jane's algorithm when the demand is 3 and above, because we did not get these results within 10 days. RSDP+O1 is faster than the proposed algortihm when the demand is 2, 3, and 4, but slower when the demand is 1 and 5. The propose algorithm is faster than Aven's algorithm when the demand is 1, 4, and 5, but slower when the demand is 2 and 3. The proposed algortihm has more advantage when the CPU time is large. Figure 6.7 presents the ratios of r_4 and r_5 in with respect to different demands.

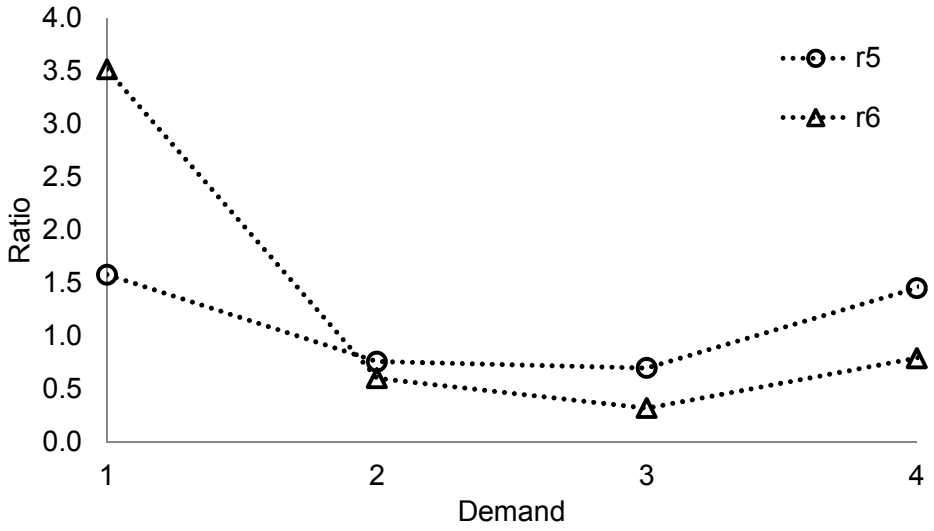

Figure 6.7: Ratio with respect to different demands for a bridge network

As can be seen that Jane's algorithm is much slower than other algorithms. The reason behind can be explained as follows. Given 31 components and 6 states for each component, the state space of this network is very large comparing to Exmaple 1. However, the structure of this network is relatively simple and the number of *d*-MPs is very small comparing to Example 1. The propsoed algorithm, RSDP+O1, and aven's algorithm are indirect approaches which require all *d*-MPs to be known in advance. Becasue the number of *d*-MPs are small, these indirect approacheses become very efficient. On the other hand, as a direct SSD approach, Jane's algorithm cannot make use of this advantage. RSDP+O1 is more efficient than the proposed algorithm when the demand is 2, 3, and 4. This is because the number of *d*-MPs is relatively small given the size of this network and number of states for each component when the demand is 2, 3, and 4. As disscussed in Section 3, when the number of *d*-MPs increases, the computational time of RSDP+O1 grows faster than the proposed algorithm. Thus, when the demand is 5, the number of *d*-MPs increases by more than 2000 compare to the case when demand is 4. In this case, the proposed algorithm is more efficient than RSDP+O1. Aven's algorihtm is more efficient than the proposed algorithm when the demand is 2 and 3. This is also because the number of *d*-MPs is relatively small given the size of this network and number of

states for each component when the demand is 2 and 3. The main advantage of the porposed algoirthm over Aven's algorithm is due to $\rho \ll m$. Since m is relatively small, computational time saving due to $\rho \ll m$ is also very small. On the other hand, the additional steps of the proposed algorithm over Aven's algorithm, particulary the process of updating stack, Q for storing all the sorted *d*-MPs with qualified ones on top consumes more computational time.

6.4.3 Example 3

We consider a network with 30 components as shown in Figure 6.8. We assume the components are independent and identically distributed (i.i.d.), with 3 states (capacities) from 0 to 2. The state distribution of each component is given as $p=(0.05,0.25,0.7)$. We consider the following different demands, which are *d*=1*, d*=2, *d*=3 and *d*=4.

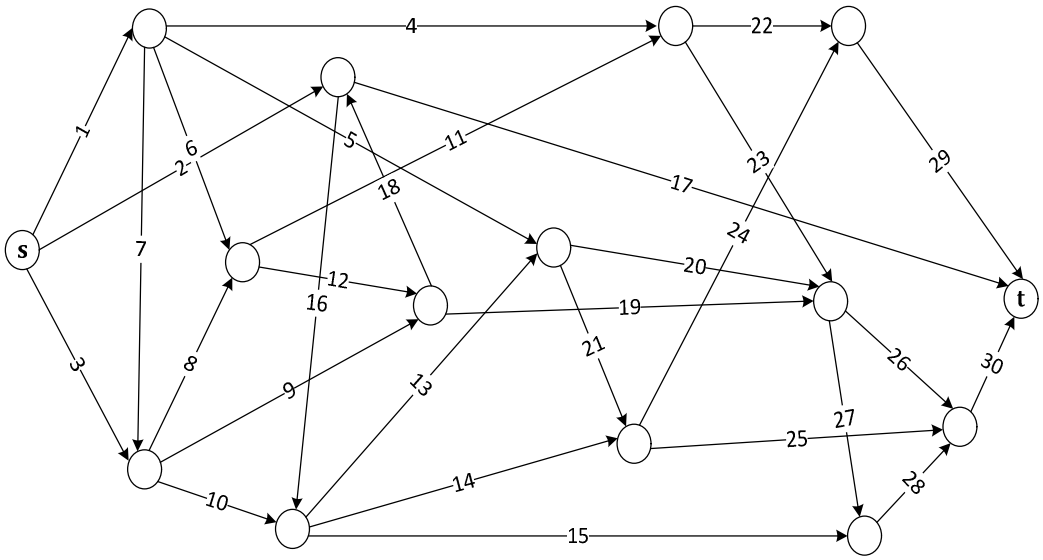

Figure 6.8: A large multistate network [19]

The results are shown in Table 6.6. We did not record the CPU time for RSDP + O1 when the demand is 4, because we did not get the calculation finished within 15 days. The proposed algorithm is faster than Jane's algorithm, Aven's algorithm and RSDP+O1. Jane's algorithm is slower than Aven's algorithm when the demand is 1, and faser when the demand is 2, 3 and 4.

Jane's algorithm is slower than RSDP+O1 when the demand is 1, and faser when the demand is 2, 3 and 4. Aven's algorithm is slower than RSDP+O1 when the demand is 2, and faster when the demand is 1, 3 and 4. Figure 6.9 presents the ratios in logarithm format with respect to different demands. The reasons for the trend of the ratios are similar to Example 1.

Table 6.6: Efficiency comparison for network with 30 components

Demand (*d*)	1	2	3	4
Number of *d*-MPs (L)	88	2573	13804	29102
Reliability (R)	0.9997	0.9956	0.9644	0.8291
CPU time by Jane's algorithm (T_4)	17.56	3983.48	8579.36	7943.91
CPU time by Aven's algorithm (T_5)	4.63	9310.64	82354.07	191553.11
CPU time by RSDP + O1 (T_6)	18.53	5416.68	724650.32	——
CPU time by proposed algorithm (T_7)	1.77	229.7476	1016.92	1172.16
Ratio ($r_4 = {}^{T_4}/_{T_7}$)	9.88	17.33	8.43	6.77
Ratio ($r_5 = {}^{T_5}/_{T_7}$)	2.60	40.52	80.98	163.41
Ratio ($r_6 = {}^{T_6}/_{T_7}$)	10.42	23.57	712.58	——

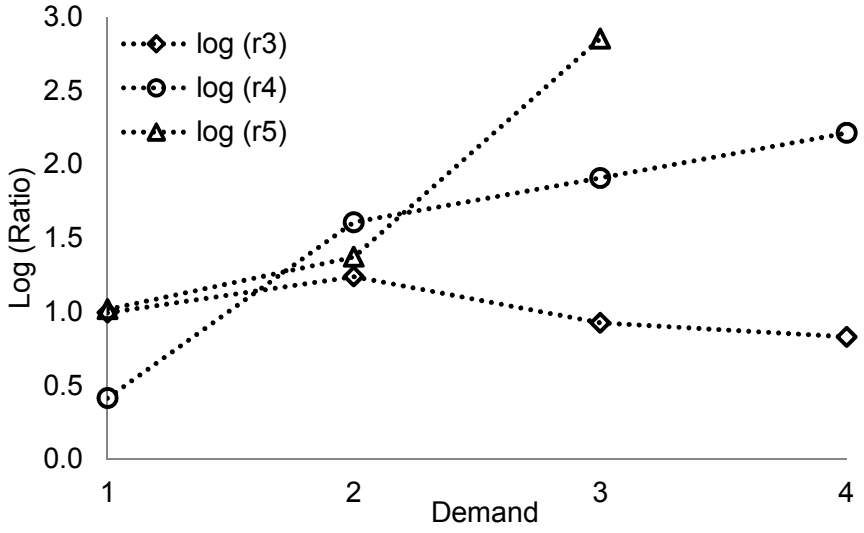

Figure 6.9: Ratio with respect to different demands for a network in [19]

6.5 The algorithm given all minimal cut vectors

The proposed algorithm for multistate network reliability evaluations covered earlier in this paper is based on *d*-MPs. However, sometimes only the minimal cut vectors, or *d*-MCs, are available. Associated with each set of unspecified states, C, there are two limiting state points $\boldsymbol{b}^0$ and $\boldsymbol{b}$ such that [38]:

$$C = \{\boldsymbol{x} \in S;\ \boldsymbol{b}^0 \leq \boldsymbol{x} \leq \boldsymbol{b}\}.$$

Initially, $\boldsymbol{b} = \boldsymbol{M}$, where $\boldsymbol{M}$ is the maximum state vector, and $\boldsymbol{b}^0 = \mathbf{0}$.

For each set of acceptable states, A, there is an associated critical value $\boldsymbol{v}^0 \in C$ (C is the set of unspecified states which is decomposed) such that [38]:

$$A = \{\boldsymbol{x} \in C;\ \boldsymbol{x} \leq \boldsymbol{v}^0\}.$$

To each set of unacceptable states, U, there is an associated critical value $\boldsymbol{v}$ such that [38]:

$$U = \{\boldsymbol{x} \in C;\ x_i > v_i, \text{for at least one } i\}.$$

The corresponding property for decompose *d*-MCs is given as follows.

Property 2 *during the decomposition process, if a d-MC is not qualified for a set of unspecified states, C, it is not qualified for all the sets of unspecified states decomposed from that C either.*

The corresponding heuristic rule for choosing a *d*-MC for decomposition is given as follows. Given v_i , choose a *d*-MC, $\boldsymbol{c}^l$, such that $\boldsymbol{c}^l$ maximizes the following equation:

$$H\left(\mathbf{c}^l\right) = \sum_{i=1}^{n}\left(v_i - \min\left\{c_i^l, b_i\right\}\right),\ \mathbf{c}^l \geq \mathbf{b}^0 . \tag{6.3}$$

During the process, whenever there is a tie, the tie is broken arbitrarily.

With property 2 and the heuristic rule for choosing a *d*-MC, we give the following algorithm for exact reliability evaluation of multistate network given all *d*-MCs.

1. Set the following parameters: initial unreliability value, $\overline{R} = 0$; index for current set of unspecified states, $k = 1$; index for current set of qualified *d*-MPs, $w = L$, where L is the total number of *d*-MCs; initial limiting state points, $\boldsymbol{b}^0 = \mathbf{0}$, and $\boldsymbol{b} = \boldsymbol{X^{Max}}$. Store all the *d*-MCs into a stack, Q. Create two matrices, B^0 and B, to store the limiting state points of each set of unspecified states respectively. Create a vector $\boldsymbol{w}$ to store the index of qualified *d*-MCs for each set of unspecified states.
2. Decompose the current set of *d*-MCs into qualified *d*-MCs, $\boldsymbol{c}^l \geq \boldsymbol{b}^0$, and nonqualified *d*-MCs, $\boldsymbol{c}^l \ngeq \boldsymbol{b}^0$; put qualified *d*-MCs on top of the stack, Q. Update the index for the current set of qualified *d*-MCs, w, which equals to the number of qualified *d*-MCs.
3. Given all the qualified *d*-MCs for current set, do the following:
   1) $c_i^* = \max\{c_i^l\}, i = 1,2, \dots, n.$
   2) $v_i = \min\{c_i^*, b_i\}, i = 1,2, \dots, n.$
   3) choose a *d*-MC, $\boldsymbol{c}^l$, such that $\boldsymbol{c}^l$ maximizes the following equation:

$$H\left(\mathbf{c}^l\right) = \sum_{i=1}^{n}\left(v_i - \min\left\{c_i^l, b_i\right\}\right),\ \mathbf{c}^l \geq \mathbf{b}^0.$$

   During the process, whenever there is a tie, the tie is broken arbitrarily.
   4) $v_i^0 = \min\{c_i^l, b_i\}, i = 1,2, \dots, n.$
4. Set $p_i = Pr\left(b_i^0 \leq x_i \leq v_i^0\right), i = 1,2, \dots, n.$

$$\overline{R} \leftarrow \overline{R} + \prod_{i=1}^{n} p_i\ .$$

5. Let a_d, $d = 1,2, \dots, s$ be the i, $i \in \{1,2, \dots, n\}$, satisfying $v_i > v_i^0$. If no such i exists, set $s = 0$. If $s \geq 1$, for $d = 1,2, \dots, s, i = 1,2, \dots, n$, do the following:

$$B^0(d + k - 1, i) = \begin{cases} v_i^0 + 1, & \text{for } i = a_d \\ b_i^0, & \text{otherwise} \end{cases}$$

$$B(d + k - 1, i) = \begin{cases} v_i^0, & \text{for } i < a_d \\ v_i, & \text{otherwise} \end{cases}$$

$$\boldsymbol{w}(d + k - 1) = w.$$

Set $k \leftarrow k-1+s$. If $k=0$, stop and output $1-\overline{R}$. Otherwise, set $\boldsymbol{b}^0 = B^0(k)$, $\boldsymbol{b} = B(k)$, $w = \boldsymbol{w}(k)$. Go to step 2.

6.6 Summary

In this chapter, we develop an improved algorithm based on SSD method for exact reliability evaluation of multistate networks. The improvement is based on the observation that the set of *d*-MPs/*d*-MCs can also be decomposed recursively and an improved heuristic rule can be used to choose a proper *d*-MP/*d*-MC for each decomposition process. The results show that the proposed algorithm can significantly improve the reliability efficiency.

Up to now, we have developed two methods for reliability evaluation of multistate networks given all *d*-MPs, namely the RSDP with ordering heuristics in Chapter 5 and the improved SSD in this chapter. Based on the efficiency investigations conducted in Section 6.3 and Section 6.4, we suggest the following guideline, which can help with choosing the proper method for certain multistate networks. For multistate networks with simple structures, in which the number of *d*-MPs is relatively small, compare to the size of the networks and/or the number of component states, we suggest using RSDP with O1 proposed in Chapter 5. For example, we suggest using RSDP with O1 when: 1) the number of *d*-MPs is less than 1000 for a network with 30 components and 6 component states. 2) the number of states for each component is greater than or equal to five for a network with 20 components and 100 *d*-MPs. 3) the number of components is greater than or equal to 25 for a network with 3 component states and 100 *d*-MPs. However, for multistate networks with complex structures, in which the number of *d*-MPs is relatively large, compare to the size of the networks and/or the number of component states, we suggest using the improved SSD algorithm proposed in this chapter. For example, we suggest using RSDP with O1 when: 1) the number of *d*-MPs is greater than 1000 for a network with 30 components and 6 component states, or the number of *d*-MPs is greater than or equal

to 88 for a network with 30 components and 3 component states. 2) the number of *d*-MPs is greater than or equal to 200 for a network with 20 components and 3 component states. The above guidelines are summarized from the numerical example investigated in Section 6.3 and Section 6.4. These specific numbers may not apply to other networks.

Chapter 7

Conclusion and Future Work

This chapter summarizes the main contributions of this thesis, and suggests several problems that can be further studied.

7.1 Conclusion

In network reliability analysis, it is often assumed that the components in a network system can take two possible states, completely working or totally failed. The system is said to be functioning if there exists at least one path from a source node to a sink node. However, in many real-world network systems, the component can work at different levels of performance. By allowing both components and systems to have finite number of performance levels (states), from perfect functioning to complete failure, we are able to represent the system conditions with more accuracy and flexibility than the binary system models.

Theoretically, the evaluation of multistate network reliability is an NP-hard problem [5]. Therefore, research aimed at developing efficient algorithms is important for analyzing complex practical networks. Many reported works have proposed efficient algorithms to evaluate the exact reliability of multistate networks. Two types of approaches are popular in the literature, namely direct approaches and indirect approaches. This thesis aims to improve the efficiency of the indirect approaches for exact evaluation of multistate network reliability. In conclusion, the contributions of this thesis are summarized as follows.

(1) An improved backtracking algorithm based on depth-first search is developed for finding all minimal paths (MPs) for binary networks. A minimal path (MP) is a path to

connect the source node and the sink node without cycles. It is a fundamental structure in system reliability analysis. At least one MP has to function for a binary network to operate. MP also serves as building blocks to generate minimal path vectors for multistate networks. In this thesis, we propose a more efficient algorithm to find all MPs in a network. The proposed algorithm becomes more advantageous as the size of the network increases. The proposed algorithm not only accelerates the indirect approaches for the evaluation of multistate network reliability, but also speeds up the path-based methods for the evaluation of binary network reliability. Although finding all MPs is an NP-hard problem, the proposed algorithm significantly enlarges the size of networks that can be analyzed.

(2) Given all the MPs, a recursive breadth-first search algorithm to search for all the *d*-MPs for all possible *d* values is developed. A minimal path vector (*d*-MP) is a combination of MPs. The number of MPs depends on the demand level *d*. One MP in its vector form is indeed a *d*-MP for demand level 1. The multistate network operates if its component state vector is greater than or equal to at least one *d*-MP. In this research, we develop an efficient recursive algorithm to generate all *d*-MPs for all *d* levels simultaneously. The proposed algorithm is the first integrated algorithm to find all *d*-MPs for all possible demand levels, and it is based on implicit enumeration mechanism. In addition, the proposed algorithm can also be used to search for *d*-MPs for one *d* level. With this more efficient algorithm, the evaluation of multistate network state distribution becomes more efficient.

(3) Given all *d*-MPs, ordering heuristics are proposed to improve the efficiency of the RSDP method for evaluating multistate network reliability. Given all the *d*-MPs, the indirect approaches evaluate the reliability of multistate networks by calculating the probability of unions of events that the component state vector is greater than or equal to at least one *d*-MP. The current efficient method is the RSDP method. In this thesis, inspired by the reported ordering heuristics of MPs for the SDP method for binary network reliability evaluation, the ordering heuristics of *d*-MPs for the RSDP method are developed for multistate network reliability

evaluation. By using the developed ordering heuristics to order the *d*-MPs before feeding to the RSDP method, the efficiency of evaluating multistate network is improved significantly.

(4) Given all *d*-MPs, an improved SSD algorithm is developed for evaluating multistate network reliability. Thorough efficiency investigations are conducted to compare the efficiency of the reported direct approaches and indirect methods, including the proposed improved SSD method and the RSDP method with ordering heuristics, for evaluating multistate network reliability. The SSD is another popular method for calculating probability of unions of events that the component state vector is greater than or equal to at least one of the *d*-MPs. In this research, the existing algorithm using SSD method is improved by incorporating the recursive decomposition of the *d*-MPs. In addition, an improved heuristic for selecting a proper *d*-MP is proposed for implementing pivotal decomposition on each set of unspecified states. With this more efficient algorithm based on SSD method and the earlier RSDP method with ordering heuristics, two advanced methods can be used for the union probability evaluation.

Given both RSDP methods with ordering heuristics and the improved SSD method, we compare their efficiencies on hypothetical networks under different settings of parameters, namely the size of the network, the number of possible states of component, and the number of *d*-MPs. With the results of the efficiency comparison, the practitioners can decide which method to choose when analyzing certain practical networks. We also compare their efficiencies on some real networks. In addition, with all the newly developed algorithms in this thesis, we have improved the overall efficiency of the indirect approaches for reliability evaluation of multistate networks. We compare the efficiency of the improved indirect approaches with that of the reported direct approaches on several real networks. From the results in this thesis, we are able to conclude that, currently, the indirect approaches improved by the results of this work is more efficient than the reported direct approaches for the exact evaluation of multistate networks.

7.2 Future work

For the exact evaluation of multistate network reliability, the MMDD method can also be used. Thus, it will be interesting to compare the performance of MMDD with the methods proposed in this thesis for evaluating the multistate network reliability. For the future topics, first, I plan to compare the MMDD with RSDP with ordering heuristics in Chapter 5 and improved SSD algorithm in Chapter 6 based on *d*-MPs. Then I plan to try to generate MMDD from multistate network directly and compare with the proposed indirect approaches in this thesis. Besides the exact evaluation of multistate network reliability, several interesting topics can also be explored for future study.

7.2.1 Approximation of multistate network reliability

The focus of this thesis is on the exact reliability evaluation of multistate networks. The algorithms developed in Chapter 3, Chapter 4, Chapter 5, and Chapter 6, are much more efficient than reported methods for exact reliability evaluation of multistate networks. However, for complex networks with a large number of components and a large number of possible states, it is still computationally expensive and impractical to obtain the exact system reliability and system state distribution. This leads one to consider its approximation when the size of the network is relatively large.

One way of approximating the reliability is the reliability bounding technique. Reliability bounds can provide approximated reliability with less computational effort. We can provide sufficient approximation for practical applications in a much shorter computation time. Then decision maker can make a tradeoff between the computational effort and the acceptable precision. There are some reported works on reliability bounds, which extend those reported studies on exact reliability evaluation [39] [41]. Thus, it is necessary to extend the exact reliability evaluation methods in this thesis and develop better reliability bounding methods.

Another approximation method is the Monte Carlo (MC) simulation. The crude MC technique is straightforward. However, it suffers from heavy computational burden if the network system is highly reliable and/or the required precision is high given that the system is critical. This leads to advanced simulation methods, such as the importance sampling MC method [23] [42]. Thus, it is necessary to extend the exact reliability evaluation methods in this thesis and develop more efficient importance sampling MC methods.

7.2.2 Study of more complex multistate networks

The focus of this thesis is on the two-terminal multistate networks with single commodity. Some real-world networks can be more complicated. One of the extensions is multistate networks with multiple sources and/or sinks. It is necessary to extend two-terminal multistate network reliability analysis to multi-terminal network reliability analysis. For multi-terminal networks, two scenarios remain further exploration. One is that demands for all specified node pairs should be satisfied simultaneously, in which each component may need to carry flows for different node pairs simultaneously. Another scenario is that the demands for all the source-sink pairs are non-simultaneous or non-interacting, in which we wish to compute the probability that every demand can be satisfied individually, without regard to other demands. For example, consider a network system in which each demand represents an “emergency” shipment of a commodity. Such emergencies are rare and unlikely to occur simultaneously.

Other extensions to the network model in this thesis include multistate networks with multiple commodities, multistate networks with time threshold for delivering the flow, multistate networks with budget constraint for delivering the flow and Multistate networks in which the flow is not preserving during transmission. All these extended models require further study. The focus is to develop efficient algorithm to generate the corresponding minimal path/cut vectors (upper/lower boundary points) for each problem. Then the methods developed in Chapter 5 and Chapter 6 can be used to obtain the corresponding reliability.

7.2.3 Design and maintenance optimization of multistate networks

Recall the ultimate goal of this thesis is to provide a powerful tool for optimal design and optimal maintenance strategy for multistate networks. With all the efficient methods and algorithms developed in this thesis, the optimal design and maintenance decision-making, which require repetitive evaluation of multistate network reliability, become more viable. For instance, the optimal design of multistate networks is often solved using importance measure. By analyzing the importance of each component in the network, a heuristic can be developed to solve the optimal component allocation problem and/or the optimal redundancy allocation problem. Thus, based on the exact reliability evaluation methods developed in this thesis, there is a need to explore efficient methods for the evaluation of importance measure of multistate network reliability.

7.2.4 Study of continuous state network reliability

One basic assumption in this thesis is that the state (capacity) of the component and/or the system is a non-negative, integer-valued random variable, following a discrete probability distribution. A more general model is to assume the state (capacity) of a component and/or a system is a continuous random variable with any arbitrary distribution. Then the question becomes how to evaluate the reliability of such continuous state networks. The structure function given in Section 2.1.3 remains the same for continuous state networks. However, the minimal path (cut) vectors do not exist for continuous state networks. All the methods developed for evaluating the multistate networks reliability cannot be applied directly to reliability analysis of continuous state networks. Thus, there is a need to explore the methods for reliability evaluation of continuous state networks.

Bibliography

[1] Andersson, G., Donalek, P., Farmer, R., Hatziargyriou, N., Kamwa, I., Kundur, P., ... & Vittal, V. (2005). Causes of the 2003 major grid blackouts in North America and Europe, and recommended means to improve system dynamic performance. *Power Systems, IEEE Transactions on*, *20*(4), 1922-1928.

[2] Buldyrev, S. V., Parshani, R., Paul, G., Stanley, H. E., & Havlin, S. (2010). Catastrophic cascade of failures in interdependent networks. *Nature*,*464*(7291), 1025-1028.

[3] The 2003 Blackouts of Western North America. URL http://www.toptenz.net/10-largest-20th-century-power-outages.php. Accessed Oct 2015.

[4] The 2012 India blackouts. URL http://www.desismartgrid.com/2012/08/indian-power-grid-blackout-reasons-and-future-requirements/. Accessed Oct 2015.

[5] Ball, M. O., Colbourn, C. J., & Provan, J. S. (1995). Network reliability. *Handbooks in operations research and management science*, *7*, 673-762.

[6] Aven, T. (1987). Availability evaluation of oil/gas production and transportation systems. *Reliability engineering*, 18(1), 35-44.

[7] Lin, Y. K., & Yeh, C. T. (2011). Maximal network reliability with optimal transmission line assignment for stochastic electric power networks via genetic algorithms. *Applied Soft Computing*, 11(2), 2714-2724.

[8] Lin, Y. K. (2009). System reliability evaluation for a multistate supply chain network with failure nodes using minimal paths. *Reliability, IEEE Transactions on*, 58(1), 34-40.

[9] Lin, Y. K., & Chang, P. C. (2012). Evaluate the system reliability for a manufacturing network with reworking actions. *Reliability Engineering & System Safety*, *106*, 127-137.

[10] Lin, Y. K., & Yeh, C. T. (2015). System reliability maximization for a computer network by finding the optimal two-class allocation subject to budget. *Applied Soft Computing*, *36*, 578-588.

[11] Frank, H., & Hakimi, S. (1965). Probabilistic flows through a communication network. *Circuit Theory, IEEE Transactions on*, *12*(3), 413-414.

[12] Doulliez, P., & Jamoulle, E. (1972). Transportation networks with random arc capacities. *RAIRO-Operations Research-Recherche Opérationnelle*, 6(V3), 45-59.

[13] Ford, L. R., & Fulkerson, D. R. (1962). Flow in networks. Princeton University Press.

[14] Evans, J. R. (1976). Maximum flow in probabilistic graphs-the discrete case. *Networks*, 6(2), 161-183.

[15] El-Neweihi, E., Proschan, F., & Sethuraman, J. (1978). Multistate coherent systems. *Journal of Applied Probability*, 675-688.

[16] Barlow, R. E., & Wu, A. S. (1978). Coherent systems with multi-state components. *Mathematics of Operations Research*, 3(4), 275-281.

[17] Griffith, W. S. (1980). Multistate reliability models. *Journal of Applied Probability*, 735-744.

[18] Alexopoulos, C. (1995). A note on state-space decomposition methods for analyzing stochastic flow networks. *Reliability, IEEE Transactions on*, 44(2), 354-357.

[19] Jane, C. C., & Laih, Y. W. (2008). A practical algorithm for computing multi-state two-terminal reliability. *Reliability, IEEE Transactions on*, 57(2), 295-302.

[20] Jane, C. C., & Laih, Y. W. (2010). A dynamic bounding algorithm for approximating multi-state two-terminal reliability. *European Journal of Operational Research*, 205(3), 625-637.

[21] Clancy, D. P., Gross, G., & Wu, F. F. (1983). Probabilistic flows for reliability evaluation of multiarea power system interconnections. *International Journal of Electrical Power & Energy Systems*, 5(2), 101-114.

[22] Fishman, G. S. (1989). Monte Carlo estimation of the maximal flow distribution with discrete stochastic arc capacity levels. *Naval Research Logistics (NRL),* 36(6), 829-849.

[23] Bulteau, S., & El Khadiri, M. (2002). A new importance sampling Monte Carlo method for a flow network reliability problem. *Naval Research Logistics (NRL),* 49(2), 204-228.

[24] Jane, C. C., Lin, J. S., & Yuan, J. (1993). Reliability evaluation of a limited-flow network in terms of minimal cutsets. *Reliability, IEEE Transactions on*, *42*(3), 354-361.

[25] Lin, J. S., Jane, C. C., & Yuan, J. (1995). On reliability evaluation of a capacitated-flow network in terms of minimal pathsets. *Networks*, 25(3), 131-138.

[26] Rai, S., & Aggarwal, K. K. (1980). On complementation of pathsets and cutsets. *Reliability, IEEE Transactions on*, 29(2), 139-140.

[27] Yeh, W. C. (2009). A simple universal generating function method to search for all minimal paths in networks. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(6), 1247-1254.

[28] Al-Ghanim, A. M. (1999). A heuristic technique for generating minimal path and cut sets of a general network. *Computers & industrial engineering*, 36(1), 45-55.

[29] Yeh, W. C. (2007). A simple heuristic algorithm for generating all minimal paths. *Reliability, IEEE Transactions on*, 56(3), 488-494.

[30] Colbourn, C. J. (1987). The combinatorics of network reliability. Oxford University Press, Inc..

[31] Chen, S. G., & Lin, Y. K. (2012). Search for all minimal paths in a general large flow network. *Reliability, IEEE Transactions on*, 61(4), 949-956.

[32] Lin, Y. K. (2001). A simple algorithm for reliability evaluation of a stochastic-flow network with node failure. Computers & Operations Research, 28(13), 1277-1285.

[33] Yeh, W-C. (2002) A simple method to verify all *d*-minimal path candidates of a limited-flow network and its reliability. The international journal of advanced manufacturing technology. 20(1), 77-81.

[34] Yeh, W. C. (2005). A novel method for the network reliability in terms of capacitated-minimum-paths without knowing minimum-paths in advance. Journal of the operational Research Society, 56(10), 1235-1240.

[35] Ramirez-Marquez, J. E., Coit, D. W., & Tortorella, M. (2006). A generalized multistate-based path vector approach to multistate two-terminal reliability. *IIE Transactions*, *38*(6), 477-488.

[36] Hudson, J. C., & Kapur, K. C. (1983a). Reliability analysis for multistate systems with multistate components. *IIE Transactions*, 15(2), 127-135.

[37] Hudson, J. C., & Kapur, K. C. (1983b). Modules in coherent multistate systems. *Reliability, IEEE Transactions on,* 32(2), 183-185.

[38] Aven, T. (1985). Reliability evaluation of multistate systems with multistate components. *Reliability, IEEE Transactions on*, *34*(5), 473-479.

[39] Zuo, M. J., Tian, Z., & Huang, H. Z. (2007). An efficient method for reliability evaluation of multistate networks given all minimal path vectors. *IIE transactions*, *39*(8), 811-817.

[40] Satitsatian, S., & Kapur, K. C. (2006). An algorithm for lower reliability bounds of multistate two-terminal networks. *Reliability, IEEE Transactions on*, *55*(2), 199-206.

[41] Hudson, J. C., & Kapur, K. C. (1985). Reliability bounds for multistate systems with multistate components. Operations Research, 33(1), 153-160.

[42] Ramirez-Marquez, J. E., & Coit, D. W. (2005). A Monte-Carlo simulation approach for approximating multi-state two-terminal reliability. *Reliability Engineering & System Safety*, *87*(2), 253-264.

[43] Bai, G., Tian, Z., & Zuo, M. J. (2016). An improved algorithm for finding all minimal paths in a network. Reliability Engineering & System Safety, 150, 1-10.

[44] Bai, G., Tian, Z & Zuo, M. J. An Improved Algorithm for Searching for Minimal Paths in Two-Terminal Networks. *9th International Conference on Mathematical Methods in Reliability (MMR)*, Tokyo, Japan, June 1-4, 2015.

[45] Bai, G., Zuo, M. J., & Tian, Z. (2015). Search for all *d*-MPs for all *d* levels in multistate two-terminal networks. *Reliability Engineering & System Safety*. 150, 1-10.

[46] Bai, G., Zuo, M. J., & Tian, Z. Search for all *d*-MPs for all *d* levels in multistate networks. *Reliability and Maintainability Symposium (RAMS)*, Palm Harbor, Florida, USA January 26-29, 2015.

[47] Bai, G.; Zuo, M.J.; Tian, Z. (2015). Ordering Heuristics for Reliability Evaluation of Multistate Networks. *Reliability, IEEE Transactions on.* 64(3), 1015-1023.

[48] Bai, G., Zuo, M. J., & Tian, Z. A heuristic for ordering *d*-MPs for evaluation of multistate network reliability. *6th Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modelling (APARM)*, Sapporo, Japan, 21-23 August 2014.

[49] Bai, G.; Tian, Z.; Zuo, M.J. (2016). Reliability Evaluation of Multistate Networks: An improved algorithm using State Space Decomposition and Experimental Comparison. IIE Transactions. Submitted.

[50] Kuo, W., & Zuo, M. J. (2003). *Optimal reliability modeling: principles and applications*. John Wiley & Sons.

[51] Ball, M. O. (1986). Computational complexity of network reliability analysis: An overview. *Reliability, IEEE Transactions on*, *35*(3), 230-239.

[52] Euler diagram for P, NP, NP-complete, and NP-hard set of problems. URL https://en.wikipedia.org/wiki/NP_(complexity). Accessed Dec 2015.

[53] Tarjan, R. E. (1983). *Data structures and network algorithms* (Vol. 14). Philadelphia, PA: Society for industrial and Applied Mathematics.

[54] Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

[55] Knuth, D. E. (1968). *The Art of Computer Programming, Vols. 1, 2, 3*. Addison-Wesley.

[56] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269-271.

[57] Luo, T., & Trivedi, K. S. (1998). An improved algorithm for coherent-system reliability. *Reliability, IEEE Transactions on*, 47(1), 73-78.

[58] Chen, S. G. (2013). Efficiency improvement in explicit enumeration for integer programming problems. In: *2013 IEEE International Conference on Industrial Engineering and Engineering Management*, Bangkok, Thailand, 10 - 13 Dec., pp. 1-3.

[59] Forghani-elahabad, M., & Mahdavi-Amiri, N. (2014). A New Efficient Approach to Search for All Multi-State Minimal Cuts. *Reliability, IEEE Transactions on*, *63*(1), 154-166.

[60] Traldi, L. (2006). Non-minimal sums of disjoint products. *Reliability Engineering & System Safety*, *91*(5), 533-538.

[61] Yeh, W. C. (2007). An improved sum-of-disjoint-products technique for the symbolic network reliability analysis with known minimal paths. *Reliability Engineering & System Safety*, *92*(2), 260-268.

[62] Abraham, J. A. (1979). An improved algorithm for network reliability.*Reliability, IEEE Transactions on*, *28*(1), 58-61.

[63] Locks, M. O. (1987). A minimizing algorithm for sum of disjoint products.*Reliability, IEEE Transactions on*, *36*(4), 445-453.

[64] Wilson, J. M. (1990). An improved minimizing algorithm for sum of disjoint products [reliability theory]. *Reliability, IEEE Transactions on*, *39*(1), 42-45.

[65] Soh, S., & Rai, S. (1993). Experimental results on preprocessing of path/cut terms in sim of disjoint products technique. *Reliability, IEEE Transactions on*, *42*(1), 24-33.

[66] Rai, S., Veeraraghavan, M., & Trivedi, K. S. (1995). A survey of efficient reliability computation using disjoint products approach. *Networks*, *25*(3), 147-163.

[67] Balan, A. O., & Traldi, L. (2003). Preprocessing minpaths for sum of disjoint products. *Reliability, IEEE Transactions on*, *52*(3), 289-295.

[68] Edmonds, J. and Karp, R.M. (1972). Theoretical improvements in the algorithmic efficiency for network flow problems. *Journal of the ACM 19*, 248-264.

[69] Shen, Y. (1995). A new simple algorithm for enumerating all minimal paths and cuts of a graph. Microelectronics Reliability**,** 35(6), 973-976.

[70] Kobayashi, K., & Yamamoto, H. (1999).A new algorithm in enumerating all minimal paths in a sparse network. Reliability Engineering & System Safety, 65(1), 11-15.

[71] Shrestha, A., Xing, L., & Coit, D. W. (2010). An efficient multistate multivalued decision diagram-based approach for multistate system sensitivity analysis. *Reliability, IEEE Transactions on*, *59*(3), 581-592.

[72] Yeh, W. C. (2015). An Improved Sum-of-Disjoint-Products Technique for Symbolic Multi-State Flow Network Reliability. *Reliability, IEEE Transactions on*, *64*(4), 1185-1193.

[73] Yeh, W. C., Bae, C., & Huang, C. L. (2015). A new cut-based algorithm for the multi-state flow network reliability problem. *Reliability Engineering & System Safety*, *136*, 1-7.

[74] Xing, L., & Dai, Y. (2009). A new decision-diagram-based method for efficient analysis on multistate systems. *Dependable and Secure Computing, IEEE Transactions on*, *6*(3), 161-174.

[75] Shrestha, A., Xing, L., & Dai, Y. (2010). Decision diagram based methods and complexity analysis for multi-state systems. *Reliability, IEEE Transactions on*, *59*(1), 145-161.