UNIVERSITY OF ALBERTA

Multi-State System Reliability Analysis and Optimization

by

Zhigang Tian 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Doctor of Philosophy

in

Department of Mechanical Engineering

Edmonton, Alberta

Fall 2007

Your file Votre référence
ISBN: 978-0-494-33080-7
Our file Notre référence
ISBN: 978-0-494-33080-7

Abstract

Reliability measures the ability of a system performing its intended functions. It is one of the most critical performance measures of today's complex systems, such as transportation systems, power systems, communication systems and aircraft systems, and has been emphasized more and more by academia, industry and government. Reliability of a system needs to be evaluated accurately, and it can be improved through design optimization.

In traditional binary reliability framework, both systems and components can only take two possible states: completely working and totally failed. However, engineering systems typically have multiple partial failure states in addition to the above-mentioned completely working and totally failed states. Reliability analysis considering multiple possible states is known as multi-state reliability analysis. Multi-state reliability analysis recognizes the multiple possible states of engineering systems, and enables more accurate system reliability analysis.

Efficient methods were not available for some types of multi-state systems, such as multi-state k-out-of-n systems and multi-state network systems. Without such efficient methods, it is time-consuming, sometimes impossible, to evaluate the reliability of complex systems, and approximation approaches have to be used. Efficient reliability evaluation methods are also required in reliability based system design, which typically involves many iterations of system

reliability evaluation.

This dissertation documents research contributions to multi-state system reliability theory, including reliability modeling, evaluation and optimal design of multi-state system. The contributions are summarized as follows:

(1) Efficient methods have been developed for reliability evaluation of multi-state systems with k-out-of-n structures.

(2) An efficient method has been developed for reliability evaluation of multi-state two-terminal networks given all minimal path vectors.

(3) The recursive computation principle is the common principle in all system reliability evaluation algorithms proposed in this work.

(4) An effective method has been developed for dealing with multiple objectives typically involved in the optimal design of multi-state series-parallel systems.

(5) A joint reliability and redundancy optimization method has been developed for multi-state series-parallel systems.

With efficient reliability evaluation methods and effective reliability based design approaches for multi-state engineering systems, the research results out of this work provides useful tools for achieving highly reliable and cost effective engineering systems.

ACKNOWLEDGEMENTS

First, I would like to thank my Ph.D. supervisor, Dr. Ming J. Zuo. None of my research achievements during the past four years would be possible without his direction, inspiration and support. I am also grateful for the precious opportunities and strong support that he has given me in my career development. From him, I have observed and learned how to become an excellent scholar and a great person. And I will continue benefiting from what I have learned from him throughout my whole career.

I would like to thank my Ph.D. examining committee members, Dr. Peter C. Flynn, Dr. John Doucette, Dr. Simaan M. AbouRizk and Dr. Kailash C. Kapur, for their help in the development of this thesis. Thanks to Dr. Kapur in particular for taking the efforts to travel from University of Washington in Seattle, USA to attend my final oral examination.

I would like to thank Dr. Doucette again for his kind help in my career development. I would also like to thank several other professors who have helped me in my research and career: Dr. Richard C.M. Yam, Dr. Hongzhong Huang, Dr. Xiaodong Wang, Dr. John Whittaker, Dr. Don Koval, Dr. Michael Lipsett, Dr. Larry Kostiuk and Dr. Gregory Levitin.

I am grateful for having the opportunity of studying and working together with a group of amazing people in the Reliability Research Lab and the Advanced Structures and Materials Lab. I

have learned a lot from them, and have really enjoyed the time that we spent together.

Finally, I am grateful to my parents for their constant support and care. And I am truly thankful to my wife, Fei Miao, for her love, encouragement and support.

Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Reliability

The reliability of a device is defined to be the probability that it will perform its intended functions satisfactorily for a specified period of time under specified operating conditions. Today's engineering systems are sophisticated in design and powerful in function. Examples of such systems include aircrafts, space shuttles, telecommunication networks, robots, and manufacturing facilities. Reliability is a critical performance measure of these systems, and it has been emphasized more and more by the industry and the government.

The reliability issue exists because of the uncertainties in engineering systems. There are uncertainties in manufacturing processes. There are also uncertainties in the external operating environment and internal operations of an engineering system. Because of these uncertain factors that can not be completely controlled or predicted, an engineering system can not be 100% guaranteed to perform its intended function at all times. Failures in one form or another sometimes are unavoidable.

Reliability engineering emerged as a separate engineering discipline during the 1950s in the United States [71, 85]. The military electronic systems became

less reliable due to the increasingly complexity of the systems, the new solid state electronics technology, and the increasing use of large number of new component types and new manufacturing processes. Against this background, the Advisory Group on Reliability of Electronic Equipment (AGREE) was set up in 1952 jointly by the US Department of Defence and the electronics industry. The AGREE report published in 1957 concluded that reliability could be specified, allocated and demonstrated, and this report was the symbol of the birth of reliability engineering as a discipline. In the 1960s, there was an increased specialization in the reliability engineering, with branches including reliability theory, reliability physics and structural reliability, and there was a trend from component-level to system-level reliability [85]. In the 1970s, system-level reliability was more emphasized, risk analysis techniques such as probabilistic risk assessment (PRA) was developed, and software reliability was focused on.

The area of reliability engineering focuses on analyzing the reliability of a component or a system and enhancing the reliability through measures in its design, manufacturing and operation. Major topics covered by reliability engineering include: (1) systematically collecting lifetime data of a device from tests or field for reliability evaluation, (2) estimating the reliability of a component based on the collected lifetime data, (3) reliability modeling for practical systems, (4) analyzing system reliability based on the reliability of components, (5) enhancing system reliability through optimal design and maintenance actions, followed by verifying the decisions by thorough analysis and test. A wide range of powerful methods and tools have been developed in this framework, such as Failure Data Analysis, Fault Tree Analysis (FTA), and Failure Modes and Effects Analysis (FMEA).

1.2 Multi-state reliability and several practical examples

Traditional binary reliability theory assumes that a system and its components may take only two possible states: totally failed and perfectly working, or state 0 and state 1. However, many practical systems can actually perform their intended functions at more than two different levels, from perfectly working to completely failed. These kinds of systems are known as multi-state systems [46]. A typical example of a multi-state component is a 300 MW thermal power generating unit in a power station presented by Billinton and Allan [6], as shown in Figure 1.1. Such a generator can work at full capacity, which is its nominal capacity, say 300 MW, when there are no failures at all. Certain types of failures, including the boiler failure and turbine failure, can cause the generator to be completely failed. Other failures, such as the forced draught fan (FD fan) failure and the pulverizer failure, will lead to the generator working at reduced capacities, say at 150 MW, 200 MW or 225 MW. A group of power generating units can be connected in parallel to make a power generation system, as shown in Figure 1.2. The capacity of the power generation system is equal to the sum of the capacities of its components. Thus, the power generation system can be regarded as a multi-state system with multiple possible states. In these cases, the binary reliability theory will be an over-simplification of the actual system or component. Reliability analysis considering multiple possible states is known as multi-state reliability analysis. Multi-state reliability analysis recognizes the multiple possible states of engineering systems. It can map component performances to system performances more accurately, and be able to answer more questions such as the probabilities of a system in different possible states.

Let's describe two more practical examples of multi-state systems:

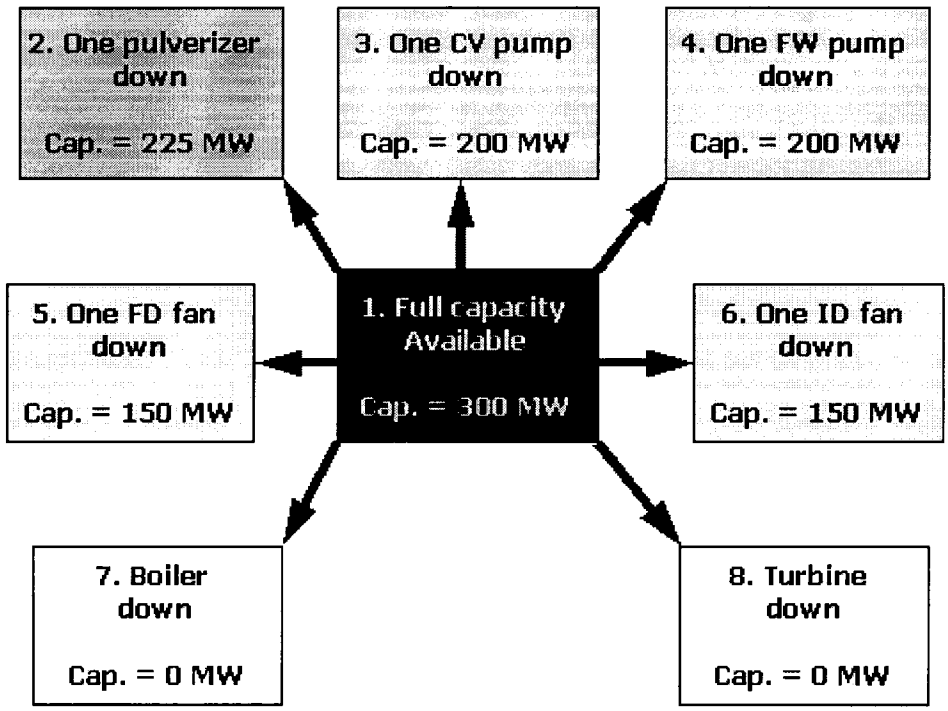

Figure 1.1: A multi-state 300 MW power generating unit [6]

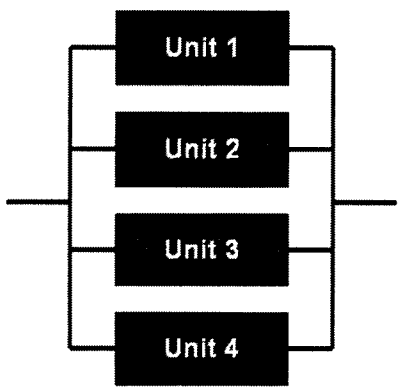

Figure 1.2: A multi-state power generation system

Example 1: coal transmission system. Consider a coal transmission system in a power station which continuously supplies coal to the boilers [55]. The transmission system consists of five basic subsystems: a primary feeder, a set of primary conveyors, a stacker-reclaimer, a secondary feeder, and a set of secondary conveyors. The feeders and the stacker-reclaimer are binary components which can only work at the nominal capacity or totally fail. The capacity of the set of primary (or secondary) conveyors are determined by the availability of individual binary-state conveyors. Thus, the set of primary (or secondary) conveyors is a subsystem with multiple levels of capacity. And the whole coal transmission system is a multi-state system with five subsystems connected in series.

Example 2: wireless communication system. Another typical example of multi-state systems is wireless communication systems [55]. Suppose that a wireless communication system consists of several transmission stations arranged in a sequence. Each station can work at three different levels of capacity. A station is in state 0 if it can not transmit any signals, it is in state 1 if it can transmit signals only to the next station, and it is in state 2 if it can transmit signals to the next two stations. Thus, such a wireless system is actually a multi-state consecutively-connected system with components that can be in three possible states.

1.3 Historical overview of multi-state system reliability

1.3.1 The introduction of multi-state system reliability

The idea of multi-state systems was first touched as early as in 1968 by Hirsch *et al.* [27]. It was systematically introduced and studied in 1970s by Barlow and Wu [3], El-neweihi *et al.* [19] and Ross [81] by considering a component or

a system having more than two possible states. In their work, the primary concepts of multi-state reliability were studied, including system structure function, minimal cut (path) set, relevancy and coherency. These concepts will be discussed in details in Chapter 2. The results by the early studies on multi-state reliability were generalized in the work of Griffith [24], Natvig [67], Hudson and Kapur [38], and Block and Savits [7]. The early advances in multi-state reliability theory was summarized by El-Neweihi and Proschan [20].

1.3.2 Multi-state system reliability analysis

An important issue is how to model practical systems in the multi-state context through careful analysis and definition. Many binary reliability models [66] have been extended to multi-state reliability models, such as the series-parallel system model [3, 47], the k-out-of-n system models [37], the weighted k-out-of-n system model [52], the network system models [55], etc. There might be more than one way to extend a binary reliability model to the multi-state context. For example, in Barlow and Wu's definition of multi-state series-parallel systems [3], the state of a parallel subsystem is equal to the state of the best component. However, in Levitin's definition of multi-state series-parallel systems, the capacity of a parallel subsystem is equal to the sum of the capacities of its constituent components. Under traditional definition of multi-state k-out-of-n:G system [8, 19], the system is in state j or above when at least k components are in state j or above. Huang *et al.* proposed the model of generalized multi-state k-out-of-n:G system by allowing different requirements of the number of components on different states [37, 33]. The model of multi-state consecutive-k-out-of-n system was also redefined [36].

The binary network reliability models have also been extended to multi-state versions by allowing the links and/or the nodes to have more than two possible states [55, 79, 84].

A multi-state system might have discrete states, such as the power generation system we mentioned early in this section. The state of a multi-state system might also be continuous [9, 10, 28, 53, 102], such as the condition of a road section.

One way to analyze multi-state systems is using a binary variable to represent a single state of a component [101]. The problem is that there will be dependencies among variables that characterize the same component. The stochastic process approach is a more universal approach in modeling and evaluating multi-state systems, and it has been applied to the reliability evaluation of power systems [6]. Because the stochastic process approaches require equation solving whose computation burden can be significantly influenced by the number of components and the number of states, the stochastic approach can only be applied to relatively small systems. Levitin *et al.* developed the Universal Generating Function (UGF) approach to evaluate multi-state systems [51, 55], which can be used to deal with a wide range of multi-state systems. Like in the reliability evaluation of binary systems, Monte-Carlo simulation can be used for the evaluation of multi-state systems [79]. But compared to analytical algorithms, the main disadvantage of this approach is that it is not computationally efficient, especially for large systems with a large number of components.

For complex systems with a large number of components and a large number of states, reliability bounds can assist us in efficient decision making. Hudson and Kapur developed methods for exact reliability evaluation and reliabil-

ity bounding of multi-state systems in early 1980's [39, 40, 41, 84]. First one needs to find all the minimal path vectors or the minimal cut vectors [100, 54]. The minimal path (cut) vectors can be used for exact reliability evaluation using the Inclusion-Exclusion (IE) method or the Sum of Disjoint Products (SDP) method.

1.3.3 Optimal design of multi-state systems

For the purpose of maximizing system performance and cost effectiveness, we need to consider the issue of optimal design of multi-state systems. Some complex engineering systems, e.g. a space shuttle, consist of hundreds of thousands of components. These components functioning together form a system. The reliable performance of the system depends on the reliable performance of its components and the system configuration. Basically, there are three ways to enhance reliability of multi-state systems [55]: (1) to provide redundancy, such as adding redundant components in parallel and using redundancy in the form of k-out-of-n systems; (2) to adjust the system configuration while keeping the constituent components the same, such as optimal arrangement of the existing components; (3) to enhance the reliability or performance of the components, either by selecting components with better performance, or by building components with high performance in the design stage which will be built into the manufacturing process. It is almost impossible to enhance the reliability of such a complex system only based on experience. Sophisticated and systematic approaches for system reliability optimization are necessary. In reliability optimization of multi-state systems, we often aim at maximizing system reliability while satisfying physical and economical constraints, or minimizing the cost in development, manufacturing and operation while satisfying

other constraints. El-Neweihi *et al.* first dealt with the problem of optimal allocation of multi-state components [21]. Meng analyzed the optimal allocation of components in multi-state coherent systems, and gave the principle for interchanging components [59]. Zuo *et al.* investigated the replacement-repair policy for multi-state deteriorating products under warranty [104]. They examined the optimal value of the deterioration degree of the item and the length of the residual warranty period, so as to minimize the manufacturer's expected warranty servicing cost per item sold. A simple heuristic algorithm was used to implement the optimization. Gurler and Kaya proposed a maintenance policy for a system with multi-state components, using an approximation of the average cost function to reduce the problem complexity and using a numerical method to solve the formulated optimization problem [25]. A heuristic algorithm was developed to solve the problem. Ramirez-Marquez and Coit proposed a heuristic approach for solving the redundancy allocation problem where the system utility was evaluated using the universal generating function approach [78, 51]. Levitin *et al.* applied the UGF and genetic algorithms (GA) to study a wide range of design optimization problems of multi-state systems [55, 47].

1.3.4 Literature on multi-state system reliability

Multi-state reliability theory has been documented in lots of research papers published in journals such as *IEEE Transactions on Reliability, Reliability Engineering and System Safety* and *IIE Transactions*, and conference proceedings such as the proceedings of the *Industrial Engineering Research Conference* (*IERC*)and the *Annual Reliability and Maintainability Symposium* (*RAMS*). Multi-state reliability theory has also been discussed in a more systematic

and detailed manner in dedicated book chapters and books. Multi-state reliability theory was first systematically discussed in Chapter 13 of the book titled "Optimal Reliability Modeling: Principles and Applications" by Kuo and Zuo [46]. Later, Levitin and Lisnianski gave more details in their book titled "Multi-state System Reliability: Assessment, Optimization and Applications" [55], and in another book titled "Universal Generating Function in Reliability Analysis and Optimization" [47].

1.4 Motivation

One deficiency of current research on multi-state reliability is that efficient evaluation algorithms are not available for certain multi-state systems, such as multi-state k-out-of-n systems and multi-state network systems. It is desirable to have efficient approaches for performance evaluation of complex systems with a large number of components. With such efficient approaches, the system performance evaluation can be completed in a much shorter time. Furthermore, the optimal design of complex systems, which requires iterative system performance evaluations, will become viable.

The other deficiency is that more effective system optimization approaches are needed. Current optimal design of multi-state systems focuses on redundancy allocation. However, state distributions of components (to be discussed in Chapter 2) could also be controlled, and thus could be treated as design variables. Through the joint reliability and redundancy optimization of multi-state systems, we would be able to obtain optimal design in the real sense. In practical situations involving reliability optimization, there often exist mutually conflicting goals such as maximizing system reliability and minimizing system cost and weight. Current multi-state optimization models usually treat one

goal as the objective function of the optimization model and the other goals as constraints. An effective approach needs to be developed to deal with design optimization of multi-state systems considering multiple objectives.

One research goal of this work is to develop effective and efficient methods for reliability analysis, including system reliability modeling and evaluation, of two types of multi-state systems, i.e., multi-state k-out-of-n systems and multi-state two-terminal networks. All of the proposed reliability evaluation algorithms in this work will be based on one common general principle, the recursive computation principle (to be discussed in Chapter 2). The other research goal is to develop effective approaches for reliability based system optimization, focusing on addressing the tradeoff among multiple conflicting objectives and how to conduct joint reliability and redundancy optimization of multi-state series-parallel systems.

1.5 Organization of the thesis

The thesis, with eight chapters, is organized as follows. In Chapter 2, the fundamental knowledge of multi-state reliability is presented, including the basic concepts and tools such as structure function, state distribution, utility, coherency, and minimal path (cut) vectors. Typical multi-state system structures are discussed.

From Chapter 3 to Chapter 5, we discuss how to model and evaluate systems with the k-out-of-n structures. In Chapter 3, we propose methods for the reliability evaluation of generalized multi-state k-out-of-n system model defined by Huang et al [37]. Efficient methods are developed for the exact reliability evaluation of generalized multi-state k-out-of-n systems with identically and independently distributed components (i.i.d. components). A reliability

bounding approach is also developed for fast system reliability approximation. In Chapter 4, another model of multi-state k-out-of-n system with higher flexibility is proposed, together with its reliability evaluation algorithm. In Chapter 5, we propose a unified k-out-of-n model, which can treat the binary k-out-of-n system models, multi-state k-out-of-n models and weighted k-out-of-n models as special cases.

In Chapter 6, we present a method to evaluate reliability of multi-state two-terminal networks given all minimal path vectors. The method is based on the recursive computation principle and the Sum of Disjoint Products (SDP) principle, and it is called the Recursive Sum of Disjoint Products (RSDP) method.

In Chapter 7, we present an approach to deal with multiple conflicting design objectives, such as system reliability and system cost, which are typically involved in reliability based design of multi-state systems. The proposed approach is based on physical programming and genetic algorithms. We also attempt to extend the redundancy allocation to joint reliability and redundancy optimization for multi-state series-parallel systems.

The contributions of this research work are summarized in Chapter 8, and some future work is discussed.

CHAPTER 2

FUNDAMENTALS OF MULTI-STATE SYSTEM RELIABILITY

The fundamental knowledge of multi-state reliability is presented in this chapter, including some basic concepts and tools such as structure function, state distribution, utility, and minimal path (cut) vectors. Typical multi-state system structures are discussed, including series-parallel systems, k-out-of-n systems and two-terminal network systems. We also present in this chapter the general problem of system reliability evaluation and the general framework of a recursive algorithm.

2.1 Basic concepts

Unless otherwise stated, the materials in this section is based on [46].

2.1.1 Structure function

A multi-state component or system can be in $M+1$ possible states: 0, 1, ..., M. Suppose a system has n components. We use a vector $\boldsymbol{x} = (x_1, x_1, ..., x_n)$ to represent the component states, where component i is in state x_i. "Structure function" represents the relationship between the component states and the

system state. $\phi(\boldsymbol{x})$ denotes the system state as a function of the component states.

2.1.2 State distribution

In binary reliability theory, the reliability of a component or a system actually refers to the probability in state 1. In the context of multi-state reliability, there are more than two possible states. We use "state distribution" instead of "reliability" to denote the probability of a multi-state entity performing its intended function at different performance levels. Suppose a component or a system has totally $M+1$ states. We use vector $\boldsymbol{p} = (p_0, p_1, ..., p_M)$ to represent the state distribution of the component or system, where p_j denotes the probability of the component of system in state j $(j = 1, 2, ..., M)$.

2.1.3 Relevancy and coherency

A component is relevant if the component state can somehow affect the system state. Coherency represents the qualitative relationship between component states and system states. A multi-state system is called a coherent system if an improvement in any component will not make the whole system worse and all the components are relevant.

2.1.4 Utility as a system performance measure

In binary-state systems, system reliability is a very important performance measure. Performance measures for multi-state systems have been studied [2, 10, 57], and utility has been a widely used performance measure for multi-state systems. A multi-state system can be in different discrete states, in contrast to that a binary system can only be in two possible states. When a

multi-state system is in a certain state s, it can produce a certain amount of benefit. Such benefit is called the system utility when it is in state s, and is denoted by u_s. A multi-state system can be in different states with different probabilities. Thus, the measure we are interested in is the expected system utility. System utility can be calculated as follows [2]:

$$U = \sum_{s=0}^{M} u_s \cdot Pr(\phi(\boldsymbol{x}) = s) \tag{2.1}$$

where U is the expected system utility, and u_s is the utility when system is in state s . In practical applications, u_s is usually known. Thus, the key issue in multi-state system utility evaluation is to evaluate the probabilities of the system in different possible states.

2.1.5 Minimal path (cut) set

Using minimal path set and minimal cut set is a general way for reliability evaluation of multi-state systems. Before defining minimal path (cut) sets for multi-state systems, we will first give definitions of minimal path (cut) vector.

Definition 2.1 (Natvig [67]) *A vector* $\mathbf{x}$ *is called a minimal path vector to level* j *if* $\phi(\mathbf{x}) \geq j$ *and* $\phi(\mathbf{y}) < j$ *for all* $\mathbf{y} < \mathbf{x}$. *A vector* $\mathbf{x}$ *is called a minimal cut vector to level* j *if* $\phi(\mathbf{x}) < j$ *and* $\phi(\mathbf{y}) \geq j$ *for all* $\mathbf{y} > \mathbf{x}$.

$\mathbf{y} < \mathbf{x}$ means $y_i \leq x_i$ for any element index i, and $y_i < x_i$ for at least one element index i. $\mathbf{y} > \mathbf{x}$ means $y_i \geq x_i$ for any element index i, and $y_i > x_i$ for at least one element index i. The set of all minimal path (cut) vectors is called the minimal path (cut) set.

Note that the minimal path (cut) set defined here for the purpose of multi-state system analysis is different from its well known use for the traditional

binary system reliability analysis. A minimal cut set for a binary system is a smallest set of components such that if all the components are simultaneously failed the system is failed. A minimal path set for a binary system is a smallest set of components such that if all of the components are simultaneously functioning the system is functioning. A minimal path set in the traditional binary reliability analysis is equivalent to a minimal path vector in the multi-state reliability analysis, and it can actually be written in the form of minimal path vector. For example, let us consider a binary system with five components, component 1 to component 5. Suppose a minimal path set for the binary system is {1, 3, 4}, meaning the minimal path set includes component 1, 3 and 4. Using the terminology in multi-state system analysis, we can say that a minimal path vector for the system is {1, 0, 1, 1, 0}.

2.2 Typical structures of multi-state systems

When we are dealing with a practical system, the first step is to identify the type of the system through reasonable simplification and careful analysis. After that, we can use the available results of the identified system to make further analysis. In this section, some typical system structures are presented. For each system structure, first we discuss it in the binary reliability framework where the system and the components can only take two possible states: state 1 (working state) and state 0 (failed state). Then the system structure is discussed in the multi-state reliability framework. We focus our discussions on four typical structures of multi-state systems to be studied in this thesis work: series-parallel systems, k-out-of-n systems, weighted k-out-of-n systems, and two-terminal network systems.

2.2.1 Series-parallel systems

A binary series-parallel system has N subsystem connected in series, and each subsystem is a parallel system [66, 46]. A parallel system is working (in state 1) as long as at least one of its components is working (in state 1). In another word, a parallel system is failed (in state 0) if all of its components are failed (in state 0). A series system is working (in state 1) if all of its components are working (in state 1).

The multi-state series-parallel system defined by Barlow and Wu [3] has been widely studied. Its structure is shown in Fig. 2.1. A multi-state series-parallel system consists of subsystems, S_1 to S_N, connected in series. Each subsystem, say S_i, has some components connected in parallel. The following assumptions are used: (1) The components in a subsystem are independent. (2) The components and the system may be in M+1 possible states, namely, 0, 1, 2, ..., M. (3) The multi-state series-parallel systems under consideration are coherent systems. According to the multi-state system definition of Barlow and Wu [3], the state of a parallel system is defined to be the state of the best component in the system, and the state of a series system is defined to be the state of the worst component in the system. Hence, the state of the series-parallel system is shown in Fig. 2.1 is

$$\phi(\mathbf{x}) = \min_{1\le i\le N} \max_{1\le j\le n_i} x_{ij} \tag{2.2}$$

where x_{ij} is the state of the component j of subsystem i.

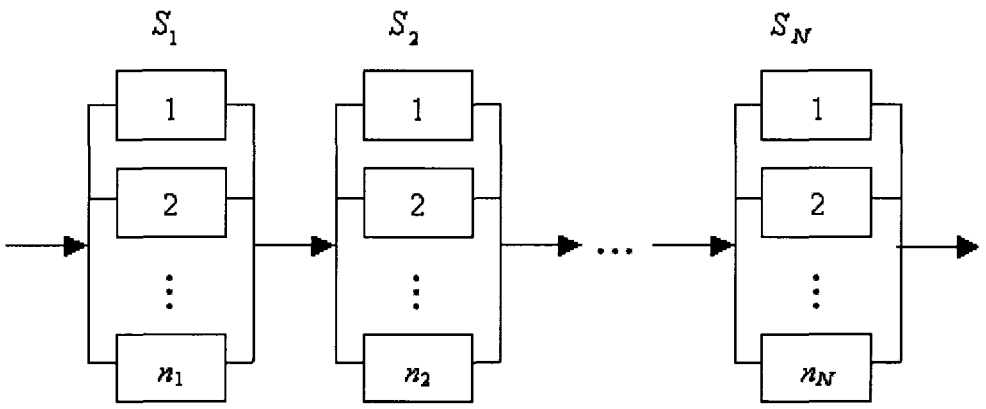

Figure 2.1: A multi-state series-parallel system

2.2.2 k-out-of-n systems

The k-out-of-n system structure is a very popular type of redundancy in fault tolerant systems, with wide applications in both industrial and military systems. Examples of such fault tolerant systems include the multi-engine system in an airplane, the multi-display system in a cockpit, and the multi-transmitter system in a communication system [46]. We first present the binary k-out-of-n system models, then discuss k-out-of-n system models in the multi-state context.

Definition 2.2 *An n-component system is called a binary k-out-of-n:G system if it is working whenever at least k components are working. An n-component system is called a binary k-out-of-n:F system if it is failed whenever at least k components are failed.*

Efficient reliability evaluation algorithms for binary k-out-of-n systems with independent components have been provided by Barlow and Heidtmann [4] and Rushdi [82], as follows:

$$R(n,k) = p_n \cdot R(n-1,k-1) + q_n \cdot R(n-1,k), \tag{2.3}$$

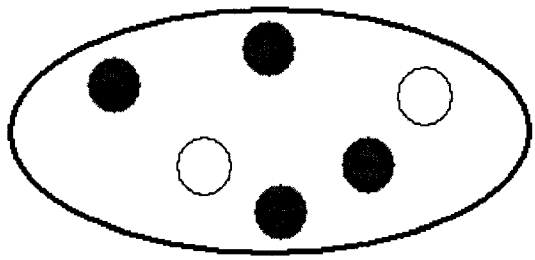

Figure 2.2: A multi-state k-out-of-n system

where $R(n,k)$ is the recursive function, representing the reliability of a k-out-of-n:G system with n components, p_n is the reliability of component n, and $q_n = 1 - p_n$. The boundary conditions are:

$$\begin{aligned} R(n,0) &= 1, \\ R(n,k) &= 0, \quad \text{for} \quad 0 < n < k. \end{aligned} \tag{2.4}$$

Binary k-out-of-n systems models have been extended to multi-state k-out-of-n system models by allowing components and systems to take more than two possible states [46, 55]. A multi-state k-out-of-n system is illustrated in Fig. 2.2. We only show 6 components in the figure, but there can be any number of components in a k-out-of-n systems. From Huang *et al.*, an n-component system is called a generalized multi-state k-out-of-n:G system if $\phi(\mathbf{x}) \geq j$ $(1 \leq j \leq M)$ whenever there exists an integer value l $(j \leq l \leq M)$ such that at least k_l components are in state l or above.

When $k_1 \leq k_2 \leq \cdots \leq k_M$, the system is called an increasing multi-state k-out-of-n:G system [37]. When $k_1 \geq k_2 \geq \cdots \geq k_M$, the system is called a decreasing multi-state k-out-of-n:G system. Let's use an example to illustrate the modeling of an engineering system as a decreasing multi-state k-out-of-n:G system model.

Example 2.1 *Consider a power station with three generators. Each generator is treated as a component and there are 3 components in this system. Each generator may be in four possible states, 0, 1, 2, and 3. When a generator is in state 3, it is capable of generating 10 megawatts in power output; in state 2, 3 megawatts; state 1, 1 megawatt; and state 0, 0 megawatt. The total power output of the system is equal to the sum of the power output from all three generators. The system may also be in four different states: 0, 1, 2, and 3. When the total output is greater than or equal to 10 megawatts, the system is considered to be in state 3; otherwise but greater than or equal to 6 megawatts, state 2; otherwise but greater than or equal to 3 megawatts, state 1; otherwise, state 0. Based on these descriptions, the system can be considered to be a decreasing multi-state k-out-of-n:G system with the following parameters:*

$$n = 3, \quad M = 3, \quad k_1 = 3, \quad k_2 = 2, \quad k_3 = 1.$$

Using the terminology of the definition of generalized multi-state k-out-of-n:G system, we can describe this model as follows: The system is in state 3 whenever at least 1 component is in state 3; in state 2 or above whenever either at least 1 component is in state 3 or at least 2 components are in state 2 or above; in state 1 or above whenever at least 1 component is in state 3, or at least 2 components are in state 2 or above, or at least 3 components are in state 1 or above.

2.2.3 Weighted k-out-of-n systems

A weighted k-out-of-n system is a special type of k-out-of-n systems. Wu and Chen generalized the binary k-out-of-n system models to the binary weighted k-out-of-n models [97].

Definition 2.3 *In a binary weighted k-out-of-n system, component i carries a weight of w_i, $w_i > 0$ for $i = 1, 2, \ldots, n$. The total weight of all components is w, $w = \sum_{i=1}^{n} w_i$. The system works if and only if the total weight of working components is at least a pre-specified threshold value k.*

In this definition, "weight" does not refer to the physical weight. Instead it refers to the utility defined earlier in this chapter.

A recursive equation is provided by Wu and Chen [97]. We use $R_w(i, j)$ to represent the probability that a system with j components can output a total weight of at least i. Then, $R_w(k, n)$ is the reliability of the weighted k-out-of-n:G system. The following recursive equation can be used for reliability evaluation of such systems.

$$R_w(i, j) = p_j R_w(i - w_j, j - 1) + q_j R_w(i, j - 1), \tag{2.5}$$

which requires the following boundary conditions:

$$R_w(i, j) = 1, \quad \text{for} \quad i \leq 0, \quad j \geq 0, \tag{2.6}$$

$$R_w(i, 0) = 0, \quad \text{for} \quad i > 0. \tag{2.7}$$

An important variation of binary weighted k-out-of-n systems is weighted voting systems. A weighted voting system consists of n units that each provide a binary decision (0 or 1) or abstain from voting (x). The system output is 1 if the cumulative weight of all 1-opting units is at least a pre-specified fraction τ of the cumulative weight of all non-abstaining units. Otherwise the system output is 0 [69, 70, 47].

A definition of a multi-state weighted k-out-of-n system is as follows [52]. A multi-state weighted k-out-of-n system can be regarded as a system with n

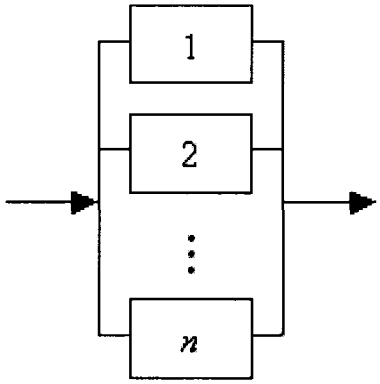

Figure 2.3: A multi-state weighted k-out-of-n system

multi-state components connected in parallel, as in Fig. 2.3. Each component has multiple possible states with multiple levels of capacities. The capacity of the system is equal to the sum of the capacities of its components. An algorithm is developed in [52] for the reliability evaluation of multi-state weighted k-out-of-n systems.

2.2.4 Multi-state networks

Reliability is a critical measure for evaluating the performance of network systems and for making decisions such as maintenance scheduling [54, 16]. Many network systems, such as power generation and transmission systems, oil and gas supply systems, and communication systems, consist of components which can work at different levels of capacity. These systems are regarded as multi-state networks. In this research work, we consider a network satisfying the following assumptions: (1) all nodes are perfect; and (2) all links are directed and failure prone. The capacity of a link is an independent discrete random variable and may take non-negative integer values following a certain probability distribution. The most general network reliability problem is the so-called all-terminal network reliability problem, which studies the connectivity among

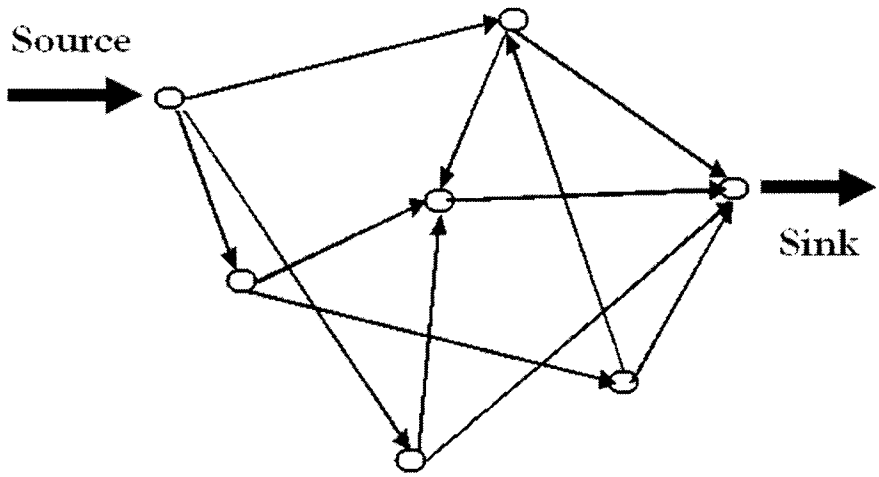

Figure 2.4: A multi-state network system

all the nodes of a network [86]. In this research work, however, we limit our discussions to reliability analysis of two-terminal networks [79, 80, 84]. This is a classical network reliability problem with a broad range of practical applications. A two-terminal network is shown in Fig. 2.4. We are interested in the flow from a single source node to a single sink node.

2.3 The general problem of reliability evaluation of multi-state systems

As shown in Figure 2.5, the general problem of reliability evaluation of multi-state systems is as follows: given the component state distributions and the system structure function $\phi(\boldsymbol{X})$, find the system state distribution.

"Component state distributions" describe the probabilities of the components in different possible states. Suppose that there are n components in a multi-state system under consideration. Component i might be in $M_i + 1$ possible states from state 0 to state M_i. p_{ij} is used to represent the probability of

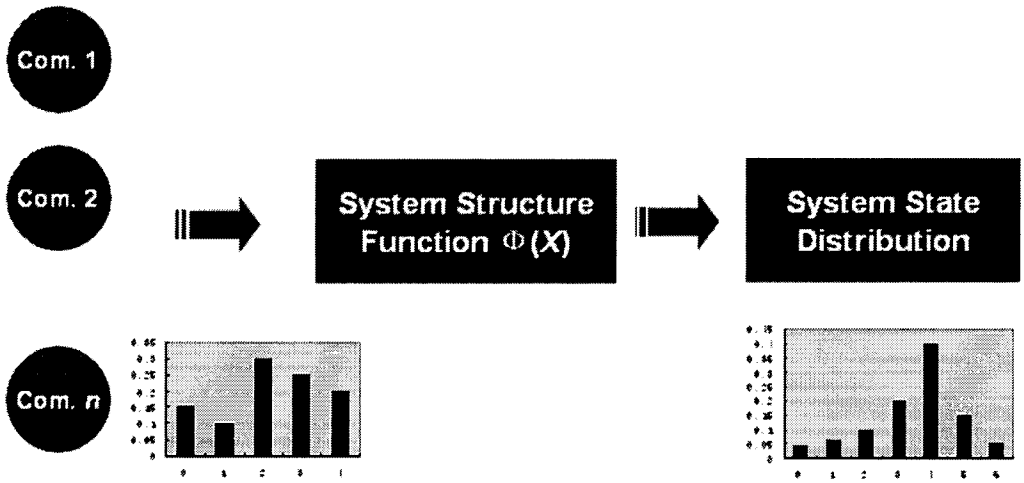

Figure 2.5: Reliability evaluation of multi-state systems

component i $(1 \leq i \leq n)$ in state j $(0 \leq j \leq M_i)$. We have

$$\sum_{j=0}^{M_i} p_{ij} = 1.$$

Thus, component state distributions can be described by p_{ij} $(1 \leq i \leq n,$ $0 \leq j \leq M_i)$.

"System structure function $\phi(\boldsymbol{X})$" gives how to determine the system state based on the component states. $\boldsymbol{X}$ is a vector with n elements, where element i $(1 \leq i \leq n)$ indicates the state of the i^{th} component.

"System state distribution" gives us the probabilities of the system in different possible states. Suppose that the system has M_s possible states. p_{sj} are used to denote the probability of the system in state j $(0 \leq j \leq M_s)$.

We need to note that the state distributions of components and systems we study in this thesis are constant values. There are two cases for the state distributions to take constant values. The first case is when we have specified mission time t. The other case is when a component or a system is in steady-

state [55, 47]. Take for example p_{sj}, the probability of the system in state j. This probability is actually a function of time $p_{sj}(t)$. When time t is large enough, the system will reach a steady state, and $p_{sj}(t)$ will converge to a specific value p_{sj}.

2.4 Fundamental elements of recursive algorithms

Recursive algorithms are widely used for system reliability evaluations [46]. As to be shown later, the reliability evaluation algorithms for multi-state k-out-of-n systems and network systems to be developed in this research are all recursive algorithms [105, 95, 52].

In this section, we would like to identify and discuss three fundamental elements in a recursive algorithm: recursive function, updating procedure and boundary conditions. We would like to use the recursive algorithm for binary k-out-of-n systems, Equation (2.3) and (2.4), as an example to illustrate these elements.

2.4.1 Recursive function

Recursive function is the key function that calls itself in the recursive algorithm. It has one or multiple input parameters, which change during the course of the recursive algorithm. A recursive function with a set of input parameters is calculated via several recursive functions with sets of simpler input parameters.

In the recursive algorithm for binary k-out-of-n systems in Equation (2.3) and (2.4), $R(n,k)$ is the recursive function, in which n and k are the parameters. As shown in the equations, the recursive function $R(n,k)$ is calculated via recursive functions $R(n-1,k-1)$ and $R(n-1,k)$ which have simpler input

parameters.

2.4.2 Updating algorithm

The updating algorithm decides how the recursive function calls itself. In another word, the updating algorithm decides the relationship between a recursive function with certain parameters and recursive functions with different parameters, which are typically less complex.

In the recursive algorithm for binary k-out-of-n systems, Equation (2.3) shows the updating algorithm, that is, how to calculate the recursive function $R(n,k)$ via two recursive functions $R(n-1,k)$ and $R(n-1,k-1)$ with simpler input parameters.

2.4.3 Boundary conditions

When one of the boundary conditions is met, the recursive function will take a certain value, or can be determined in a specific and simple way.

In the recursive algorithm for binary k-out-of-n systems, Equation (2.4) shows the boundary conditions. There are two boundary conditions in this case. The first one is when $k = 0$, the value of the recursive function $R(n,k)$ is 1. The other boundary condition is when $0 < n < k$, the value of the recursive function $R(n,k)$ is 0.

Chapter 3

Reliability Analysis of Generalized Multi-State k-out-of-n Systems

In this chapter, we propose methods for the reliability evaluation of generalized multi-state k-out-of-n systems defined by Huang et al [37]. We focus on generalized multi-state k-out-of-n systems with identically and independently distributed (i.i.d.) components. First the method for the exact reliability evaluation of generalized multi-state k-out-of-n systems is presented. Then, a reliability bounding approach is developed for fast system reliability approximation.

3.1 Reliability evaluation of generalized multi-state k-out-of-n systems

In this section, we present an algorithm for the reliability evaluation of generalized multi-state k-out-of-n systems with i.i.d. components. The materials in this section have been published in [105] and [94].

Assumptions:

- The state space of each component and the system is $\{0, 1, 2, \ldots, M\}$.

- The states of all components are identically and independently distributed (i.i.d.) random variables.
- The state of the system is completely determined by the states of the components.
- A lower state level represents a worse or equal performance of the component or the system.

Notation:

- x_i: state of component i, $x_i = j$ if component i is in state j, $0 \leq j \leq M$, $1 \leq i \leq n$
- $\mathbf{x}$: an n-dimensional vector representing the states of all components, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$
- $\phi(\mathbf{x})$: state of the system, which is also called the structure function of the system, $0 \leq \phi(\mathbf{x}) \leq M$
- p_j: probability that a component is in state j when all components are i.i.d.
- P_j: probability that a component is in state j or above when all components are i.i.d.
- Q_j: $1 - P_j$
- $R_{s,j}$: $\Pr(\phi(\mathbf{x}) \geq j)$
- $Q_{s,j}$: $1 - R_{s,j}$
- $r_{s,j}$: $\Pr(\phi(\mathbf{x}) = j)$

- $R(k, n; j)$: probability that the k-out-of-n:G system is in state j or above
- N: number of components of a nominal increasing multi-state k-out-of-n:F system
- m: number of possible states of a nominal increasing multi-state k-out-of-n:F system minus 1
- $\bar{k}_j$: the k value to state j for a nominal increasing multi-state k-out-of-n:F system
- $\bar{p}_j$: the probability of the components in state j for a nominal increasing multi-state k-out-of-n:F system
- $Q(\bullet)$: the recursive function used by the proposed recursive algorithm, $Q = Q(m, N, \boldsymbol{k}, \boldsymbol{p})$

3.1.1 Introduction

Under traditional definition of multi-state k-out-of-n:G system [8, 19], the system is in state j or above when at least k components are in state j or above. Huang *et al.* [37] proposed the generalized multi-state k-out-of-n:G system model, where there can be different k values with respect to different states. That is, the system is in state j or above if there exists an integer value l ($j \leq l \leq M$) such that at least k_j components in state j or above. The generalized multi-state k-out-of-n:G model allows describing practical multi-state systems with more flexibilities. The generalized multi-state k-out-of-n:G model was later extended to multi-state consecutive-k-out-of-n system as a special case [36]. Huang *et al.* [37] also provided an algorithm to calculate the state distribution of generalized multi-state k-out-of-n:G systems. However,

this performance evaluation algorithm is enumerating in nature, and therefore it is not efficient. In addition, this algorithm is applicable only when the k_i values follow a monotonic pattern. A more efficient and more general algorithm is needed for the evaluation of generalized multi-state k-out-of-n:G systems.

Under the traditional definition of multi-state k-out-of-n systems, there is an equivalent multi-state k-out-of-n:F system with respect to each multi-state k-out-of-n:G system [46]. We will first present the definition of generalized multi-state k-out-of-n:F systems, and there is an equivalent relationship between generalized multi-state k-out-of-n:G and F systems. The general form of minimal cut vector for generalized multi-state k-out-of-n:F systems is presented. Based on minimal cut vectors, we develop an efficient recursive algorithm to evaluate the state distributions of a generalized multi-state k-out-of-n:F system. Furthermore, a generalized multi-state k-out-of-n:G system can first be transformed into its equivalent generalized multi-state k-out-of-n:F system, and then be evaluated using the proposed recursive algorithm. Several numerical examples are given to illustrate the effectiveness and efficiency of the proposed recursive algorithms.

3.1.2 The generalized multi-state k-out-of-n:G system defined by Huang *et al.*

Huang *et al.* [37] propose the definition of generalized multi-state k-out-of-n:G system and develop reliability evaluation algorithms for this multi-state k-out-of-n:G system model.

Definition 3.4 (Huang *et al.* [37]) *An n-component system is called a generalized multi-state k-out-of-n:G system if* $\phi(\mathbf{x}) \geq j$ $(1 \leq j \leq M)$ *whenever there exists an integer value* l $(j \leq l \leq M)$ *such that at least* k_l *components*

are in state l or above.

A generalized multi-state k-out-of-n:G system can also be called multi-state $(k_1, k_2, \ldots, k_M)$-out-of-n:G system, or $(k_1, k_2, \ldots, k_M)$-out-of-n:G system. Though the k_j values are not necessarily in monotone ordering, they consider the following two special cases of this definition:

- When $k_1 \leq k_2 \leq \cdots \leq k_M$, the system is called an increasing multi-state k-out-of-n:G system. In this case, for the system to be in a higher state level j or above, a larger number of components must be in state j or above. In other words, there is an *increasing* requirement on the number of components that must be in a certain state or above for the system to be in a higher state level or above. That is why it is called an increasing multi-state k-out-of-n:G system.

- When $k_1 \geq k_2 \geq \cdots \geq k_M$, the system is called a decreasing multi-state k-out-of-n:G system. In this case, for a higher system state level j, there is a *decreasing* requirement on the number of components that must be in state level j or above.

When k_j is a constant, i.e., $k_1 = k_2 = \cdots = k_M = k$, the structure of the system is the same for all system state levels. This reduces to the definition of the simple multi-state k-out-of-n:G system studied by El-Neweihi [19] and Boedigheimer and Kapur [8]. Such systems are called constant multi-state k-out-of-n:G systems. All the concepts and results of binary k-out-of-n:G systems can be easily extended to the constant multi-state k-out-of-n:G systems. The constant multi-state k-out-of-n:G system is treated as a special case of the increasing multi-state k-out-of-n:G system in our later discussions.

For an increasing multi-state k-out-of-n:G system, i.e., $k_1 \leq k_2 \leq \ldots \leq k_M$, Definition 3.4 can be rephrased as follows:

> *$\phi(\mathbf{x}) \geq j$ if and only if at least k_j components are in state j or above.*

If at least k_j components are in state j or above (these components can be considered "working" as far as state level j is concerned), then the system will be in state j or above (the system is considered to be "working") for $1 \leq j \leq M$. The only difference between this case of Definition 3.4 and the constant multi-state k-out-of-n:G system structure is that the number of components required to be in state j or above for the system to be in state j or above may change from state to state. Other characteristics of this case of the generalized multi-state k-out-of-n:G system are exactly the same as those of a constant multi-state k-out-of-n:G system structure. Algorithms for binary k-out-of-n:G system reliability evaluation can also be extended to the increasing multi-state k-out-of-n:G system for reliability evaluation. Especially, when all components are i.i.d., we have

$$R_j(k_j, n) = P_j R_j(k_j - 1, n - 1) + Q_j R_j(k_j, n - 1), \tag{3.1}$$

where $R_j(b, a)$ is the probability that at least b out of a components are in state j or above. The following boundary conditions are needed for equation (3.1).

$$R_j(0, a) = 1, \quad \text{for} \quad a \geq 0, \tag{3.2}$$

$$R_j(b, a) = 0, \quad \text{for} \quad b > a > 0. \tag{3.3}$$

Since all components are i.i.d., the probability that the system is in j or above, R_{sj}, can also be expressed as:

$$R_{sj} = \sum_{k=k_j}^{n} \binom{n}{k} P_j^k (1 - P_j)^{n-k}. \tag{3.4}$$

For a decreasing multi-state k-out-of-n:G system, i.e., $k_1 \geq k_2 \geq \ldots \geq k_M$, the wording of its definition is not as simple. The system is in level M if at least k_M components are in level M. The system is in level $M-1$ or above if at least k_{M-1} components are in level $M-1$ or above or at least k_M components are in level M. Generally speaking, the system is in level j or above ($1 \leq j \leq M$) if at least k_j components are in level j or above, at least k_{j+1} components are in level $j+1$ or above, at least k_{j+2} components are in level $j+2$ or above, ..., or at least k_M components are in level M. In this case, the definition of the system can be rephrased in terms of the system being exactly in a certain state:

> *$\phi(\mathbf{x}) = j$ if and only if, at least k_j components are in state j or above and at most $k_l - 1$ components are at state l or above for $l = j+1, j+2, \cdots, M$ where $j = 1, 2, \cdots, M$.*

When all of the components are i.i.d., Huang *et al.* [37] provide the following equation for evaluation of the probability that the system is in state j for $j = 1, 2, \ldots, M$.

$$r_{sj} = \sum_{k=k_j}^{n} \binom{n}{k} Q_j^{n-k} \left(p_j^k + \sum_{l=j+1, k_l>1}^{M} \beta_l(k) \right), \tag{3.5}$$

where $\beta_l(k)$ is the probability that at least one and at most $k_l - 1$ components are in state l, at most $k_u - 1$ components are in state u for $j+1 \leq u < l$, the

total number of components that are at states between j and l inclusive is k, and the system is in state j. The following equation was proposed to calculate $\beta_l(k)$ [37]

$$\begin{aligned}\beta_l(k) &= \sum_{i_1=1}^{k_l-1}\binom{k}{i_1}p_l^{i_1}\times\sum_{i_2=0}^{k_{l-1}-1-I_1}\binom{k-I_1}{i_2}p_{l-1}^{i_2}\times\sum_{i_3=0}^{k_{l-2}-1-I_2}\binom{k-I_2}{i_3}p_{l-2}^{i_3}\times\cdots \\ &\times\sum_{i_{l-j}=0}^{k_{j+1}-1-I_{l-j-1}}\binom{k-I_{l-j-1}}{i_{l-j}}p_{j+1}^{i_{l-j}}\times p_j^{k-I_{l-j}}. \qquad (3.6)\end{aligned}$$

The algorithm above by Huang *et al.* [37] provides a way to evaluate the state distribution of decreasing multi-state k-out-of-n:G systems. However, this algorithm is enumerating in nature, and therefore it is not efficient. In addition, this algorithm is applicable only when the k_i values follow a monotonic pattern, that is, decreasing or increasing. A more efficient and more general algorithm is needed for the evaluation of multi-state k-out-of-n:G systems.

An example to illustrate the modeling of an engineering system as a decreasing multi-state k-out-of-n:G model, a power generation system example, is provided in Section 1.2. Another example of a decreasing multi-state k-out-of-n:G system, a mining operation example, is given in Huang and Zuo [34].

3.1.3 The generalized multi-state k-out-of-n:F systems

3.1.3.1 Definition of generalized multi-state k-out-of-n:F systems

Given the results reported in Huang *et al.* [37] as we have reviewed in the previous section, we observe that the definition of the generalized multi-state k-out-of-n:G system can be stated in terms of the system's state being below

a certain level, as follows:

> $\phi(\mathbf{x}) < j$ *if and only if at least* $n - k_l + 1$ *components are below state* l *for all* l *such that* $j \leq l \leq M$.

In this perspective of linking system state to the states of the components, we are focusing on the events that the states of both the system and the components are below a certain level. This corresponds to the definition of a k-out-of-n:F system in the binary context. Thus, we propose the following definition of the generalized multi-state k-out-of-n:F system.

Definition 3.5 *An n-component system is called a generalized multi-state k-out-of-n:F system if* $\phi(\mathbf{x}) < j$ $(1 \leq j \leq M)$ *whenever the states of at least* k_l *components are below* l *for all* l *such that* $j \leq l \leq M$.

Here we also consider the following two special cases of this definition:

Definition 3.6 *A generalized multi-state k-out-of-n:F system is called increasing multi-state k-out-of-n:F system if* $k_1 < k_2 < \cdots < k_M$.

Definition 3.7 *A generalized multi-state k-out-of-n:F system is called decreasing multi-state k-out-of-n:F system if* $k_1 \geq k_2 \geq \cdots \geq k_M$.

Note that in the definition of increasing multi-state k-out-of-n:F system, k_j is strictly increasing. This definition is convenient for the descriptions of the minimal cut vectors of generalized multi-state k-out-of-n:F systems and the proposed recursive algorithms which will be presented later.

Like in the binary case, the generalized multi-state k-out-of-n:G system defined in Definition 3.4 and the generalized multi-state k-out-of-n:F system defined in Definition 3.5 can be considered to be mirror images of each other.

For a given set of n components with given state distributions, the state distribution of a generalized multi-state k-out-of-n:G system is equal to the state distribution of a generalized multi-state k-out-of-n:F system if $k_j^F = n - k_j^G + 1$ for $j = 1, 2, \ldots, M$. As a result, an increasing multi-state k-out-of-n:G system becomes a decreasing multi-state $(n-k+1)$-out-of-n:F system and a strictly decreasing multi-state k-out-of-n:G system becomes an increasing multi-state $(n-k+1)$-out-of-n:F system.

Suppose that we are interested in evaluating the state distribution of a multi-state k-out-of-n:G system with given k_j values for $1 \leq j \leq M$ and given component state distributions. If it is an increasing multi-state k-out-of-n:G system, we may use equation (3.1) directly to evaluate the state distribution of the system. If it is a decreasing multi-state k-out-of-n:G system, we may use equation (3.5) directly to evaluate the state distribution of the system. However, equation (3.5) is not efficient. In this section, we will present an efficient recursive algorithm to evaluate the state distribution of the generalized multi-state k-out-of-n:F systems, and thus we can use them to evaluate generalized multi-state k-out-of-n:G systems as well.

Consider a strictly decreasing multi-state k-out-of-n:G system with $k_1 > k_2 > \ldots > k_M$ and i.i.d. component state distributions. Note that here we only consider the case in which k_j are strictly decreasing. The cases in which k_j are not strictly decreasing are regarded as general generalized multi-state k-out-of-n:G systems, if they are not increasing multi-state k-out-of-n:G systems. We can find $k'_j = n - k_j + 1$ for $j = 1, 2, \ldots, M$. Now we can concentrate on an increasing multi-state k-out-of-n:F system with $k'_1 < k'_2 < \ldots < k'_M$. The state distribution of this increasing multi-state k-out-of-n:F system is equal to the state distribution of the original multi-state k-out-of-n:G system.

Now, we will focus on the structure of an increasing multi-state k-out-of-n:F system with $1 \leq k_1 < k_2 < \ldots < k_M \leq n$, as defined in Definition 3.6.

3.1.3.2 Minimal cut vectors of increasing multi-state k-out-of-n:F systems

In this section, we will present the form of minimal cut vector for increasing multi-state k-out-of-n:F systems. A minimal cut vector to system state M has the following form:

$$\underbrace{(\underbrace{M-1, M-1, \ldots, M-1}_{k_M}, M, \ldots, M)}_{n},$$

Let us use $\boldsymbol{v}$ to represent this vector. From the definition of generalized multi-state k-out-of-n:F system, we can see that $\phi(\boldsymbol{v}) < M$ since there are k_M components in states below M. When any component's state is raised, if it can be, there will be less than k_M components which are in states below M, that is, we will have $\phi(\boldsymbol{v}) \geq M$. Therefore, vector $\boldsymbol{v}$ is a minimal cut vector to system state M. Based on the definition of generalized multi-state k-out-of-n:F system, all permutations of the elements of this minimal cut vector are all minimal cut vectors to system state M.

Similarly, a minimal cut vector to system state $M-1$ has the following form:

$$\underbrace{\underbrace{(\underbrace{M-2, M-2, \ldots, M-2}_{k_{M-1}}, M-1, \ldots, M-1}_{k_M}, M, \ldots, M)}_{n}.$$

All permutations of the elements of this minimal cut vector are all minimal cut vectors to system state $M-1$.

Finally, a minimal cut vector to system state 1 has the following form:

$$(\underbrace{0,0,\ldots,0}_{k_1},\underbrace{1,\ldots,1}_{k_2},\ldots,\underbrace{M-1,\ldots,M-1}_{k_M},\underbrace{M,\ldots,M}_{n}).$$

Generally speaking, a minimal cut vector to system state j has the following form:

$$(\underbrace{j-1,j-1,\ldots,j-1}_{k_j},\underbrace{j,\ldots,j}_{k_{j+1}},\ldots,\underbrace{M-1,\ldots,M-1}_{k_M},\underbrace{M,\ldots,M}_{n}).$$

We will use the symbol $\star$ as a superscript to this minimal cut vector to represent all permutations of the elements of this minimal cut vector, which are all minimal cut vectors to system state j, that is

$$(j-1,j-1,\ldots,j-1,j,\ldots,j,\ldots,M-1,\ldots,M-1,M,\ldots,M)^{\star}$$

represents all minimal cut vectors to state j, where $1 \leq j \leq M$.

3.1.3.3 Minimal cut vectors of general multi-state k-out-of-n:F systems

For a general multi-state k-out-of-n:F system, the k_j values are not necessarily in a monotonic ordering. The minimal cut vector form of general multi-state k-out-of-n:F system is similar to that of increasing multi-state k-out-of-n:F

system, except that we need to do some transformations first.

Considering the general case, here we will try to find the minimal cut vectors with respect to state j for a general multi-state k-out-of-n:F system with values k_1, k_2, ..., k_M. First we need to find the k_l values ($j \leq l \leq M$) which are in strictly increasing order. For instance, if we find that $k_j < k_a < k_b$ ($j < a < b \leq M$), it means that $k_l \leq k_j$ ($j \leq l < a$), $k_l \leq k_a$ ($a \leq l < b$), and $k_l \leq k_b$ ($b \leq l \leq M$). Then a minimal cut vector to state j has the following form:

$$(\underbrace{j-1, j-1, \ldots, j-1}_{k_j}, \underbrace{a-1, \ldots, a-1}_{k_a}, \underbrace{b-1, \ldots, b-1}_{k_b}, \underbrace{M, \ldots, M}_{n}) \quad (3.7)$$

Compared with the minimal cut vector structure to state j of increasing multi-state k-out-of-n:F system, it can be considered that k_l ($j < l < a$) are "absorbed" by k_j, k_l ($a < l < b$) are "absorbed" by k_a, and k_l ($b < l \leq M$) are "absorbed" by k_b. All permutations of the elements of this minimal cut vector are all minimal cut vectors to system state j, which are represented by attaching the symbol $\star$ as a superscript to the minimal cut vector above.

As a special case, a minimal cut vector to state j for a decreasing multi-state k-out-of-n:F system is:

$$(\underbrace{j-1, j-1, \ldots, j-1}_{k_j}, \underbrace{M, \ldots, M}_{n}). \quad (3.8)$$

Consider for example the following general multi-state k-out-of-n:F system:

$$n = 4, \quad M = 3, \quad k_1 = 2, \quad k_2 = 3, \quad k_3 = 1.$$

A minimal cut vector to system state 3 is (2, 3, 3, 3). A minimal cut vector to system state 2 is (1, 1, 1, 3), and it can be considered that k_3 is "absorbed" by k_2 because k_3 is less or equal to k_2. A minimal cut vector to system state 1 is (0, 0, 1, 3). All minimal cut vectors to a specific system state are all permutations of the elements of the corresponding minimal cut vector presented above.

3.1.4 Recursive algorithms for reliability evaluation of generalized multi-state k-out-of-n systems

3.1.4.1 Recursive algorithm for increasing multi-state k-out-of-n:F systems

First we consider the evaluation of increasing multi-state k-out-of-n:F system with $k_1 < k_2 < \cdots < k_M$. To evaluate the state distribution of an increasing multi-state k-out-of-n:F system, we need to calculate $Q_{s,1}, Q_{s,2}, \ldots, Q_{s,M}$. The probability that the system state is below j, i.e. $Q_{s,j}$, is equal to the probability that there exists a minimal cut vector of the system so that each component state is no more than the corresponding element of the minimal cut vector [46]. Based on the form of minimal cut vector for increasing multi-state k-out-of-n:F systems we presented in Section 3.1.3.2, to any state j, we have

$$Q_{s,j} =$$

$$\Pr(\mathbf{x} \leq (\underbrace{j-1, j-1, \ldots, j-1}_{k_j}, \underbrace{j, \ldots, j}_{k_{j+1}}, \ldots, \underbrace{M-1, \ldots, M-1}_{k_M}, \underbrace{M, \ldots, M}_{n})^{\star}) \tag{3.9}$$

In the following part of this section, we will present a recursive algorithm to calculate the probability on the right hand side of equation (3.9).

When $j = M$, we have

$$Q_{s,M} = \Pr(\mathbf{x} \leq (\underbrace{M-1, M-1, \ldots, M-1}_{k_M}, \underbrace{M, \ldots, M}_{n})^{\star}) \tag{3.10}$$

The following formula can be used to evaluate the probability expression in equation (3.10):

$$Q_{s,M} = \sum_{i_0=k_M}^{n} \binom{n}{i_0} Q_M^{i_0} p_M^{n-i_0} \tag{3.11}$$

The rationale behind this formula is straight forward. We calculate the probability when there are exactly i_0 components in states below M ($k_M \leq i_0 \leq M$), while taking the combination of the components into consideration, and add them together. We can also think in this way: there are only two possible states, state M (so-called state "1") and state below M (so-called state "0") , the probability of the components in these two possible states are p_M and Q_M respectively, and thus equation (3.11) can be interpreted as the probability of the two-state system in state "0".

When $j = M-1$, we have

$$Q_{s,M-1} =$$

$$\Pr(\mathbf{x} \leq (\underbrace{\underbrace{\underbrace{M-2, M-2, \ldots, M-2}_{k_{M-1}}, M-1, \ldots, M-1}_{k_M}, M, \ldots, M}_{n})^{\star}) \quad (3.12)$$

The following formula can be used to evaluate the probability expression in equation (3.12):

$$\begin{aligned} Q_{s,M-1} &= \sum_{i_1=k_{M-1}}^{k_M-1} \binom{n}{i_1} Q_{M-1}^{i_1} \left(\sum_{i_0=k_M-i_1}^{n-i_1} \binom{n-i_1}{i_0} p_{M-1}^{i_0} p_M^{n-i_1-i_0} \right) \\ &\quad + \sum_{i_0=k_M}^{n} \binom{n}{i_0} Q_{M-1}^{i_0} P_{M-1}^{n-i_0} \end{aligned} \quad (3.13)$$

where i_0 and i_1 are enumeration variables. We may think that there are totally three possible states, state M (so-called state "2"), state $M-1$ (so-called state "1") and state below $M-1$ (so-called state "0"). The probability of the components in these three possible states are p_M, p_{M-1} and Q_{M-1} respectively. Thus equation (3.13) can be interpreted as the probability of the three-state system in state "0".

The expression on the right hand side of equation (3.13) consists of two terms which are connected by the "+" sign. The first term represents the case when there are exactly i_1 ($k_{M-1} \leq i_1 \leq k_M - 1$) components in state "0" (states below $M-1$). The probability when there are exactly i_1 ($k_{M-1} \leq i_1 \leq k_M - 1$)

components in state"0" and the system state is "0" is

$$Q_{M-1}^{i_1}\left(\sum_{i_0=k_M-i_1}^{n-i_1}\binom{n-i_1}{i_0}p_{M-1}^{i_0}p_M^{n-i_1-i_0}\right),$$

as shown in equation (3.13). The part in the bracket is the probability that the other $n-i_1$ components are in state "1" (state $M-1$) or above and at least k_M-i_1 components are in states below "2" (state M). Therefore, the part in the bracket can be considered to be the probability that a nominal increasing multi-state k-out-of-n:F system (to be defined below) with two possible states ("0" and "1") is in state "0" .

Definition 3.8 *A nominal increasing multi-state k-out-of-n:F system is the same as an increasing multi-state k-out-of-n:F system except that the probability of a component in all possible states may be less than 1.*

We use the symbol " - " to indicate the parameters of this nominal increasing multi-state k-out-of-n:F system, and we have $\bar{n}=n-i_1$, $\bar{k}_1=k_M-i_1$, $\bar{p}_1=p_M$, and $\bar{p}_0=p_{M-1}$. That is, the part in the bracket is equal to the probability

$$\Pr(\bar{\boldsymbol{x}}\le(\underbrace{\underbrace{\text{"0"},\text{"0"},\ldots,\text{"0"}}_{\bar{k}_1},\text{"1"},\ldots,\text{"1"}}_{\bar{n}})^{\star}) \tag{3.14}$$

Equation (3.11) can be used to calculate this probability. Note that $\sum_i \bar{p}_i$ is not necessarily equal to 1 in the nominal increasing multi-state k-out-of-n:F system.

The second term of the expression on the right hand side of equation (3.13) also can be considered to be the probability that a nominal increasing multi-state k-out-of-n:F system with two possible states is in state "0". We use the

symbol " - " to indicate the parameters of this nominal increasing multi-state k-out-of-n:F system too, and we have $\bar{n} = n$, $\bar{k}_1 = k_M$, $\bar{p}_1 = p_M + p_{M-1}$, and $\bar{p}_0 = Q_{M-1}$.

In summary, the expression on the right hand side of equation (3.13) can be interpreted as the probability that a nominal increasing multi-state k-out-of-n:F system with three possible states is in state "0". This probability can further be calculated via the probabilities that some nominal increasing multi-state k-out-of-n:F systems with two possible states are in state "0".

To the original increasing multi-state k-out-of-n:F system, when j is equal to $M-2$ or below, we have similar observations as those of the cases when $j = M-1$ and $j = M$. Having these observations in mind, we provide a recursive algorithm to calculate the $Q_{s,j}$ value to any state j for equation (3.9).

The recursive function we are using in the proposed recursive algorithm is denoted by $Q(m, N, \boldsymbol{k}, \boldsymbol{p})$, where

m: the number of possible states of the nominal increasing multi-state k-out-of-n:F system minus 1,

N: the number of components of the nominal increasing multi-state k-out-of-n:F system,

$\boldsymbol{k}$: the $\boldsymbol{k}$ vector of the nominal increasing multi-state k-out-of-n:F system, $\boldsymbol{k} = (\bar{k}_1, \bar{k}_2, \ldots, \bar{k}_m)$,

$\boldsymbol{p}$: the probability vector of the nominal increasing multi-state k-out-of-n:F system, $\boldsymbol{p} = (\bar{p}_0, \bar{p}_1, \ldots, \bar{p}_m)$.

The recursive function $Q(m, N, \boldsymbol{k}, \boldsymbol{p})$ is designed to represent the probability for the system to be in state "0" of a nominal increasing multi-state k-out-of-n:F system with $m+1$ possible states, totally N components, vector $\boldsymbol{k}$, and

probability vector $\boldsymbol{p}$. Thus, to calculate the $Q_{s,j}$ value as shown in equation (3.9) of an increasing multi-state k-out-of-n:F system, we can first transform it into a nominal increasing multi-state k-out-of-n:F system to state j. In a general case, assume that the considered increasing multi-state k-out-of-n:F system has n components, $M+1$ possible states, k vector $(k_1,\ k_2,\ ,\ldots,k_M)$, and probability vector $(p_0,\ p_1,\ ,\ldots,p_M)$. To calculate $Q_{s,j}$ value to a certain state j, we will get a nominal increasing multi-state k-out-of-n:F system to state j, with $m = M-j+1$, $N = n$, $\boldsymbol{k} = (k_j,\ k_j+1,\ ,\ldots,k_M)$, and $\boldsymbol{p} = \left(\sum_{i=0}^{j-1} p_i,\ p_j,\ p_{j+1},\ldots,\ p_M\right)$. Using the recursive algorithm, the value $Q_{s,j}$ is equal to $Q(m,N,\boldsymbol{k},\boldsymbol{p})$.

Assuming the parameters of the nominal increasing multi-state k-out-of-n:F system to state j are m, N, $\boldsymbol{k}$ and $\boldsymbol{p}$, we have

$$Q_{s,j} = Q(m,N,\boldsymbol{k},\boldsymbol{p}) \tag{3.15}$$

The procedure of the recursive algorithms is as follows:

$$\begin{aligned} Q(m,N,\boldsymbol{k},\boldsymbol{p}) &= \sum_{i=\bar{k}_1}^{\bar{k}_2-1} \binom{N}{i} \bar{p}_0^i \cdot Q(m-1,N-i,\dot{\boldsymbol{k}},\dot{\boldsymbol{p}}) \\ &\quad + Q(m-1,N,\ddot{\boldsymbol{k}},\ddot{\boldsymbol{p}}) \end{aligned} \tag{3.16}$$

where

$\dot{\boldsymbol{k}} = \left(\bar{k}_2-i,\bar{k}_3-i,\ldots,\bar{k}_m-i\right)$, $\dot{\boldsymbol{p}} = (\bar{p}_1,\bar{p}_2,\ldots,\bar{p}_m)$,

$\ddot{\boldsymbol{k}} = \left(\bar{k}_2,\bar{k}_3,\ldots,\bar{k}_m\right)$, $\ddot{\boldsymbol{p}} = (\bar{p}_0,\bar{p}_1+\bar{p}_2,\bar{p}_3,\ldots,\bar{p}_m)$,

The boundary condition for the recursive algorithm is

$$Q(m,N,\boldsymbol{k},\boldsymbol{p}) = \sum_{i=\bar{k}_1}^{N} \binom{N}{i} \bar{p}_0^i \cdot \bar{p}_1^{N-i} \tag{3.17}$$

for $m = 1$.

From equations (3.16) and (3.17), we can see that this algorithm is actually recursive on the parameter m, not on the number of components N. Therefore, we can apply the recursive algorithm to large systems including a large number of components, without leading to exponential growth of computation time.

After calculating the $Q_{s,j}$ value of an increasing multi-state k-out-of-n:F system for all state j, we can use the following equations to get $r_{s,j}$, the probability of the system in state j

$$\begin{aligned} r_{s,0} &= Q_{s,1} \\ r_{s,M} &= 1 - Q_{s,M} \\ r_{s,j} &= Q_{s,j+1} - Q_{s,j} \quad (1 \leq j < M) \end{aligned} \tag{3.18}$$

3.1.4.2 Examples for the proposed recursive algorithm

We will use two examples to illustrate and verify the recursive algorithm for increasing multi-state k-out-of-n:F systems proposed in the previous section.

Example 3.2 *We consider an increasing multi-state k-out-of-n:F system with 10 i.i.d. components and 4 possible states, i.e.,* $n = 10$ *and* $M = 3$. *The* $\mathbf{k}$ *vector is* $\mathbf{k} = (k_1,\ k_2,\ k_3) = (3,\ 6,\ 8)$. *The state distribution of components is* $\mathbf{p} = (0.1,\ 0.3,\ 0.4,\ 0.2)$.

To calculate $Q_{s,3}$, *we get a nominal increasing multi-state k-out-of-n:F system to state 3, with* $m = 1$, $N = 10$, $\mathbf{k} = (k_3) = 8$, *and* $\mathbf{p} = (p_0 + p_1 + p_2,\ p_3) = (0.8,\ 0.2)$. *Using the recursive algorithm, the value* $Q_{s,3}$ *is equal to* $Q(m, N, \mathbf{k}, \mathbf{p})$, *which is 0.6778.*

To calculate $Q_{s,2}$, *we get a nominal increasing multi-state k-out-of-n:F sys-*

tem to state 2, with $m = 2$, $N = 10$, $\mathbf{k} = (k_2, k_3) = (6, 8)$, *and* $\mathbf{p} = (p_0 + p_1, p_2, p_3) = (0.4, 0.4, 0.2)$. *Using the recursive algorithm, the value* $Q_{s,2}$ *is equal to* $Q(m, N, \mathbf{k}, \mathbf{p})$, *which is 0.1523.*

To calculate $Q_{s,1}$, *we get a nominal increasing multi-state k-out-of-n:F system to state 1, with* $m = 3$, $N = 10$, $\mathbf{k} = (k_1, k_2, k_3) = (3, 6, 8)$, *and* $\mathbf{p} = (p_0, p_1, p_2, p_3) = (0.1, 0.3, 0.4, 0.2)$. *Using the recursive algorithm, the value* $Q_{s,1}$ *is equal to* $Q(m, N, \mathbf{k}, \mathbf{p})$, *which is 0.0308.*

Using equation (3.18), we can get the probability of the system at each individual state

$$r_{s,0} = 0.0308, \quad r_{s,1} = 0.1214, \quad r_{s,2} = 0.5255, \quad r_{s,0} = 0.3222$$

We also used the enumerating method to calculate the state distribution of this increasing multi-state k-out-of-n:F system. The results agree with those we get using the proposed recursive algorithm, which verifies the correctness of the recursive algorithm.

Example 3.3 *In this example, we will mainly investigate the efficiency as well as the correctness of the proposed recursive algorithm, by varying the number of possible states and number of components of the considered increasing multi-state k-out-of-n:F system.*

First, we consider an increasing multi-state k-out-of-n:F system with the following settings: $n = 10$, $M = 4$, *the* $\mathbf{k}$ *vector is* $\mathbf{k} = (1, 2, 4, 5)$, *and the state distribution of components is* $\mathbf{p} = (0.1, 0.2, 0.1, 0.4, 0.2)$. *The results obtained using the proposed recursive algorithm are*

$\mathbf{Q} = (0.4587, 0.6009, 0.6174, 0.9936)$

$\mathbf{r} = (0.4587, 0.1422, 0.0164, 0.3763, 0.0064)$

The enumerating method is also used to calculate the state distribution of this increasing multi-state k-out-of-n:F system. The results agree with those by

Table 3.1: Computation time with respect to different number of components

n	5	8	10	15	20	30	50	100
Recursive Algorithm	0.01	0.02	0.03	0.04	0.06	0.08	0.14	0.32
Enumerating Method	0.04	3.38	80.45	-	-	-	-	-

the recursive algorithm, which further illustrate the correctness of the recursive algorithm.

Varying the number of components n while keeping other parameters the same, we can investigate the efficiency of the recursive algorithm with respect to parameter n. The computation time (seconds) with respect to different n values are shown in Table 3.1. We can see that with the increase of the number of components, the computation time increases, but it does not increase exponentially. We also use the enumerating method to evaluate the increasing multi-state k-out-of-n:F systems with varied number of components n, and listed the computation time (seconds) in Table 3.1 as well. With respect to each number of components n, the enumerating method needs more computation time than the recursive algorithm. With the increase of the number of components, the computation time of the enumerating method increases dramatically. We do not list the computation time of the enumerating method for n greater than 10 because the computation time for those cases is too long to endure. This example has illustrated that the recursive algorithm is far more efficient than the enumerating method, especially to systems with a large number of components.

3.1.4.3 Evaluating general multi-state k-out-of-n:F systems using the recursive algorithm

We have presented the form of minimal cut vector of general multi-state k-out-of-n:F systems in Section 3.1.3.3. Assuming a minimal cut vector to state j is represented by $\boldsymbol{v}_j$, we can calculate the probability that the system is in state below j, $Q_{s,j}$, by

$$Q_{s,j} = \Pr(\mathbf{x} \leq (\boldsymbol{v}_j)^{\star}) \tag{3.19}$$

where the symbol $\star$ is used as a superscript to represent all permutations of the elements of this minimal cut vector.

The following procedure is proposed to calculate $Q_{s,j}$ value to any state j of a general multi-state k-out-of-n:F system:

(1). Find the minimal cut vector to state j of this system using the method presented in Section 3.1.3.3.

(2). Generate a nominal increasing multi-state k-out-of-n:F system to state j, and calculate the parameters m, N, $\boldsymbol{k}$ and $\boldsymbol{p}$. For instance, assume that the minimal cut vector to state j of the general multi-state k-out-of-n:F system has the form shown in equation (3.7). Thus the nominal system has 4 possible states, i.e., $m = 3$. The number of components of the nominal system is the same as the general multi-state k-out-of-n:F system, i.e., $N = n$. The $\boldsymbol{k}$ vector is $\boldsymbol{k} = (k_j, k_a, k_b)$. The probability vector $\boldsymbol{p}$ is $\boldsymbol{p} = \left(\sum_{i=0}^{j-1} p_i, \sum_{i=j}^{a-1} p_i, \sum_{i=a}^{b-1} p_i, \sum_{i=b}^{M} p_i\right)$

(3). Using the recursive algorithm to calculate the $Q_{s,j}$ value: $Q_{s,j} = Q(m, N, \boldsymbol{k}, \boldsymbol{p})$.

Example 3.4 *This example is used to illustrate the procedure of evaluating*

a general multi-state k-out-of-n:F system. The system we investigate has the following settings: $n = 4$, $M = 4$, *the* $\mathbf{k}$ *vector is* $\mathbf{k} = (2, 3, 3, 1)$ *which is not in monotonic ordering, and the state distribution of components is* $\mathbf{p} = (0.1, 0.2, 0.1, 0.4, 0.2)$.

To state $j = 4$, *a minimal cut vector has the form of* $(3, 4, 4, 4)$. *To calculate* $Q_{s,4}$, *we get a nominal increasing multi-state k-out-of-n:F system to state 4, with* $m = 1$, $N = 4$, $\mathbf{k} = 1$, *and* $\mathbf{p} = (0.8, 0.2)$. *Using the recursive algorithm, we get* $Q_{s,4} = Q(m, N, \mathbf{k}, \mathbf{p}) = 0.9984$.

To state $j = 3$, *a minimal cut vector has the form of* $(2, 2, 2, 4)$. *To calculate* $Q_{s,3}$, *we get a nominal increasing multi-state k-out-of-n:F system to state 3, with* $m = 1$, $N = 4$, $\mathbf{k} = 3$, *and* $\mathbf{p} = (0.4, 0.6)$. *Using the recursive algorithm, we get* $Q_{s,3} = Q(m, N, \mathbf{k}, \mathbf{p}) = 0.1792$.

To state $j = 2$, *a minimal cut vector has the form of* $(1, 1, 1, 4)$. *To calculate* $Q_{s,2}$, *we get a nominal increasing multi-state k-out-of-n:F system to state 2, with* $m = 1$, $N = 4$, $\mathbf{k} = 3$, *and* $\mathbf{p} = (0.3, 0.7)$. *Using the recursive algorithm, we get* $Q_{s,2} = Q(m, N, \mathbf{k}, \mathbf{p}) = 0.0837$.

To state $j = 1$, *a minimal cut vector has the form of* $(0, 0, 1, 4)$. *To calculate* $Q_{s,1}$, *we get a nominal increasing multi-state k-out-of-n:F system to state 1, with* $m = 2$, $N = 4$, $\mathbf{k} = (2, 3)$, *and* $\mathbf{p} = (0.1, 0.2, 0.7)$. *Using the recursive algorithm, we get* $Q_{s,1} = Q(m, N, \mathbf{k}, \mathbf{p}) = 0.0229$.

Using equation (3.18), we can get the probability of the system at each individual state

$$\mathbf{r}_s = (0.0229,\ 0.0608,\ 0.0955,\ 0.8192,\ 0.0016)$$

We also used the enumerating method to calculate the state distribution of this general multi-state k-out-of-n:F system. The results agree with those by the proposed recursive algorithm, which verifies the correctness of the recursive

algorithm to general multi-state k-out-of-n:F systems.

Note that the approach proposed in this section is for "general" multi-state k-out-of-n:F systems. That is, this approach can be applied to any special cases such as increasing multi-state k-out-of-n:F systems, decreasing multi-state k-out-of-n:F systems and constant multi-state k-out-of-n:F systems. Especially, when a system is decreasing or constant multi-state k-out-of-n:F system, to any state j, it has minimal vectors in the form shown in equation (3.8), and $Q_{s,j}$ can be evaluated simply by using equation (3.11).

3.1.4.4 Evaluating generalized multi-state k-out-of-n:G systems using the recursive algorithm

As we mentioned in Section 3.1.3.1, there is equivalent relationship between generalized multi-state k-out-of-n:G and F systems. Thus the following procedure can be used to evaluate a generalized multi-state k-out-of-n:G system:

1. Generate the equivalent generalized multi-state k-out-of-n:F system of the considered generalized multi-state k-out-of-n:G by letting $k_j^F = n - k_j^G + 1$ for $j = 1, 2, \ldots, M$.

2. Using the recursive approach presented in Section 3.1.4.3 to evaluate the generated generalized multi-state k-out-of-n:F system, the state distributions of which are the same as the considered generalized multi-state k-out-of-n:G system.

Example 3.5 *This example is used to verify the correctness of the proposed recursive algorithm to evaluate generalized multi-state k-out-of-n:G systems. The generalized multi-state k-out-of-n:G system we investigate is taken from Example 7 of Huang* et al. *[37]. It has totally 4 components and 5 possible*

states, i.e., $n = 4$ and $M = 4$. the $\mathbf{k}^G$ vector is $\mathbf{k}^G = (4,\ 3,\ 2,\ 1)$, and the state distribution of components is $\mathbf{p} = (0.1,\ 0.2,\ 0.3,\ 0.3,\ 0.1)$.

The equivalent generalized multi-state k-out-of-n:F system has the $\mathbf{k}$ *vector* $\mathbf{k} = n - \mathbf{k}^G + 1 = (1,\ 2,\ 3,\ 4)$.

while other parameters are kept the same.

Using the recursive approach presented in Section 3.1.4.3 to evaluate the equivalent generalized multi-state k-out-of-n:F system, we get the results as follows

$\mathbf{Q}_s = (0.1331,\ 0.2187,\ 0.3888,\ 0.6561)$

$\mathbf{r}_s = (0.1331,\ 0.0856,\ 0.1701,\ 0.2673,\ 0.3439)$

The results agree with those in Huang et al. *[37] and those we get using enumerating method, which verifies the correctness of the proposed recursive algorithm to evaluate generalized multi-state k-out-of-n:G systems.*

3.2 Reliability bounds for generalized multi-state k-out-of-n systems

In this section, we present a reliability bounding approach for generalized multi-state k-out-of-n systems with i.i.d. components. The materials in this section have been published in [90].

Exact performance evaluation algorithms have been available for multi-state k-out-of-n systems with identical and independent (i.i.d.) components as discussed in Section 3.1, and multi-state k-out-of-n systems with independent components [93]. These algorithms are much more efficient than enumeration methods. However, for complex systems with a large number of components and a large number of possible states, the calculation of system state distribution will still require a significant amount of time. In practical situations,

sometimes we do not have to obtain the exact system state distribution. We would rather get a good enough range of the system reliability in a much shorter computation time, which will allow us to make decisions more efficiently. This is why "reliability bounds" is an interesting and significant issue.

The issue of reliability bounding has been extensively studied, both in the binary context [46] and multi-state context [46, 36, 35, 41, 60]. Several binary reliability bounding approaches are generalized to multi-state systems by Block and Savits [7], and analyzed by Meng [60]. These approaches are some simple formulas generalized from the binary case. Hudson and Kapur [41] developed bounding approaches for multi-state systems using Inclusion-Exclusion (IE) method and Sum of Disjoint Product (SDP) methods, assuming that the minimal cut vectors or minimal path vectors are given. Huang *et al.* developed bounding approaches for generalized multi-state k-out-of-n:G systems [35] and consective multi-state k-out-of-n systems [36], by simplifying the minimal path or cut vectors to include no more than two different states. The limitation of their approaches are apparent, that is, we can not include more than two different states to seek better bounds.

In general, a systematic and flexible approach is still not available to obtain reliability bounds for multi-state k-out-of-n systems with i.i.d. components. As mentioned, an efficient recursive algorithm has been available for the exact performance evaluation of multi-state k-out-of-n systems [94, 105]. In this section, we will propose a systematic and flexible reliability bounding approach based on the recursive algorithm. Using the bounding approach, we can obtain a good estimate of the exact system reliability value while significantly reducing the computation time. This approach is attractive especially to complex systems with a large number of components and a large number of possible

states. A numerical example will be used to illustrate the significance of the proposed bounding approach.

In addition to the notations listed in Section 3.1, the following additional notations are given.

Notation:

- $Q(\bullet)$: the recursive function used by the proposed recursive algorithm, $Q = Q(m, N, \boldsymbol{k}, \boldsymbol{p})$
- $Q_{\boldsymbol{k}}(\bullet)$: the $Q_{s,d}$ value when vector $\boldsymbol{k}$ has a specific value
- $\mathtt{Pr}_{\mathtt{ub}}$: the obtained upper bound
- $\mathtt{Pr}_{\mathtt{lb}}$: the obtained lower bound
- $t_{\mathtt{ub}}$: the computation time for calculating the upper bounds
- $t_{\mathtt{lb}}$: the computation time for calculating the lower bounds

3.2.1 The proposed reliability bounding approach

The systems under consideration are multi-state k-out-of-n systems with i.i.d. components. We will focus on the probability of the system in states below a certain state d, that is

$$Q_{s,d} = \Pr(\phi(\mathbf{x}) < d) \tag{3.20}$$

The probability of the system in state d or above, represented by R_{sd}, is equal to $1 - Q_{s,d}$. Based on the recursive algorithm for performance evaluation of multi-state k-out-of-n systems with i.i.d. components [105], a reliability bounding approach is proposed in this section.

As mentioned in Section 3.1, the calculation of $Q_{s,d}$ can be transformed into the probability for the system to be in state "0" of a generated nominal increasing multi-state k-out-of-n:F system with $m+1$ possible states, totally N components, vector $\boldsymbol{k}$, and probability vector $\boldsymbol{p}$. Furthermore, the algorithm in [105] is actually recursive on the parameter m, not on the number of components N. The algorithm can be applied to large systems including a large number of components without leading to exponential growth of computation time. However, when the number of possible nominal states m increases, the computation time will increase much faster than the case when N increases. Thus, a bounding approach that requires a shorter vector $\boldsymbol{k}$ and therefore a much shorter computation time is promising.

$Q_{s,d}$ is used to represent the exact probability value that the system is in states below nominal state d. We use $Q_{\boldsymbol{k}}(k_1,\ k_2,\ ...,\ k_m)$ to represent the $Q_{s,d}$ value when $\boldsymbol{k} = (k_1,\ k_2,\ ...,\ k_m)$. We have the following property:

Property: For any nominal state j $(1 \leq j \leq m)$ of a nominal increasing multi-state k-out-of-n:F system, we have

$$Q_{\boldsymbol{k}}(k_1,\ ...,\ k_j,\ ...,\ k_m) \leq Q_{\boldsymbol{k}}(k_1,\ ...,\ k_j - 1,\ ...,\ k_m) \tag{3.21}$$

The reason is whenever there are k_j components in states below j, there will be always $k_j - 1$ components in states below j. This property provides us a basis to generate bounds for $Q_{s,d}$.

Let's consider a specific example first. Suppose that the generated $\boldsymbol{k}$ vector is (1, 2, 3, 4), which includes 4 elements in strictly increasing order. Based on formula (3.21), if $\boldsymbol{k}$ vector is equal to (1, 1, 1, 1), we will have a bigger $Q_{s,d}$ value, since the requirement on nominal state 2, 3 and 4 become less strict.

Thus we have an upper bound for $Q_{s,d}$. Based on similar analysis, we can find $Q_{\boldsymbol{k}}(4,\ 4,\ 4,\ 4)$ is smaller than $Q_{s,d}$. Thus, we have a pair of upper and lower bounds for $Q_{s,d}$

$$Q_{\boldsymbol{k}}(4,\ 4,\ 4,\ 4) \leq Q_{s,d} = Q_{\boldsymbol{k}}(1,\ 2,\ 3,\ 4) \leq Q_{\boldsymbol{k}}(1,\ 1,\ 1,\ 1) \tag{3.22}$$

The $Q_{s,d}$ value of a general multi-state k-out-of-n:F system, where the $\boldsymbol{k}$ vector is not necessarily in a strictly increasing order, can be calculated by transforming the system into a nominal increasing multi-state k-out-of-n:F system [105]. Therefore, $Q_{\boldsymbol{k}}(1,\ 1,\ 1,\ 1)$ can be simplified to include only one element in the $\boldsymbol{k}$ vector, that is, $Q_{\boldsymbol{k}}(1,\ 1,\ 1,\ 1) = Q_{\boldsymbol{k}}(1)$. Note that in the case of $Q_{\boldsymbol{k}}(1)$, there are only two nominal states, state 0 and 1. Specifically, state 1, 2, 3, 4 of the original system are combined into one nominal state 1. Thus, the upper and lower bounds in (3.22) can be written as

$$Q_{\boldsymbol{k}}(4) \leq Q_{s,d} = Q_{\boldsymbol{k}}(1,\ 2,\ 3,\ 4) \leq Q_{\boldsymbol{k}}(1) \tag{3.23}$$

In the bounds in (3.23), we include only one element in the $\boldsymbol{k}$ vectors. We will have tighter bounds if we include more elements in them. Specifically, in the case of lower bounds, we have

$$\begin{aligned} Q_{\boldsymbol{k}}(4,\ 4,\ 4,\ 4) &\leq Q_{\boldsymbol{k}}(1,\ 4,\ 4,\ 4) \leq Q_{\boldsymbol{k}}(1,\ 2,\ 4,\ 4) \\ &\leq Q_{\boldsymbol{k}}(1,\ 2,\ 3,\ 4) = Q_{s,d} \end{aligned} \tag{3.24}$$

And in the case of upper bounds, we have

$$Q_{s,d} = Q_{\boldsymbol{k}}(1,\ 2,\ 3,\ 4) \leq Q_{\boldsymbol{k}}(1,\ 2,\ 3,\ 3) \leq Q_{\boldsymbol{k}}(1,\ 2,\ 2,\ 2)$$

$$\leq \; Q_{\boldsymbol{k}}\,(1,\ 1,\ 1,\ 1) \tag{3.25}$$

If we write them in a simplified way as in Equation (3.23), we have

$$Q_{\boldsymbol{k}}\,(4) \leq Q_{\boldsymbol{k}}\,(1,\ 4) \leq Q_{\boldsymbol{k}}\,(1,\ 2,\ 4) \leq Q_{\boldsymbol{k}}\,(1,\ 2,\ 3,\ 4) = Q_{s,d} \tag{3.26}$$

and

$$Q_{s,d} = Q_{\boldsymbol{k}}\,(1,\ 2,\ 3,\ 4) \leq Q_{\boldsymbol{k}}\,(1,\ 2,\ 3) \leq Q_{\boldsymbol{k}}\,(1,\ 2) \leq Q_{\boldsymbol{k}}\,(1) \tag{3.27}$$

All the $Q_{\boldsymbol{k}}\,(\bullet)$ values in (3.26) and (3.27) can be calculated using the efficient recursive algorithm presented in [105].

Now we consider the general case $Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ ...,\ k_m)$, where there are m strictly increasing elements in the $\boldsymbol{k}$ vector. Based on the property in Equation (3.21) and the analysis on the specific example above, we have the series of lower bounds for $Q_{s,d}$ as follows (showing all the m elements)

$$\begin{aligned} &Q_{\boldsymbol{k}}\,(k_m,\ k_m,\ \ldots,\ k_m) \leq Q_{\boldsymbol{k}}\,(k_1,\ k_m,\ldots,\ k_m) \leq Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ k_m,\ldots,\ k_m) \leq \ldots \\ &\leq Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ \ldots,\ k_{m-2},\ k_m,\ k_m) \leq Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ \ldots,\ k_{m-1},\ k_m) = Q_{s,d}, \end{aligned} \tag{3.28}$$

or in the simplified form

$$\begin{aligned} Q_{\boldsymbol{k}}\,(k_m) \;&\leq\; Q_{\boldsymbol{k}}\,(k_1,\ k_m) \leq \ldots \leq Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ \ldots,\ k_{m-2},\ k_m) \\ &\leq\; Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ \ldots,\ k_{m-1},\ k_m) = Q_{s,d} \end{aligned} \tag{3.29}$$

And the series of upper bounds for $Q_{s,d}$ are (showing all the m elements)

$$Q_{s,d} \;=\; Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ \ldots,\ k_{m-1},\ k_m) \leq Q_{\boldsymbol{k}}\,(k_1,\ k_2,\ \ldots,\ k_{m-1},\ k_{m-1}) \leq \ldots$$

$$\leq Q_{\boldsymbol{k}}(k_1, k_2, k_2, \ldots, k_2) \leq Q_{\boldsymbol{k}}(k_1, k_1, \ldots, k_1) \tag{3.30}$$

or in the simplified form

$$\begin{aligned} Q_{s,d} &= Q_{\boldsymbol{k}}(k_1, \ldots, k_m) \leq Q_{\boldsymbol{k}}(k_1, \ldots, k_{m-1}) \leq \ldots \\ &\leq Q_{\boldsymbol{k}}(k_1, k_2) \leq Q_{\boldsymbol{k}}(k_1) \end{aligned} \tag{3.31}$$

In the general case, all the $Q_{\boldsymbol{k}}(\bullet)$ values in (3.29) and (3.31) can be calculated using the efficient recursive algorithm presented in [105]. When calculating the bounds, the more elements we include in the $\boldsymbol{k}$ vector, the tighter the bounds will be, and the longer the computation time will be.

There is actually another way, other than that in Equations (3.28) and (3.29), to calculate the lower bounds:

$$\begin{aligned} Q_{\boldsymbol{k}}(k_m, k_m, \ldots, k_m) &\leq Q_{\boldsymbol{k}}(k_{m-1}, \ldots, k_{m-1}, k_m) \\ &\leq Q_{\boldsymbol{k}}(k_{m-2}, \ldots, k_{m-2}, k_{m-1}, k_m) \\ &\leq \ldots \leq Q_{\boldsymbol{k}}(k_2, k_2, \ldots, k_{m-1}, k_m) \\ &\leq Q_{\boldsymbol{k}}(k_1, k_2, \ldots, k_{m-1}, k_m) = Q_{s,d}. \end{aligned} \tag{3.32}$$

However, the lower bounds obtained using Equation (3.32) are not as good as those obtained using Equations (3.28) and (3.29). This will be illustrated using an example later in Section 3.2.2.

Any multi-state k-out-of-n:G system has an equivalent multi-state k-out-of-n:F system, as pointed out in Chapter 3, therefore, we will focus only on multi-state k-out-of-n:F systems. Given a general multi-state k-out-of-n:F system, the procedure to calculate $Q_{s,d}$ with respect to a certain state d is as

follows:

1. Generate the nominal increasing multi-state k-out-of-n:F system with the $\boldsymbol{k}$ vector including m elements in a strictly increasing order.

2. Based on how complex the system is, that is, how many components there are and how large m is, we decide how many elements we want to include in the $\boldsymbol{k}$ vector when calculating the reliability bounds. A simple and effective way is starting from including only one elements, investigating the obtained bounds, and increasing the number of elements included if necessary. We can certainly include different number of elements when calculating the upper bound and lower bound. From our numerical experiments on the bounding approach, the upper bound is usually better than the lower bound when including the same number of elements in the $\boldsymbol{k}$ vector. Therefore, it is recommended to include more elements in the $\boldsymbol{k}$ vector when calculating the lower bound.

3. Based on how good the obtained bounds are and how efficient the calculation is, we have the flexibility to choose to include different number of elements in the $\boldsymbol{k}$ vector and investigate more options.

3.2.2 Examples

In this section, we will use a numerical example to investigate the accuracy and efficiency of the proposed reliability bounding approach. As mentioned in Section 3.2.1, $Q_{s,d}$, the probability of a multi-state k-out-of-n system in states below d, can be calculated through the probability of a generated nominal increasing multi-state k-out-of-n:F system in nominal state "0". The time for generating the nominal increasing multi-state k-out-of-n:F system is negligible.

In this example, we will use the generated nominal increasing multi-state k-out-of-n:F system directly to investigate the proposed bounding approach. Thus, d is specified to be 1, and $Q_{s,d}$ represents the probability of the system in state 0.

The nominal increasing multi-state k-out-of-n:F system used in this example has 100 i.i.d. components, and 8 possible states from state 0 to state 7. Thus we have $n = 100$ and $M = 7$. The $\boldsymbol{k}$ vector is specified to be

$$\boldsymbol{k} = (10,\ 15,\ 20,\ 25,\ 30,\ 35,\ 40) \tag{3.33}$$

For convenience, we set the probabilities of a component in different states to be identical, that is, the state distribution vector is $\boldsymbol{p}$ is

$$\boldsymbol{p} = (0.125,\ 0.125,\ 0.125,\ 0.125,\ 0.125,\ 0.125,\ 0.125,\ 0.125) \tag{3.34}$$

Actually, the value of state distribution vector $\boldsymbol{p}$ will not influence the computation time of the bounding approach. The factors that influence the computation time of the bounding approach are the value of n, M and vector $\boldsymbol{k}$.

To calculate the exact $Q_{s,d}$ value, we need to include the total 7 elements in Equation (3.33) into the $\boldsymbol{k}$ vector. Using the proposed bounding approach in Section 3.2.1, we can get a series of upper bounds and lowers bounds by including different number of elements into the $\boldsymbol{k}$ vector. The upper bounds and lower bounds are calculated using Equation (3.31) and (3.29) respectively. The programs for this example are developed with MATLAB 6.5, and implemented on a computer with Pentium M 1.7GHz CPU and 512 RAM. The results by the bounding approach are shown in Table 3.2, where m represents the number of

Table 3.2: Reliability bounding results

m	$\mathtt{Pr_{lb}}$	$\mathtt{Pr_{ub}}$	$t_{\mathtt{lb}}$	$t_{\mathtt{ub}}$
1	4.32E-12	0.81630461	0.03	0.03
2	6.86E-04	0.81595876	0.27	0.13
3	0.31592575	0.81595735	0.91	0.40
4	0.80768654	0.81595735	3.98	1.51
5	0.81595701	0.81595735	17.24	7.65
6	0.81595735	0.81595735	71.32	41.60
7	0.81595735	0.81595735	233.63	233.63

elements included in the $\boldsymbol{k}$ vector when calculating the bounds, $\mathtt{Pr_{lb}}$ and $\mathtt{Pr_{ub}}$ represent the obtained lower bound and upper bound, and $t_{\mathtt{lb}}$ and $t_{\mathtt{ub}}$ are the computation time for calculating the bounds, respectively. When $m = M = 7$, both $\mathtt{Pr_{ub}}$ and $\mathtt{Pr_{lb}}$ are equal to the exact $Q_{s,d}$ value, 0.81595735.

From Table 3.2, we will have more accurate upper and lower bounds with the increase of m. It can be found that the obtained upper bounds are close to the exact $Q_{s,d}$ value even when $m = 1$. From $m = 4$, the upper bounds are the same in Table 3.2 as the exact $Q_{s,d}$ value, up to 8 decimal places. On the other hand, the lower bounds are not so good when m is small, but they are close to the exact $Q_{s,d}$ value as well when m is 4 or larger. We also investigated other increasing multi-state k-out-of-n:F systems by varying the value of M, n, vector $\boldsymbol{k}$ and vector $\boldsymbol{p}$. It turns out that we can always get good upper bounds which are close to the exact $Q_{s,d}$ value even when m is relatively small. The obtained lower bounds show similar accuracy performance as those of the system in Table 3.2.

The computation times $t_{\mathtt{lb}}$ and $t_{\mathtt{ub}}$ in Table 3.2 increase greatly with the increase of m. Therefore, considering the accuracy of the bounds we mentioned in the previous paragraph, it would be a good idea to use appropriate upper

and lower bounds if we do not have to find the exact value. For example, we can use the upper bound when $m = 2$, which is 0.81595876, and the lower bound when $m = 4$, which is 0.80768654. This whole range will be only about 1 percent of the exact $Q_{s,d}$ value, 0.81595735, and the total computation time is only 4.11 seconds, about 1.8 percent of that for calculating the exact value. These bounds will give us a good idea of the actual $Q_{s,d}$ value in a much shorter time.

We also investigated the efficiency of the bounding approach when the number of the components n increases. For instance, when calculating the upper bound with $m = 4$, if we increase n from 100 to 200 while keeping other settings the same, the computation time will increase from 1.51 to 3.24. This confirms that the recursive algorithm in Section 3.1 used in the bounding approach is efficient versus the number of components n. Thus, the proposed bounding approach can be used as an efficient performance evaluation approach for multi-state k-out-of-n systems with a large number of components and possible states.

As mentioned in Section 3.2.1, there is another method for calculating the lower bounds, using Equation (3.32). The performances of the two methods are compared. The $\boldsymbol{k}$ vectors (before being simplified) used in the two methods are listed in Table 3.3, where "Method 1" refers to the method using Equations (3.28) and (3.29), and "Method 2" refers to the method using Equation (3.32). The results are shown in Table 3.4. When the number of elements m included in the $\boldsymbol{k}$ vector is 1, the two methods give the same lower bound value, this value is very close to 0 and thus, is not useful for system reliability approximation at all. As the m value increases, the lower bounds provided by method 1 grows much faster than those provided by method 2. This shows

Table 3.3: The $\boldsymbol{k}$ vectors for calculating the lower bounds with two different methods

m	Method 1	Method 2
1	(40, 40, 40, 40, 40, 40, 40)	(40, 40, 40, 40, 40, 40, 40)
2	(10, 40, 40, 40, 40, 40, 40)	(35, 35, 35, 35, 35, 35, 40)
3	(10, 15, 40, 40, 40, 40, 40)	(30, 30, 30, 30, 30, 35, 40)
4	(10, 15, 20, 40, 40, 40, 40)	(25, 25, 25, 25, 30, 35, 40)
5	(10, 15, 20, 25, 40, 40, 40)	(20, 20, 20, 25, 30, 35, 40)
6	(10, 15, 20, 25, 30, 40, 40)	(15, 15, 20, 25, 30, 35, 40)
7	(10, 15, 20, 25, 30, 35, 40)	(10, 15, 20, 25, 30, 35, 40)

Table 3.4: The lower bounds results with two different methods

m	$\mathtt{Pr}_{\mathtt{lb}}$ (Method 1)	$\mathtt{Pr}_{\mathtt{lb}}$ (Method 2)	$t_{\mathtt{lb}}$ (Method 1)	$t_{\mathtt{lb}}$ (Method 2)
1	4.32E-12	4.32E-12	0.03	0.03
2	6.86E-04	6.16E-09	0.27	0.09
3	0.31592575	3.03E-06	0.91	0.35
4	0.80768654	4.78E-04	3.98	1.23
5	0.81595701	0.02203005	17.24	6.61
6	0.81595735	0.26478972	71.32	38.56
7	0.81595735	0.81595735	233.63	233.63

that method 1 provides much tighter lower bounds for system reliability evaluation. Of course, when $m = M = 7$, all the elements in the $\boldsymbol{k}$ vector are used, both methods give the exact system reliability value.

3.2.3 Conclusions

A reliability bounding approach is proposed in this section based on the recursive algorithm for performance evaluation of multi-state k-out-of-n systems with i.i.d. components. The upper and lower bounds of $Q_{s,d}$ are calculated by reducing the length of the $\boldsymbol{k}$ vector when using the recursive algorithm presented in Section 3.1. Usually we can get better upper bounds than lower

bounds when including the same number of elements in the $\boldsymbol{k}$ vectors. Using the bounding approach, we can obtain a good estimate of the exact $Q_{s,d}$ value while significantly reducing the computation time. Generally speaking, the proposed bounding approach can be used as an efficient performance evaluation approach to multi-state k-out-of-n systems with a large number of components and possible states.

The contributions of the proposed reliability bounding approach are: (1) An approach to obtain reliability bounds for multi-state k-out-of-n systems is proposed. Specifically, the upper and lower bounds of $Q_{s,d}$ are calculated by reducing the length of the $\boldsymbol{k}$ vector when using the recursive algorithm presented in Section 3.1. (2) The bounding approach provides a fast reliability evaluation way, attractive especially to complex systems with a large number of components and a large number of possible states. (3) By controlling the length of the $\boldsymbol{k}$ vector used in the proposed bounding approach, we can obtain reliability bounds with different levels of accuracy.

3.3 Concluding Remarks

In this chapter, efficient methods have been developed for the reliability evaluation of generalized multi-state k-out-of-n systems defined by Huang et al [37]. We focus on generalized multi-state k-out-of-n systems with i.i.d. components. A method for the exact reliability evaluation of generalized multi-state k-out-of-n systems is presented in Section 3.1. And a reliability bounding approach is developed as well for fast system reliability evaluation in Section 3.2.

Chapter 4

Another Multi-State k-out-of-n System Model and Its Evaluation

4.1 Introduction

In Chapter 3, we present methods to evaluate the reliability of generalized multi-state k-out-of-n systems defined by Huang *et al.* [37, 105]. It is a natural extension from the binary k-out-of-n system model that we allow different k values with respect to different states. However, Huang's model of multi-state k-out-of-n systems [37] suffers from the fact that few practical applications can fit into this model. In Section 2.2.2, we described a power station with three generators as an example of decreasing multi-state k-out-of-n:G system model. The limitations are that there can only be three possible states and thus two k values, k_1 and k_2, and k_2 can only be equal to 1. There might be other applications of Huang's model yet to be identified.

In this chapter, we attempt to develop a new multi-state k-out-of-n system model which allows different k values with respect to different states as well, and, very importantly, more practical engineering systems can fit into this model. Two categories of applications have been identified for this model. In

the first category, multiple states are interpreted as multiple levels of capacity. As an example, let's consider an oil transmission pipeline. The pipeline is used to transmit oil from the oil source to spot A, B and C aligned in order along the pipeline. We say that the pipeline is in state 0 if it can not transmit oil to any of the three spots; it is in state 1 if the oil can reach spot A; it is in state 2 if the oil can reach up to spot B, i.e., spots A and B; it is in state 3 if the oil can reach up to spot C. Thus, an oil transmission system, which will be discussed in details later in this chapter, has different requirements on the number of components on different levels. In the second category, multiple states are interpreted in terms of multiple failure modes. The working state of a component has positive cumulative contributions to the system, some failure states of a component have no contributions whatsoever to the system, while other failure states of a component have negative cumulative contributions to the system. The new multi-state k-out-of-n system model and the detailed descriptions of these two categories of applications will be presented later in this chapter.

It is critical to find efficient reliability evaluation algorithms for multi-state k-out-of-n systems. In Section 4.3, we will propose an approach for reliability evaluation of multi-state k-out-of-n systems with i.i.d. components. This approach is very similar to that for generalized multi-state k-out-of-n:F system evaluation, as discussed in Section 3.1, except that it uses minimal path vectors while the algorithm in Section 3.1 uses minimal cut vectors, based on the differences between the multi-state k-out-of-n models they deal with. In Section 4.4, we will propose a recursive algorithm for reliability evaluation of multi-state k-out-of-n systems with independent components. This algorithm, however, can be considered to be an extension from the binary k-out-of-n

system reliability evaluation algorithms [4, 82]. Efficiencies of the proposed algorithms will be investigated as well in the following sections of this chapter.

The materials in this chapter has been documented in paper [95].

Assumptions:

- The state space of each component and the system is $\{0, 1, 2, \ldots, M\}$.
- The state of the system is completely determined by the states of the components.

Notation:

- n: The number of components of a system
- M: The maximum state level of a multi-state system and its components
- x_i: state of component i, $x_i = j$ if component i is in state j, $0 \leq j \leq M$, $1 \leq i \leq n$
- $\mathbf{x}$: an n-dimensional vector representing the states of all components, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$
- $\phi(\mathbf{x})$: state of the system, $0 \leq \phi(\mathbf{x}) \leq M$
- k_j: the k value with respect to level j of a generalized multi-state k-out-of-n system
- $\boldsymbol{v}_j$: a minimal cut vector to level j of an increasing multi-state k-out-of-n:F system
- $P_{s,j}$: $\Pr(\phi(\mathbf{x}) \geq j)$

- $Q_{s,j}$: $\Pr(\phi(\mathbf{x}) < j)$, i.e., $1 - P_{s,j}$
- $r_{s,j}$: $\Pr(\phi(\mathbf{x}) = j)$
- $P(\bullet)$: the recursive function, $P = P(n, \boldsymbol{k}, \boldsymbol{P})$
- $\boldsymbol{k}$: the $\boldsymbol{k}$ vector of a multi-state k-out-of-n system, $\boldsymbol{k} = (k_1, k_2, \ldots, k_M)$
- $\boldsymbol{P}$: the component state distribution matrix for a nominal multi-state k-out-of-n system
- $p_{n,j}$: the probability of component n in state j
- $\boldsymbol{k}^j$: the generated $\boldsymbol{k}$ vector when component n is in state j
- $\boldsymbol{P}^j$: the generated $\boldsymbol{P}$ matrix when component n is in state j

4.2 The multi-state k-out-of-n system model and its applications

4.2.1 Definition of the multi-state k-out-of-n system model

We define a new multi-state k-out-of-n:G system model as follows:

Definition 4.9 *An n-component system is called a multi-state k-out-of-n:G system if* $\phi(\mathbf{x}) \geq j$ $(1 \leq j \leq M)$ *whenever at least* k_l *components are in state* l *or above for all* l *such that* $1 \leq l \leq j$.

Intuitively, to be in state j or above, the system has to meet all the requirements on the number of components at states from 1 to j. In Huang's model of multi-state k-out-of-n:G system in Definition 3.4 in Section 3.1, however, the system is in state j or above if any of the requirements on the number of components at states from j to M can be met.

A special case of the proposed multi-state k-out-of-n:G system model given in Definition 4.9 is defined as follows:

Definition 4.10 *A multi-state k-out-of-n:G system is called a decreasing multi-state k-out-of-n:G system if* $k_1 > k_2 > \cdots > k_M$.

As will be shown later in this chapter, the reliability evaluation of the general case of multi-state k-out-of-n:G system given in Definition 4.9 can be handled through a decreasing multi-state k-out-of-n:G system given in Definition 4.10.

We can also define the multi-state k-out-of-n:F system model as:

Definition 4.11 *An n-component system is called a multi-state k-out-of-n:F system if* $\phi(\mathbf{x}) < j$ $(1 \leq j \leq M)$ *whenever there exists an integer value* l $(1 \leq l \leq j)$ *such that at least* k_l *components are in states below* l.

There is an equivalent multi-state k-out-of-n:G system with respect to each multi-state k-out-of-n:F system, and vice the versa. As to be discussed later in this section, The minimal path vectors of the multi-state k-out-of-n:G system model have special patterns, which enables us to develop efficient reliability evaluation algorithms for it. A multi-state k-out-of-n:F system can be evaluated via its equivalent multi-state k-out-of-n:G system. Thus, we will focus only on multi-state k-out-of-n:G systems in later discussions.

4.2.2 Applications of the multi-state k-out-of-n:G system model

Many engineering systems can fit into the proposed multi-state k-out-of-n:G system model given in Definition 4.9. In this section, we will present two categories of applications that have been identified.

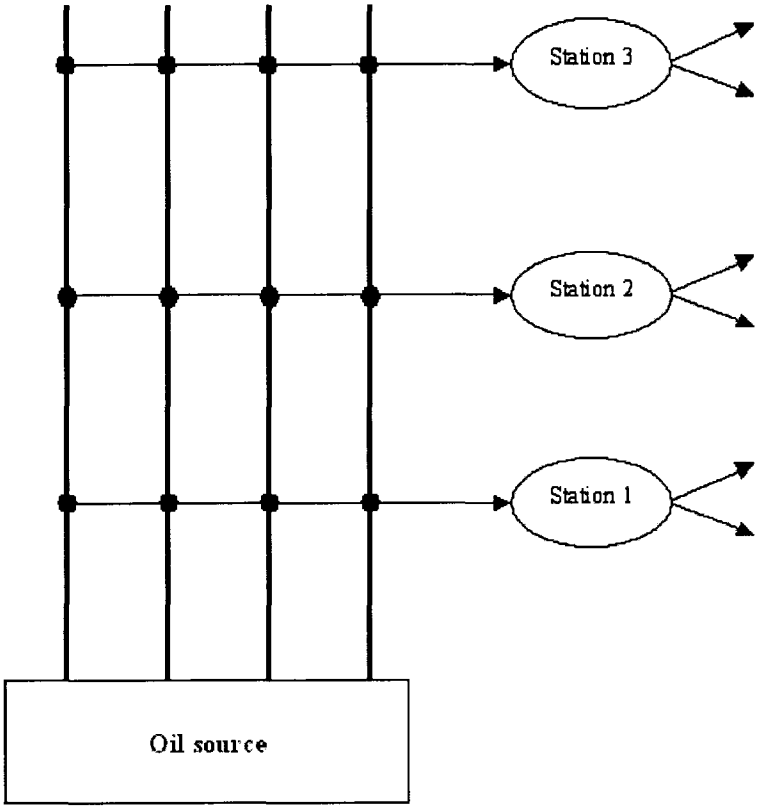

Figure 4.1: An oil supply system

4.2.2.1 Multiple capacity level interpretation of multiple states

In the first category of applications of multi-state k-out-of-n:G system model, multiple states are interpreted as multiple levels of capacity. To fit into the multi-state k-out-of-n:G system model, an engineering system should have the following characteristics: (1) the system is capable of meeting one or multiple types of demands. (2) a component has different levels of capacity, where higher level of capacity means that the component can contribute to meet additional types of demands on the system level. (3) each type of demand on the system level requires at least a certain number of components to contribute to meet the demand. We present the following example of this category of applications of the multi-state k-out-of-n:G system model.

Example 4.6 *Consider an oil supply system, as shown in Fig 4.1. Oil is delivered from the oil source to three stations through 4 oil pipelines. A pipeline is considered to be a multi-state component (thus $n = 4$). Due to the possible failures in different parts of a pipeline and due to the pumping performance of the oil source, a pipeline might be in four possible states:*

- *state 0: oil can not reach any stations.*
- *state 1: oil can reach up to station 1.*
- *state 2: oil can reach up to station 2.*
- *state 3: oil can reach up to station 3.*

Each station has different demands on oil:

- *station 1: requires at least 4 pipelines working to meet its demand.*
- *station 2: requires at least 2 pipelines working to meet its demand.*
- *station 3: requires at least 3 pipelines working to meet its demand.*

On the system level, the oil supply system has four states:

- *system state 0: it can not meet the oil demand of any of the stations.*
- *system state 1: it can meet the oil demand of station 1 only.*
- *system state 2: it can meet the oil demands of station 1 and station 2 only.*
- *system state 3: it can meet the oil demands of station 1, 2 and 3.*

Based on the descriptions above, this oil supply system can be regarded as a multi-state k-out-of-n:G system given in Definition 4.9 with $n = 4$, and $k_1 = 4$, $k_2 = 2$, $k_3 = 3$.

Similar applications can be found in power supply systems and telecommunication systems.

4.2.2.2 Multiple failure mode interpretation of multiple states

In the second category of applications of multi-state k-out-of-n:G system model, multiple states are interpreted in terms of multiple failure modes [22]. A component in a k-out-of-n system has one working state and several failure states. Our view is that a multi-state component might have negative contributions as well as positive contributions to the system. The working state of the component has positive cumulative contributions to the system to perform its intended function, some failure states of the component have no contributions whatsoever to the system, while other failure states of the component have some kinds of negative cumulative contributions to the system. When there are more than a certain number of components in one failure modes with negative contributions, the system will fail due to not being able to tolerate the negative effects, even though there are enough components in the working state to provide positive contributions.

Example 4.7 *Consider a lighting system with n lighting cells. A lighting cell has three possible states (one working state and two failure states):*

- *state 0: in failure state 1, e.g., short circuit.*
- *state 1: in failure state 2, e.g., not available due to open circuit, etc.*
- *state 2: in normal working state.*

On the system level, there are three system states:

- *system state 0: system damaged due to overheat or burned.*

- *system state 1: system is not damaged, but can not meet the demand on lighting.*
- *state 2: system is working properly.*

To prevent system damage due to overheat, at least k_1 lighting cells should be in state 1 or above. To provide enough lighting, it is required that at least k_2 lighting cells should be in state 2, and at least k_1 lighting cells in state 1 or above to prevent system damage. Thus, based on the descriptions above, this lighting system can be regarded as a multi-state k-out-of-n:G system with parameters k_1 and k_2.

4.2.3 Multi-state k-out-of-n:G system model with constant k values

The multi-state k-out-of-n system model with constant k values, which is a special case of the general multi-state k-out-of-n system model, has been studied for a long time, but only on the theoretical stage [19, 8]. After some rephrasing, we present the k-out-of-n emergency shutdown system in a power plant [58] as the first practical application of the multi-state k-out-of-n system model with constant k values.

There are n channels in the emergency shutdown system of a nuclear power plant, detecting whether operating parameters are in the safe ranges. If k channels warn that operating parameters are out of the safe ranges, the power plant will be shut down. A channel has three states:

- state 0: unavailable, i.e., does not warn when it should.
- state 1: warning properly.
- state 2: spurious operation, i.e., warn when it is and is not supposed to.

Here we assume that when a channel is in state 2, it also warns when it should.

The k-out-of-n emergency shutdown system also has three states:

- system state 0: unavailable, i.e., does not warn when it should.
- system state 1: warning properly.
- system state 2: false alarming, i.e., warn when it is not supposed to.

The basic logic for the shutdown system is that if k channels warn, the power plant will be shutdown. Thus, if at least k channels are in state 1 or 2, the system will be shutdown. However, if at least k channels are in state 2, the system will be shutdown spuriously. Thus, the k-out-of-n emergency shutdown system in a power plant is an example of the multi-state k-out-of-n system model with constant k values. Based on the probabilities of the components in different possible states, this model can be used to determine the probabilities of the system in different possible states: unavailable, warning properly, and false alarming.

4.2.4 Minimal path vectors

Using minimal path (cut) vectors is a general way of making reliability evaluations. The definition of minimal path (cut) vectors are given in Section 2.1.5. In this section, we will present the minimal path vectors for the multi-state k-out-of-n:G systems described in this chapter.

4.2.4.1 Minimal path vectors of decreasing multi-state k-out-of-n:G systems

First let's look at the minimal path vectors of strictly decreasing multi-state k-out-of-n systems, where $k_1 > k_2 > \cdots > k_M$.

We would like to present and justify the following form of a minimal path vector to system state 1 of a decreasing multi-state k-out-of-n:G system:

$$\underbrace{(\underbrace{1,1,\ldots,1}_{k_1},0,\ldots,0)}_{n},$$

Let us use $\boldsymbol{v}$ to represent this vector. From the definition of multi-state k-out-of-n:G system, we can see that $\phi(\boldsymbol{v}) \geq 1$ since there are k_1 components in state 1 or above. When any component's state is lowered, if it can be, there will be less than k_1 components which are in state 1 or above, that is, we will have $\phi(\boldsymbol{v}) < 1$. Therefore, vector $\boldsymbol{v}$ is a minimal path vector to system state 1. Since it is assumed that all components are independent, all permutations of the elements of this minimal path vector are all minimal path vectors to system state 1.

Generally speaking, a minimal path vector to system state j has the following form:

$$\underbrace{\underbrace{\underbrace{\underbrace{(\underbrace{j,j,\ldots,j}_{k_j},j-1,\ldots,j-1}_{k_{j-1}},\ldots,1,\ldots,1}_{k_1},0,\ldots,0)}_{n}.$$

We use the symbol $\star$ as a superscript to this minimal path vector to represent all permutations of the elements of this minimal path vector, which are all minimal path vectors to system state j, that is

$$(j,j,\ldots,j,j-1,\ldots,j-1,\ldots,1,\ldots,1,0,\ldots,0)^{\star}$$

represents all minimal path vectors to state j, where $1 \leq j \leq M$.

4.2.4.2 Minimal path vectors of general multi-state k-out-of-n:G systems

For a general multi-state k-out-of-n:G system, the k_j values are not necessarily in a monotonic ordering. The minimal path vector form of multi-state k-out-of-n:G system is similar to that of decreasing multi-state k-out-of-n:G system where the k values are in a strictly decreasing order, except that we need to do some transformations first.

Considering the general case, here we will try to find the minimal path vectors with respect to state j for a multi-state k-out-of-n:G system with values $k_1, k_2, \ldots, k_M$. First we need to find the k_l values $(1 \leq l \leq j)$ which are in strictly increasing order, starting with k_j, for l from j to 1, ignoring those k_l that are not following this order. For instance, if we find that $k_b > k_a > k_j$ $(1 \leq b < a < j)$, it means that $k_l \leq k_j$ $(a < l < j)$, $k_l \leq k_a$ $(b < l < a)$, and $k_l \leq k_b$ $(1 \leq l \leq b)$. And a minimal path vector to state j has the following form:

$$(\underbrace{j, j, \ldots, j}_{k_j}, a, \ldots, a, b, \ldots, b, 0, \ldots, 0) \qquad (4.1)$$

k_j, k_a, k_b, n

All permutations of the elements of this minimal path vector are all minimal path vectors to system state j, which are represented by attaching the symbol $\star$ as a superscript to the minimal path vector above.

Example 4.8 *Let's look at a general multi-state k-out-of-n:G system with 4 components. There are 4 possible states 0, 1, 2 and 3, and* $k_1 = 3$, $k_2 = 1$ *and*

$k_3 = 2$. *We are going to find the minimal path vectors with respect to state 3. First we need to find the* k_l *values which are in strictly increasing order for* l *from 3 to 1, and we got* $k_1 = 3$ *and* $k_3 = 2$ *(*k_2 *is ignored because* k_2 *is not greater than* k_3*). Thus, from Equation (4.1), a minimal path vector to state 3 is* $\mathbf{v}_3 = (3, 3, 1, 0)$*, and thus all the minimal path vectors to state 3 are:*

$\mathbf{v}_3^\star$ = {*(3, 3, 1, 0), (3, 3, 0, 1), (3, 0, 1, 3), (3, 0, 3, 1), (3, 1, 0, 3), (3, 1, 3, 0), (1, 3, 3, 0), (1, 3, 0, 3), (1, 0, 3, 3), (0, 3, 3, 1), (0, 3, 1, 3), (0, 1, 3, 3)*}.

A minimal path vector to state 2 is $\mathbf{v}_2 = (2, 1, 1, 0)$*, and thus all the minimal path vectors to state 2 are:*

$\mathbf{v}_2^\star$ = {*(1, 1, 2, 0), (1, 1, 0, 2), (1, 0, 2, 1), (1, 0, 1, 2), (1, 2, 0, 1), (1, 2, 1, 0), (2, 1, 1, 0), (2, 1, 0, 1), (2, 0, 1, 1), (0, 1, 1, 2), (0, 1, 2, 1), (0, 2, 1, 1)*}.

A minimal path vector to state 1 is $\mathbf{v}_1 = (1, 1, 1, 0)$*, and thus all the minimal path vectors to state 1 are:*

$\mathbf{v}_1^\star$ = {*(1, 1, 1, 0), (1, 1, 0, 1), (1, 0, 1, 1), (0, 1, 1, 1)*}.

As a special case, a minimal path vector to state j for an increasing multi-state k-out-of-n:G system, where the k values are in increasing order, is:

$$\underbrace{(\underbrace{j, j, \ldots, j}_{k_j}, 0, \ldots, 0)}_{n},$$

4.2.5 Nominal decreasing multi-state k-out-of-n:G system model

Similar to the nominal increasing multi-state k-out-of-n:F system model defined in Section 3.1.4, here we define the nominal decreasing multi-state k-out-of-n:G system model.

Definition 4.12 *A nominal decreasing multi-state k-out-of-n:G system is the same as an decreasing multi-state k-out-of-n:G system except that the probability of a component in all possible states may be less than 1.*

A nominal decreasing multi-state k-out-of-n:G system is a result of only considering part of the component states and/or combining several adjacent states together. Suppose we have a general multi-state k-out-of-n:G system with a minimal path vector to state j shown in equation (4.1). To calculate the probability of this system in state j or above, i.e. $P_{s,j}$, we need to generate a nominal decreasing multi-state k-out-of-n:G system with four nominal states. States 0 to $b-1$ of the original system are combined into nominal state "0", states b to $a-1$ are combined into nominal state "1", states a to $j-1$ are combined into nominal state "2", and states j to M are combined into nominal state "3". The probability of a component in nominal state "0" is equal to the sum of the probabilities of the component in state 0 to $b-1$ of the original system, and so forth. As can be seen, the number of k_l values that are useful in equation (4.1) (i.e., k_j, k_a and k_b) is equal to the number of nominal states minus 1.

The nominal decreasing multi-state k-out-of-n:G system model plays an important role in the evaluation of multi-state k-out-of-n:G systems. The $P_{s,j}$ for any state j of a general multi-state k-out-of-n:G system can be evaluated by transforming this general multi-state k-out-of-n:G system into a nominal decreasing multi-state k-out-of-n:G system, and calculating the probability of this nominal decreasing multi-state k-out-of-n:G system in the highest nominal state.

4.3 Evaluation of multi-state k-out-of-n:G systems with i.i.d. components

4.3.1 Reliability evaluation of decreasing multi-state k-out-of-n:G systems with i.i.d. components

To evaluate the state distribution of a decreasing multi-state k-out-of-n:G system, we need to calculate $P_{s,1}, P_{s,2}, \ldots, P_{s,M}$. The probability that the system is in state j or above, i.e. $P_{s,j}$, is equal to the probability that there exists a minimal path vector of the system so that each component state is no less than the corresponding element of the minimal path vector.

Based on the form of minimal path vector for decreasing multi-state k-out-of-n:G systems, to any state j, we have

$$P_{s,j} = \Pr(\mathbf{x} \geq (\underbrace{j, j, \ldots, j}_{k_j}, \underbrace{j-1, \ldots, j-1}_{k_{j-1}}, \ldots, \underbrace{1, \ldots, 1}_{k_1}, \underbrace{0, \ldots, 0}_{n})^{\star}) \quad (4.2)$$

In Section 3.1, we proposed an efficient algorithm for the reliability evaluation of multi-state k-out-of-n:F systems under Huang's definition [37] based on minimal cut vectors. In that algorithm, the probability of the system in states below j, i.e., $Q_{s,j}$, is equal to the probability that there exists a minimal cut vector of the system so that each component state is not bigger than the corresponding element of the minimal path vector:

$$Q_{s,j} =$$

$$\Pr(\mathbf{x} \leq (\underbrace{j-1, j-1, \ldots, j-1}_{k_j}, \underbrace{j, \ldots, j}_{k_{j+1}}, \ldots, \underbrace{M-1, \ldots, M-1}_{k_M}, \underbrace{M, \ldots, M}_{n})^*) \quad (4.3)$$

The probability calculations of $P_{s,j}$ in equation (4.2) and $Q_{s,j}$ in equation (4.3), we would say, are mathematically the same. Thus, we would be able to reformat $P_{s,j}$ and use the algorithm by in Section 3.1 to do the probability calculation. Actually, $P_{s,j}$ in equation (4.2) can be expressed in the form in equation (4.3) by reversing the order of the component states, i.e., letting $p_j = p_{M-j}$. Under the reversed order of component states, $P_{s,j}$ can be expressed as

$$P_{s,j} =$$

$$\Pr(\mathbf{x} \leq (\underbrace{M-j, \ldots, M-j}_{k_j}, \underbrace{M-j+1, \ldots, M-j+1}_{k_{j-1}}, \ldots, \underbrace{M-1, \ldots, M-1}_{k_1}, \underbrace{M, \ldots, M}_{n})^*) \quad (4.4)$$

The probability in Equation (4.4) has the same form as that in Equation (4.3), and thus can be calculated using the algorithm presented in Section 3.1.

The algorithm proposed in Section 3.1, which is simple and elegant, has been shown to be very efficient. In their efficiency investigation example of

an increasing multi-state k-out-of-n:F system with $k_1 = 1$, $k_2 = 2$, $k_3 = 3$, $k_4 = 4$ and $k_5 = 5$, the reliability evaluation time only increases approximately linearly with the increase of the number of components.

4.3.2 Reliability evaluation of general multi-state k-out-of-n:G systems with i.i.d. components

We have presented the minimal path vectors for general multi-state k-out-of-n:G systems in Section 4.2.4.2. The following procedure is proposed to calculate $P_{s,j}$ value to any state j of a general multi-state k-out-of-n:G system:

(1). Find the minimal path vector to state j of this system using the method presented in Section 4.2.4.2.

(2). Generate a nominal decreasing multi-state k-out-of-n:G system to state j, and determine the number of nominal states, $\boldsymbol{k}$ vector and the probability vector $\boldsymbol{p}$. For instance, assume that the minimal path vector to state j of the general multi-state k-out-of-n:G system has the form shown in equation (4.1). Thus the nominal system has 4 possible states, i.e., $M = 3$. The number of components of the nominal system is the same as the general multi-state k-out-of-n:G system. The $\boldsymbol{k}$ vector is $\boldsymbol{k} = (k_b,\ k_a,\ k_j)$. The probability vector $\boldsymbol{p}$ is $\boldsymbol{p} = \left(\sum_{i=0}^{b-1} p_i, \sum_{i=b}^{a-1} p_i,\ \sum_{i=a}^{j-1} p_i,\ \sum_{i=j}^{M} p_i\right)$

(3). Using the approach presented in Section 4.3.1 to calculate the probability of the system in the highest nominal state, i.e. state 3 in this example, of the nominal decreasing multi-state k-out-of-n:G system. This value is equal to $P_{s,j}$ of the considered general multi-state k-out-of-n:G system.

4.4 Evaluation of multi-state k-out-of-n systems with independent components

4.4.1 The proposed reliability evaluation algorithm

A multi-state k-out-of-n:G system with independent components is a more general and practical case. In this case, the probabilities of different components in a certain state can be different. The probability that a system with independent components is in state j or above, i.e. $P_{s,j}$, can be calculated using Equation (4.2). The challenge in evaluating a multi-state k-out-of-n:G systems with independent components is that different components typically have different state distributions in such a system.

The $P_{s,j}$ for any state j of a multi-state k-out-of-n:G system can be evaluated by transforming this multi-state k-out-of-n:G system into a nominal decreasing multi-state k-out-of-n:G system, and calculating the probability of this nominal decreasing multi-state k-out-of-n:G system in the highest nominal state. Therefore, in the following part, we will focus on evaluating the probability of a nominal decreasing multi-state k-out-of-n:G systems with independent components in the highest nominal state.

We propose a recursive algorithm to calculate the probability of a nominal decreasing multi-state k-out-of-n:G systems with independent components in the highest nominal state. **The recursive function** we are using in this recursive algorithm is denoted by $P(n, \boldsymbol{k}, \boldsymbol{P})$, where

- n: the number of components of the nominal decreasing multi-state k-out-of-n:G system,
- $\boldsymbol{k}$: the $\boldsymbol{k}$ vector of the nominal decreasing multi-state k-out-of-n:G system, $\boldsymbol{k} = (k_1, k_2, \ldots, k_M)$, where M is the number of possible states of

the nominal decreasing multi-state k-out-of-n:G system minus 1,

- $\boldsymbol{P}$: the component state distribution matrix for the nominal decreasing multi-state k-out-of-n:G system,

$$\boldsymbol{P} = \begin{pmatrix} p_{1,0} & p_{1,1} & \cdots & p_{1,M} \\ p_{2,0} & p_{2,1} & \cdots & p_{2,M} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n,0} & p_{n,1} & \cdots & p_{n,M} \end{pmatrix}$$

The recursive function $P(n, \boldsymbol{k}, \boldsymbol{P})$ is designed to represent the probability in the highest nominal state of a nominal decreasing multi-state k-out-of-n:G system with n independent components, vector $\boldsymbol{k}$, and probability matrix $\boldsymbol{P}$. To evaluate $P_{s,j}$ in equation (4.2), the states j and above are combined into the highest nominal state of the generated nominal decreasing multi-state k-out-of-n:G system with vector $\boldsymbol{k}$ and probability matrix $\boldsymbol{P}$. Thus we have $P_{s,j} = P(n, \boldsymbol{k}, \boldsymbol{P})$.

The recursive algorithm is as follows:

$$P(n, \boldsymbol{k}, \boldsymbol{P}) = \sum_{j=0}^{M} p_{n,j} \cdot P(n-1, \boldsymbol{k}^j, \boldsymbol{P}^j) \tag{4.5}$$

The basic idea of this algorithm is similar to that of the recursive algorithm for binary multi-state k-out-of-n systems: we enumerate the cases where component n is in different possible states, and thus evaluate a system with n components via evaluating several systems with $n-1$ components. For each certain j on the right hand side of equation (4.5), we need to reorganize $\boldsymbol{k}$ and $\boldsymbol{P}$ to generate $\boldsymbol{k}^j$ and $\boldsymbol{P}^j$.

First for $0 < j < M$,

$$k_l^j = k_l, \quad \text{for} \quad l > j$$

$$k_l^j = k_l - 1, \qquad \text{for} \quad l \leq j. \tag{4.6}$$

The idea is that if component n is in state j, the required number of components for any state equal to or below j should be decreased by 1. For $j = M$, we have $k_l^j = k_l - 1$ for $1 \leq l \leq M$. For $j = 0$, we have $k_l^j = k_l$ for $1 \leq l \leq M$. $\boldsymbol{P}^j$ is obtained by deleting the n^{th} row from matrix $\boldsymbol{P}$, and thus $\boldsymbol{P}^j$ is a $n-1$ by $M+1$ matrix.

There is a special case when generating $\boldsymbol{k}^j$ and $\boldsymbol{P}^j$, under which a certain state will be "absorbed" by the adjacent upper state, and we should make further transformations on $\boldsymbol{k}^j$ and $\boldsymbol{P}^j$. This special case is the case when $k_h^j = k_{h-1}^j$ for a state h $(1 \leq h \leq M)$. In this case, state $h-1$ is absorbed by state h. k_{h-1}^j is deleted from $\boldsymbol{k}^j$; $p_{i,h}^j = p_{i,h}^j + p_{i,h-1}^j$ for $1 \leq i \leq n-1$, and then the column $h-1$ is deleted from $\boldsymbol{P}^j$. Thus, the number of possible states is decreased from $M+1$ to M. This is actually a major reason that the computation time using this algorithm will not increase exponentially with the increase of n. By generating $\boldsymbol{k}^j$ and $\boldsymbol{P}^j$ in this way, $\boldsymbol{k}^j$ will always be a strictly increasing vector.

The boundary conditions for the recursive algorithm are as follows:

Boundary condition 1: $k_1 > n$. In this case, $P(n, \boldsymbol{k}, \boldsymbol{P}) = 0$.

Boundary condition 2: $M = 1$. In this case, the decreasing multi-state k-out-of-n:G system is reduced to a binary k-out-of-n:G system with independent components. The recursive algorithms by by Barlow and Heidtmann [4] and Rushdi [82] can be used for the reliability evaluation.

The approaches to evaluate other types of multi-state k-out-of-n:G systems with independent components are as follows.

1. As mentioned in the previous section, the $P_{s,j}$ for any state j of a general

multi-state k-out-of-n:G system can be evaluated by transforming this system into a nominal decreasing multi-state k-out-of-n:G system, and calculating the probability of this nominal decreasing multi-state k-out-of-n:G system in the highest nominal state "M" .

2. An increasing multi-state k-out-of-n:G system is a special case of a general multi-state k-out-of-n:G system. An increasing multi-state k-out-of-n:G system can also be transformed into a nominal decreasing multi-state k-out-of-n:G system. To any state j, the generated nominal decreasing multi-state k-out-of-n:G system is actually a binary k-out-of-n:G system with $k = k_j$. The probability of component i in nominal state "1" is $\sum_{l=j}^{M} p_{i,l}$. "Boundary condition 2" of the proposed recursive algorithm will be activated, and binary k-out-of-n:G algorithms can be used for reliability evaluation.

4.4.2 Examples

In this section, we will use several examples to illustrate the correctness and efficiency of the proposed recursive algorithm for reliability evaluation of multi-state k-out-of-n:G systems with independent components.

4.4.2.1 Example 4.1: Evaluation of an decreasing multi-state k-out-of-n:G systems with i.i.d. components

A decreasing multi-state k-out-of-n:G system with i.i.d. components is a special case of decreasing multi-state k-out-of-n:G system with independent components, and thus the proposed recursive algorithm should be applicable in this case.

We consider a decreasing multi-state k-out-of-n:G system with four i.i.d.

components and three possible states. The $\boldsymbol{k}$ vector is $\boldsymbol{k} = (k_1,\ k_2) = (3,\ 2)$. The state distribution of components is $\boldsymbol{p} = (0.5,\ 0.3,\ 0.2)$. Using the recursive algorithm, we got

$$\boldsymbol{P}_s = (0.3125,\ 0.1208), \quad \boldsymbol{r}_s = (0.6875,\ 0.1917,\ 0.1208).$$

That is, the probability of the system in state 2 or above is 0.1208, and the probability of the system in state 1 or above is 0.3125. And the probability of the system in state 0, 1, and 2 is 0.6875, 0.1917 and 0.1208, respectively.

We also applied the enumerating method and the algorithm for systems with i.i.d. components presented in Section 4.3 to this example. Their results agree with the result listed above, which illustrates the correctness of the proposed recursive algorithm in the case of i.i.d. components.

4.4.2.2 Example 4.2: Evaluation of a decreasing multi-state k-out-of-n:G systems with independent components

In this example, we consider an decreasing multi-state k-out-of-n:G system with independent components. The system under consideration has five independent components and five possible states. The $\boldsymbol{k}$ vector is $\boldsymbol{k} = (k_1,\ k_2,\ k_3,\ k_4) = (4,\ 3,\ 2,\ 1)$. The state distribution matrix is

$$\boldsymbol{P} = \begin{pmatrix} 0.2 & 0.4 & 0.1 & 0.2 & 0.1 \\ 0.3 & 0.3 & 0.1 & 0.1 & 0.2 \\ 0.4 & 0.2 & 0.1 & 0.2 & 0.1 \\ 0.2 & 0.1 & 0.1 & 0.2 & 0.4 \\ 0.4 & 0.2 & 0.2 & 0.1 & 0.1 \end{pmatrix}$$

Using the recursive algorithm, we got

$$\boldsymbol{P}_s = (0.5261,\ 0.3350,\ 0.2939,\ 0.2526)$$

$$\boldsymbol{r}_s = (0.4739,\ 0.1910,\ 0.0412,\ 0.0413,\ 0.2526).$$

That is, the probability of the system in state 0, 1, , 2, 3 and 4 is 0.4739,

0.1910, 0.0412, 0.0413 and 0.2526, respectively. We also used the enumerating method to evaluate this system, and the results agree with the results by the recursive algorithm, which illustrates the correctness of the recursive algorithm in case of independent components.

4.4.2.3 Example 4.3: Efficiency investigation of the proposed recursive algorithm

In this section, first we consider an decreasing multi-state k-out-of-n:G system with 5 possible states and n independent components. The state distribution of each component is randomly generated. The $\boldsymbol{k}$ vector is $\boldsymbol{k} = (k_1,\ k_2,\ k_3,\ k_4) = (4,\ 3,\ 2,\ 1)$. Apparently $P_{s,4}$ requires the longest calculation time among all $P_{s,l}$ $(1 \leq l \leq 4)$. Here we will investigate the time to calculate $P_{s,4}$ with respect to different number of components n. The resulting computation time (seconds) of the recursive algorithm with respect to n is shown in Table 4.1. The calculations were made with a computer with Pentium-M (1.7GHz) CPU, 512MB RAM, Windows XP Professional and MATLAB 6.5. The enumerating method is also used to evaluate these systems, and the computation time is listed in Table 4.1 as well.

Table 4.1: Computation time (seconds) for $P_{s,4}$ versus n for systems with 5 possible states

n	5	8	10	15	20	30
Recursive Algorithm (Seconds)	0.06	0.30	0.60	2.50	7.55	36.9
Enumerating Method (Seconds)	0.05	3.78	89.50	-	-	-

We can see that with the increase of the number of components, the computation time of the proposed algorithm increases, but does not increase dramatically. The increase of computation time seems to be polynomial. On

the other hand, the computation time by the enumerating method increases dramatically with the increase of n. This example has illustrated that the recursive algorithm is far more efficient than the enumerating method, especially to systems with a large number of components.

Next, we consider the case when the considered systems have 3 possible states. The $\boldsymbol{k}$ vector is $\boldsymbol{k} = (k_1, k_2) = (3, 2)$. We will investigate the time to calculate $P_{s,2}$ with respect to different n value. The results are shown in Table 4.2. We can see that the increase magnitude of the computation time with respect to n increased from a certain number to another (say, from 15 to 20) is relatively smaller than the case of systems with 5 possible states.

Table 4.2: Computation time for $P_{s,2}$ versus n for systems with 3 possible states

n	5	8	10	15	20	30	40
Computation time (Seconds)	0.01	0.05	0.08	0.24	0.40	0.80	2.05

To have a better idea on how good the efficiency of the proposed recursive algorithm is, we will investigate the efficiency of the algorithm by Barlow and Heidtmann [4] for evaluating a binary k-out-of-n system. The considered system is a binary k-out-of-n:G system with n independent components, and $k = 3$. The reliability of each component is randomly generated. The efficiency investigation results are shown in Table 4.3. We can see that the trend of the computation time versus n in evaluating the binary k-out-of-n:G systems using the binary recursive algorithm is similar to those when evaluating multi-state systems with 5 or 3 possible state using the proposed recursive algorithm. If we examine them more closely, we can see when n is increased from 15 to 20, for instance, the computation time of evaluating the binary systems will in-

crease about 2.3 times (from 0.07 to 0.16), the computation time of evaluating multi-state systems with 3 possible states will increase about 1.7 times (from 0.24 to 0.40), and the computation time of evaluating multi-state systems with 5 possible states will increase about 3 times (from 2.50 to 7.55). These increase magnitudes of the computation time with the increase of n are close, at least they are at a similar level. Therefore, we can say that the efficiency of the proposed recursive algorithm for multi-state systems is comparable to that of the binary system recursive evaluation algorithms. And the proposed recursive algorithm seems to be a natural extension of the binary system recursive evaluation algorithm.

Table 4.3: Efficiency of the algorithm for binary k-out-of-n systems

n	5	8	10	15	20	30	40	60
Computation time (Seconds)	0.01	0.015	0.02	0.07	0.16	0.31	0.63	1.76

4.4.3 Computational complexity analysis

In the following, we will provide an upper bound on the computational complexity of the recursive algorithm presented in Section 4.4.1.

For evaluation of each $P(n, \boldsymbol{k}, \boldsymbol{P})$ term in Equation (4.5) given all its arguments, it takes $2M + 1$ summation and multiplication operations. The argument n can take values from 1 to n. The jth element of vector $\boldsymbol{k}$ can take values in $[0,\ k_j]$. Therefore, the complexity of the proposed algorithm using

Equation (4.5) is

$$O\left(M \cdot n \cdot \left(\prod_{j=1}^{M}(k_j+1)\right)\right) \tag{4.7}$$

When the proposed algorithm is used to evaluate the system reliability of a binary k-out-of-n system, there is only one element in the $\boldsymbol{k}$ vector. The proposed algorithm described in Section 4.4.1 is reduced to the algorithm by Rushdi [82] and the second BASIC program by Barlow and Heidtmann [4]. The computational complexity in this case becomes $O(k(n-k+1))$.

An alternative to the proposed approach for reliability evaluation of multi-state k-out-of-n:G systems with independent components is to use a generalized multinomial distribution. We will illustrate this idea here and show that it will result in a less efficient algorithm.

The idea of the generalized multinomial distribution can be described as follows [5]. Consider a statistical experiment wherein there are L groups and on the nth trial, the probability of joining group j is $q_{n,j}$. Let $\mathbf{x} = (x_1, x_2, \ldots, x_L)$ denote the numbers of items in each of the L groups. One is interested in the probability of being in state $\mathbf{x}$ after n trials.

We now describe the idea of the generalized multinomial distribution using the terminology of multi-state reliability analysis. Consider a system with n components wherein each component and the system may be in $M+1$ possible states. Let x_j denote the number of components in state j for $j = 0, 1, 2, \ldots, M$, and $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_M)$. If we are interested in finding the probability of having exactly x_j components in state j for $j = 0, 1, 2, \ldots, M$,

the following formula may be used:

$$P_n(\mathbf{x}) = \sum_{j=0}^{M} q_{n,j} P_{n-1}(T_j\mathbf{x}), \tag{4.8}$$

where $P_j(\bullet)$ is the recursive function, $(T_j\mathbf{x})$ is the updated vector of $\mathbf{x}$ when component n is in state j, and

$$\begin{aligned} (T_j\mathbf{x})_i &= x_j - 1, \quad \text{for } i = j; \\ (T_j\mathbf{x})_i &= x_i, \quad \text{for } i \neq j. \end{aligned}$$

Equation (4.8) can be used to evaluate the probability that *exactly* x_j components are in state j for $j = 0, 1, 2, \ldots, M$. However, the definitions of the multi-state k-out-of-n:G systems are in terms of *at least* a certain number of components being in a certain state or above a certain state. Thus, M additional summation loops have to be performed if one is to use Equation (4.8) for evaluation of the state distribution of generalized multi-state k-out-of-n systems. The computational complexity of this approach would then be $O(Mn^{M+1})$. When a binary k-out-of-n system is considered, $M = 1$, this approach is similar to the first BASIC program by Barlow and Heidtmann [4], with the computational complexity of $O(n^2)$. Thus, this approach is less efficient than the one proposed in this chapter, which evaluates the probability that *at least* a certain number of components are in a certain state range directly.

4.5 Concluding remarks

In this chapter, we attempt to develop a new multi-state k-out-of-n system model which allows different requirements on the number of components for different state levels, and, very importantly, more practical engineering systems can fit into this model. The developed model is a multi-state k-out-of-n:G system model. Two categories of applications of the multi-state k-out-of-n system model have been identified: (1) In the first category, multiple states are interpreted as multiple levels of capacity. The system has different requirements at the number of components on different levels. (2) In the second category, multiple states are interpreted in terms of multiple failure modes. The working state of a component has positive cumulative contributions to the system, some failure states of a component have no contributions whatsoever to the system, while other failure states of a component have some kinds of negative cumulative contributions to the system. An approach is presented for efficient reliability evaluation of multi-state k-out-of-n systems with i.i.d. components. A recursive algorithm is proposed for reliability evaluation of multi-state k-out-of-n systems with independent components. Numerical investigations show that the proposed algorithm is efficient. The emergency shutdown system in a power plant is presented as an application of the multi-state k-out-of-n system model with constant k value, which is a special case of the general multi-state k-out-of-n system model.

The multi-state k-out-of-n system model presented in this chapter is more flexible than the model proposed by Huang *et al.* [37]. However, it is still not flexible enough to explain many engineering systems with k-out-of-n system structures. A more flexible and unified k-out-of-n model is yet to be developed.

CHAPTER 5

A UNIFIED K-OUT-OF-N SYSTEM MODEL AND ITS EVALUATION

5.1 Introduction

We have discussed the generalized multi-state k-out-of-n system model defined by Huang *et al.* [37] in Chapter 3, and another more flexible multi-state k-out-of-n model in Chapter 4. However, these models are still not flexible enough to explain many engineering systems with k-out-of-n system structures. In this chapter, we propose a unified k-out-of-n model for reliability modeling and evaluation of systems with complex k-out-of-n structures. The materials in this section has been documented in paper [89].

5.1.1 Reported models of k-out-of-n systems

There are currently several models of k-out-of-n systems, listed as follows (here we don't consider consecutive systems [36]).

(1) Binary k-out-of-n system models. A system with n components is called k-out-of-n:G system if it is working as long as there are at least k components working, and a system with n components is called k-out-of-n:F system if it

is failed as long as there are at least k components failed. There are wide applications of binary k-out-of-n systems in both industrial and military systems. Efficient reliability evaluation algorithms for binary k-out-of-n systems with independent components have been provided by Barlow and Heidtmann [4] and Rushdi [82].

(2) Multi-state k-out-of-n system models. Boedigheimer and Kapur [8] and El-Neweihi and Proschan [19] presented multi-state k-out-of-n models with a constant k value. Huang *et al.* [37] proposed the generalized multi-state k-out-of-n model which allows different k values with repsect to different states, with efficient evaluation algorithms following up (Section 3.1). We proposed another multi-state k-out-of-n model so that more practical applications can fit into it (Chapter 4). The reasons that a component is multi-state, we would like to summarize here, are the following three:

(a) the component has multiple levels of performance in a certain performance measure [46, 47, 55], e.g., multi-state power generator with different levels of power outputs;

(b) the component has multiple failure modes [22], e.g., a light bulb has three possible states: working state, failed open state and failed closed state as discussed in Section 4.2.2. Another example is the fluid valves in the flow control system [47];

(c) the component can provide multiple functions. The example of oil pipeline, as described in Section 4.2.2, could be considered as an example of this. Another example is a machine tool that can do two machining processes, such as milling and turning, and these two functions are not totally independent in terms that they will be both failed if the main power of the machine fails.

(3) Weighted k-out-of-n system models. The weighted k-out-of-n system model was proposed in the binary context by Wu and Chen [97], and generalized to the multi-state context by Li and Zuo [52]. Each state of a component is associated with a weight, and the system is said to be working if the total weight of the components is above a pre-specified threshold value.

5.1.2 The core characteristics of k-out-of-n systems

There are three core characteristics that all k-out-of-n systems mentioned above share:

(1) a constituent component of the system has some performance indexes (functions and/or failure modes);

(2) for each of a group of selected performance indexes, the system has certain requirement on the combined effect of the components;

(3) the topological positions of the components do not matter.

Based on these common characteristics, we define the unified k-out-of-n system model to be presented in this chapter.

5.1.3 Motivation

The reported models of k-out-of-n systems have certain limitations. The multi-state k-out-of-n model by Huang *et al.* has limited applications [37], as discussed in Section 4.1. The model by proposed in Chapter 4 has more practical applications, however, the states of a component have to satisfy certain requirements in orde for the model to be used, as described in Section 4.2. The model of weighted k-out-of-n systems only considers one performance measure of a system and its components [97, 12, 52]. On the other hand, practical applications might involve more general and complex situations, where requirements

on multiple performance indexes might be involved, and these performance indexes might be dependent.

In this chapter, we will propose a so-called "unified k-out-of-n model". This model aims to provide high flexibility to model and analyze practical and complex problems involving k-out-of-n structures. Efficient reliability evaluation algorithms will also be developed for this model.

Based on the common characteristics of k-out-of-n systems, we define the unified k-out-of-n system model. The unified k-out-of-n system model proposed in this chapter can deal with all the k-out-of-n related scenarios covered so far in the literature: (1) a component can have multiple performance indexes (functions and/or failure modes); (2) a function can have multiple levels of performance; (3) the performance indexes can be dependent.

In the remainder of this chapter, first we will describe the proposed unified k-out-of-n system model with its fundamental elements. An algorithm will be developed for the reliability evaluation of the unified k-out-of-n systems, and an numerical example will be used to illustrate the reliability evaluation algorithm.

5.2 The unified k-out-of-n model

5.2.1 Fundamental elements of the unified k-out-of-n model

5.2.1.1 Function and performance level

A component may have only one function, e.g., a power generator for generating power. A component may also have multiple functions. For example, a machine tool might be able to conduct two different machining processes such as turning and milling.

A function of a component might have multiple "performance levels". For example, a power generator might have three performance levels: 0 MW (mega watt) output, 4 MW output and 10 MW output.

5.2.1.2 Failure modes and performance index

A component might have one or multiple failure modes. For example, a fluid valve can fail open or fail closed. The system might fail if there are over a certain number of components in a certain failure mode [46, 47].

The term "performance index" is used to denote a function or a failure mode that we are interested in.

5.2.1.3 Dependencies among functions and/or failure modes in a component

The functions and/or failure modes of a component might be dependent. They can be regarded as the attributes of the component. Take for example the machine tool that is able to conduct two machining processes. These two functions are dependent in that both of them will be failed if the motor of the machine is failed.

The capability of dealing with components with dependent functions and/or failure modes is a key advantage of the unified k-out-of-n model, comparing to previous k-out-of-n models.

5.2.1.4 Function diagram

A function diagram is used to describe the relationships among the functions and/or failure modes of a component, and thus determine the component state distribution. Examples of the function diagrams will be given later in Section 5.2.4.

5.2.1.5 Component utilities

A group of performance indexes (functions and/or failure modes) will influence the overall system performance, and these are the performance indexes that we consider.

A function, or a performance level of a function if the function has multiple levels, carries a certain utility depending on the contribution it makes to the whole system. A function can make positive or negative contribution to the system. Positive contribution of a component is easy to understand, e.g., power output of a power generator. Examples of negative contributions of a component include resource consumptions, noise, pollution, etc. If the requirement of the system on the function is expressed as at least (or at most) how many components should be able to perform the function, it is a k-out-of-n requirement; If the requirement of the system on the function is expressed as the total utilities of all the components should be at least (or at most) a certain value, it is a weighted-k-out-of-n requirement.

A failure mode under consideration will make negative contributions to the system. It can be treated as a function with negative contributions.

The utility of a function or failure mode with k-out-of-n requirement is 1. The utility of a function or failure mode with weighted-k-out-of-n requirement is a positive integer value. If a component does not have a certain function or failure mode, the component's utility with respect to the function or failure mode is 0.

5.2.1.6 System requirements

The system has certain requirement on the total utilities of all components on each of the selected group of performance indexes (functions and/or failure

modes). For the system to be in system state s or above, it has to meet all the requirements. A requirement, indexed by j, might be expressed in one of the forms shown in Table 5.1 (suppose there are n components in the system), where u_{ij} is the utility (associated with index j) of component i, and k_j^s is the system requirement on this index. For a function with positive contribution, it is like a k-out-of-n:G structure, and the "$\geq$" is used. For a function with negative contribution or a failure mode, it is like a k-out-of-n:F structure, and the "$<$" is used.

Table 5.1: System requirement on a function or failure mode

	Requirement
Function (positive contribution)	$\sum_{i=1}^{n} u_{ij} \geq k_j^s$
Function (negative contribution)	$\sum_{i=1}^{n} u_{ij} < k_j^s$
Failure mode	$\sum_{i=1}^{n} u_{ij} < k_j^s$

5.2.2 Definition of the unified k-out-of-n model

A system with n components is called a unified k-out-of-n system if:

(1) a component has one or multiple performance indexes (functions and/or failure modes), associated with certain utilities. Some of the performance indexes can be dependent.

(2) in order to be in a certain system state s or above, the system has to meet certain requirement on each of a selected group of performance indexes (functions and/or failure modes). A requirement j on a performance index with positive contribution to the system is expressed as $\sum_{i=1}^{n} u_{ij} \geq k_j^s$. And a requirement j on a performance index with negative contribution is expressed as $\sum_{i=1}^{n} u_{ij} < k_j^s$.

J is used to denote the total number of the selected performance indexes (functions and/or failure modes).

5.2.3 Component states in the unified k-out-of-n model

Functions, performance levels and failure modes are not component states. It is more appropriate to describe them as the causes for a component to have multiple states. A component state is characterized by its utility vector where each element corresponds to a certain function or failure mode. (1) All the states of a component are mutually exclusive; (2) All the states of a component make a complete component state space.

Usually, the component states are the combinations of the J selected functions (and their performance levels) and/or failure modes. However, it is also possible that two functions, for example, are exclusively associated with one state.

System states are defined based on the requirements on the performance indexes. An example will be given in the next section to illustrate how the system states are defined.

5.2.4 Examples of the unified k-out-of-n model

Several examples of the unified k-out-of-n systems are given in this section. The function diagrams are given, and the component states are defined.

5.2.4.1 Machine tool example

Many multi-functional machining centers have the capability to do several machining processes.

Consider a machine tool with two functions: turning and milling. The

function diagram is shown in Figure 5.1. Link L1, L2 and L3 determine how the functions fail. Link L1 is the common cause of the failures of the two functions. The turning function will fail if link L1 or link L2 fails, and the milling function will fail if link L1 or link L3 fails. Based on the two functions, four states are defined for the component:

- state 0: the component has neither of the two functions;
- state 1: the component has the turning function, but does not have the milling function;
- state 2: the component has the milling function, but does not have the turning function;
- state 3: the component has both of the two functions.

Suppose that a system with n machine tools requires that at least k_1 machines with the turning function to meet the turning demand, and at least k_2 machines with the milling function to meet the milling demand. Four system states can be defined:

- state 0: the system can not meet either of the demands;
- state 1: the system can meet the turning demand, but can not meet the milling demand;
- state 2: the system can meet the milling demand, but can not meet the turning demand;
- state 3: the system can meet both of the demands.

Thus, this system is a unified k-out-of-n system with k-out-of-n requirements on two functions.

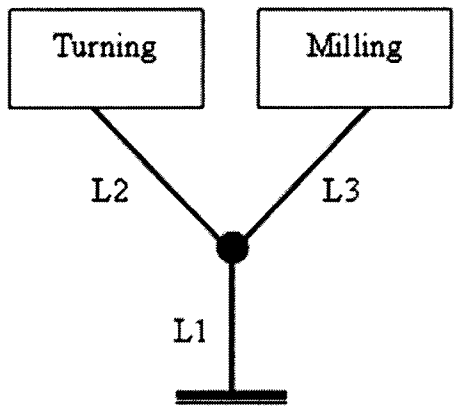

Figure 5.1: Function diagram of the machine tool

5.2.4.2 Machine tool involving multiple performance levels

The case above can be extended to the case where one or more of the functions has multiple performance levels, as shown in Figure 5.2. In this case, the turning function has two performance levels, while the milling function has only one performance level. And the component states are defined as follows based again on the two functions:

- state 0: the component has neither of the two functions;
- state 1: the component has the turning function at level 1, but does not have the milling function;
- state 2: the component has the turning function at level 2, but does not have the milling function;
- state 3: the component does not have the turning function, but it has the milling function.
- state 4: the component has the turning function at level 1, and it has the milling function.

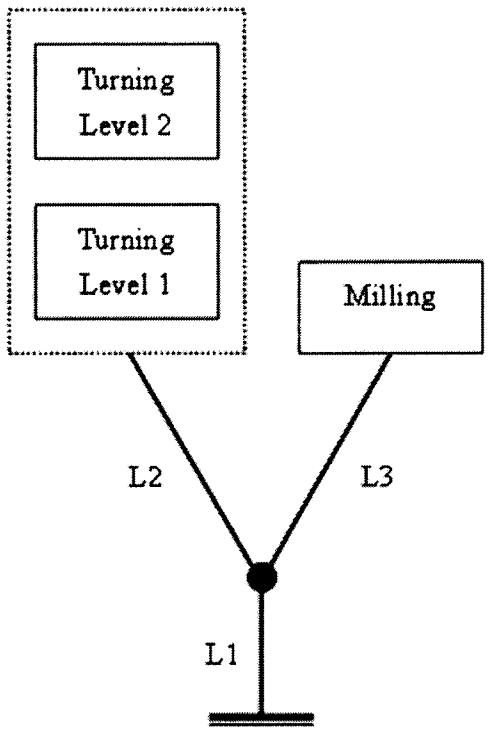

Figure 5.2: Function diagram of a machine tool involving multiple performance levels

- state 5: the component has the turning function at level 2, and it has the milling function.

Suppose that a system with n machine tools requires that at least k_2 machines with milling function should be available to meet the demand. Performance levels 1 and 2 of the turning function correspond to certain utilities u_{11} and u_{12}. The sum of the turning function utilities of all the components should be great than or equal to k_1 to meet the system demands. Four system states can be defined in the same way as that in the previous example:

- state 0: the system can not meet either of the demands;
- state 1: the system can meet the turning demand, but can not meet the

milling demand;

- state 2: the system can meet the milling demand, but can not meet the turning demand;
- state 3: the system can meet both of the demands.

Thus, this system is a unified k-out-of-n system with weighted-k-out-of-n requirement on the turning function and k-out-of-n requirement on the milling function.

5.2.4.3 Fluid valve example

Consider a flow control system with n valves connected in parallel. A valve, as shown in Figure 5.3, has two failure modes: "stuck open", failed to close when it is demanded to, and "stuck closed", failed to open when it is demanded to. That is, a valve (component) has three possible states: state 0 (stuck closed state), state 1 (working state) and state 2 (stuck open state).

A flow control system is working properly if it can control the flow rate fr within a certain range, from the lower limit fr_{ll} to the upper limit fr_{ul} [46, 47]. Thus, the system would be failed when there are at least k_1 (a pre-specified value) valves in "stuck open" failure mode, since it won't meet the flow rate lower limit in this case. And the system would be failed if there are at least k_2 valves in "stuck closed" failure mode, since it won't be able to meet the flow rate upper limit in this case. Thus, this flow control system is a unified k-out-of-n system with k-out-of-n requirements on two failure modes.

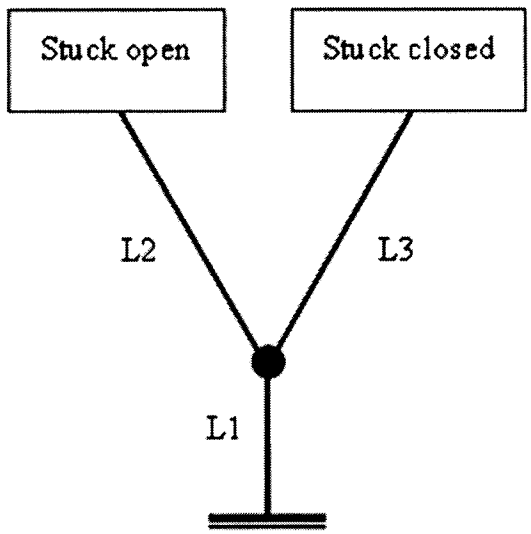

Figure 5.3: Function diagram of a fluid valve

5.2.4.4 Light bulb example

Consider a lighting system with n light bulbs. Each light bulb has dual failure modes: failed open and failed short, as shown in Figure 5.4.

If one light bulb is in short failure mode, the whole lighting system will be disconnected. Thus, for the system to be working, there are two requirements: at least k_1 light bulb working to provide enough lighting, and no light bulb should be in the short failure mode. Thus, this is another example of a unified k-out-of-n system with k-out-of-n requirements on multiple failure modes.

5.2.4.5 A production system

Consider a production system with n machines. The machines have different production rates, and they have different operating costs. The system is regarded as working satisfactorily if the total production rate is at least k_1, and the total operating cost is at most k_2. Thus, the production system is a system with k-out-of-n requirements on both production rate and cost.

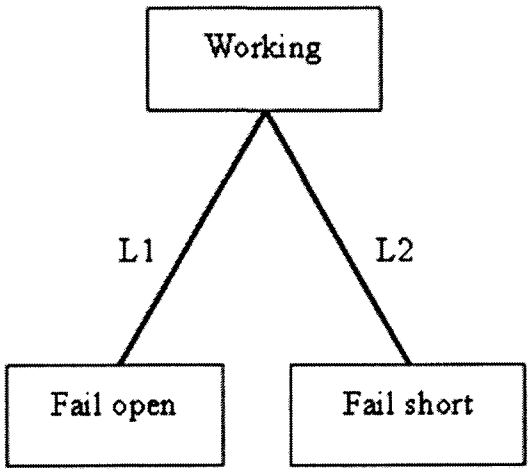

Figure 5.4: Function diagram of a light bulb with dual failure modes

5.3 The reliability evaluation algorithms

The reliability evaluation algorithms are the extensions of the recursive algorithms for binary k-out-of-n systems [4, 82], multi-state k-out-of-n systems presented in Chapter 4, and weighted-k-out-of-n systems [97, 52].

The general definition of unified k-out-of-n systems is given in Section 5.2.2. However, there are several special cases of the general model. The special cases are defined based on whether k-out-of-n requirements and/or weighted k-out-of-n requirements are included, and whether requirements on positive and/or negative contributions are included.

We will first present the reliability evaluation algorithms for some special cases, and finally we will present the algorithm for the general model. We are interested in calculating the probability of the system in a certain state s or above. First, some notions are defined as follows:

- n: the number of components of the system.
- J: the total number of performance indexes (functions and/or failure

modes) of interest.

- M: the total number of possible states of a component minus 1.
- p_{im}: probability of component i in state m.
- $\boldsymbol{k}^s$: the $\boldsymbol{k}$ vector of the system with respect to system state s, $\boldsymbol{k}^s = (k_1^s, k_2^s, \ldots, k_J^s)$.
- $\boldsymbol{k}^{sG}$, $\boldsymbol{k}^{sF}$, $\boldsymbol{k}^{sWG}$, $\boldsymbol{k}^{sWF}$: the $\boldsymbol{k}$ vector for the performance indexes with k-out-of-n:G requirements, k-out-of-n:F requirements, weighted-k-out-of-n:G requirements and weighted-k-out-of-n:F requirements, respectively.
- $R(i, \boldsymbol{k}^s)$: recursive function.
- $\boldsymbol{u}^{im}$: the utility vector of component i when it is in state m.

5.3.1 Reliability evaluations of some special cases of the unified k-out-of-n systems

5.3.1.1 Systems with k-out-of-n:G structure

This is the case where only k-out-of-n requirements and positive contributions are involved. The machining system with components presented in Figure 5.1 is an example of this case. We define the reliability of the system as the probability of the multi-state system in system state s or above, that is, the probability that the system meet the requirements specified by vector $\boldsymbol{k}^s$.

The "**recursive function**" $R(i, \boldsymbol{k}^s)$ represents the reliability of the system with the first i components, i.e., the probability of such a system meeting the requirements specified by vector $\boldsymbol{k}^s$.

The **updating algorithm** for the reliability evaluation is as follows:

$$R(i, \boldsymbol{k}^s) = \sum_{m=0}^{M} p_{i,m} \cdot R(i-1, \boldsymbol{k}^{sm}) \tag{5.1}$$

$$\boldsymbol{k}^{sm} = \boldsymbol{k}^s - \boldsymbol{u}^{im} \quad (k_l^{sm} = 0, \ \texttt{if} \ \ k_l^{sm} < 0) \tag{5.2}$$

where $\boldsymbol{u}^{im}$ represents the utility vector of component i when it is in state m, and k_l^{sm} is the lth element of vector $\boldsymbol{k}^{sm}$. $\boldsymbol{u}^{im}$ is a vector with J elements, where an element j represent the utility with respect to performance index j. In this case of k-out-of-n:G structure, the elements of $\boldsymbol{u}^{im}$ can only take 0 or 1.

The **boundary conditions** are:

$$R(i, \boldsymbol{k}^s) = 1, \quad \texttt{if} \ \ \texttt{max}(\boldsymbol{k}^s) = 0 \tag{5.3}$$

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if} \ \ i < \texttt{max}(\boldsymbol{k}^s) \tag{5.4}$$

where $\texttt{max}(\boldsymbol{k}^s)$ represents the biggest element of vector $\boldsymbol{k}^s$.

The computational complexity of the algorithm is

$$O\left(M \cdot n \cdot \left(\prod_{j=1}^{J} (k_j^s + 1)\right)\right) \tag{5.5}$$

It is worth noting that the updating algorithm in Equation (5.1) and (5.2) is the general updating algorithm for unified k-out-of-n systems, and it is applicable to all the cases of unified k-out-of-n systems. The only differences among the algorithms for different special cases are their boundary conditions. This algorithm can be regarded as an extension of the recursive algorithms for binary k-out-of-n systems [4, 82], multi-state k-out-of-n systems as discussed

in Chapter 4, and weighted-k-out-of-n systems [97, 52], as mentioned earlier in this section.

5.3.1.2 Systems with k-out-of-n:F structure

This is the case where only k-out-of-n requirements and negative contributions are involved. The flow control system with components presented in Figure 5.3 is an example of this case.

As mentioned in the previous section, the updating algorithm is still the same, as presented in Equation (5.1) and (5.2). The k-out-of-n:F structure is different from the k-out-of-n:G structure in that the relationship among different requirements is "OR" instead of "AND".

With all the other parts of the reliability evaluation algorithm the same, the **boundary conditions** of this case are:

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if } \texttt{min}(\boldsymbol{k}^s) = 0 \tag{5.6}$$

$$R(i, \boldsymbol{k}^s) = 1, \quad \texttt{if } i < \texttt{min}(\boldsymbol{k}^s) \tag{5.7}$$

where $\texttt{min}(\boldsymbol{k}^s)$ gives the smallest element of vector $\boldsymbol{k}^s$.

5.3.1.3 Systems with k-out-of-n structure and both G and F requirements

This is the case where only k-out-of-n requirements and both positive and negative contributions are involved. The lighting system with light bulbs presented in Figure 5.4 is an example of this case.

In the $\boldsymbol{k}^s$ vector, there are both G (positive) requirements and F (negative) requirements. We use vector $\boldsymbol{k}^{sG}$ to group all the G requirements, and use vector $\boldsymbol{k}^{sF}$ to group all the F requirements. Thus, we have $\boldsymbol{k}^s = (\boldsymbol{k}^{sG}, \boldsymbol{k}^{sF})$.

The updating algorithm is still the same, as presented in Equation (5.1) and (5.2).

The **boundary conditions** are:

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if } \min(\boldsymbol{k}^{sF}) = 0 \tag{5.8}$$

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if } i < \max(\boldsymbol{k}^{sG}) \tag{5.9}$$

$$R(i, \boldsymbol{k}^s) = 1, \quad \texttt{if } i < \min(\boldsymbol{k}^{sF}) \texttt{ AND } \max(\boldsymbol{k}^{sG}) = 0 \tag{5.10}$$

5.3.1.4 Systems with weighted-k-out-of-n:G structure

This is the case where only weighted-k-out-of-n requirements and only positive contributions are involved. Utilities with respect to such a performance index are non-netative integer numbers, instead of just 0 or 1.

The updating algorithm is the same as that in Equation (5.1) and (5.2).

The **boundary conditions** are:

$$R(i, \boldsymbol{k}^s) = 1, \quad \texttt{if } \max(\boldsymbol{k}^s) = 0 \tag{5.11}$$

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if } i = 0 \texttt{ AND } \max(\boldsymbol{k}^s) > 0 \tag{5.12}$$

Because of the ways the intermediate results are saved and reused [4, 97], the algorithms for evaluating weighted-k-out-of-n systems are not as efficient as those for evaluating k-out-of-n systems. Therefore, it is preferable that a requirement is expressed in a k-out-of-n structure.

5.3.1.5 Systems with weighted-k-out-of-n:F structure

This is the case where only weighted-k-out-of-n requirements and only negative contributions are involved. The updating algorithm is still the same, as

presented in Equation (5.1) and (5.2).

The **boundary conditions** are:

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if } \texttt{min}(\boldsymbol{k}^s) = 0 \tag{5.13}$$

$$R(i, \boldsymbol{k}^s) = 1, \quad \texttt{if } i = 0 \tag{5.14}$$

5.3.1.6 Systems with weighted-k-out-of-n structure and both G and F requirements

This is the case where only weighted-k-out-of-n requirements and both positive and negative contributions are involved. We use vector $\boldsymbol{k}^{sWG}$ to group all the G requirements, and use vector $\boldsymbol{k}^{sWF}$ to group all the F requirements. Thus, we have $\boldsymbol{k}^s = (\boldsymbol{k}^{sWG}, \boldsymbol{k}^{sWF})$. The updating algorithm is still the same, as presented in Equation (5.1) and (5.2).

The **boundary conditions** are:

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if } \texttt{min}(\boldsymbol{k}^{sWF}) = 0 \tag{5.15}$$

$$R(i, \boldsymbol{k}^s) = 0, \quad \texttt{if } i = 0 \texttt{ AND } \texttt{max}(\boldsymbol{k}^{sWG}) > 0 \tag{5.16}$$

$$R(i, \boldsymbol{k}^s) = 1, \quad \texttt{if } i = 0 \texttt{ AND } \texttt{max}(\boldsymbol{k}^{sWG}) = 0 \tag{5.17}$$

Certainly there are other special cases. But we will not talk in details about them. The evaluation algorithms for the special cases presented in this section help to derive and understand the general algorithm for evaluating unified k-out-of-n systems, which will be presented in the next section.

5.3.2 General reliability evaluation algorithm for unified k-out-of-n systems

First, we need to group the requirements on different performance indexes. We use vector $\boldsymbol{k}^{sG}$ to group all the k-out-of-n:G requirements, use vector $\boldsymbol{k}^{sF}$ to group all the k-out-of-n:F requirements, use vector $\boldsymbol{k}^{sWG}$ to group all the weighted-k-out-of-n:G requirements, and use vector $\boldsymbol{k}^{sWF}$ to group all the weighted k-out-of-n:F requirements. Thus, we have $\boldsymbol{k}^{s} = (\boldsymbol{k}^{sG}, \boldsymbol{k}^{sWG}, \boldsymbol{k}^{sF}, \boldsymbol{k}^{sWF})$.

The **updating algorithm** is shown in Equation (5.1) and (5.2). And they are copied here as follows:

$$\begin{aligned} R(i, \boldsymbol{k}^{s}) &= \sum_{m=0}^{M} p_{i,m} \cdot R(i-1, \boldsymbol{k}^{sm}) \\ \boldsymbol{k}^{sm} &= \boldsymbol{k}^{s} - \boldsymbol{u}^{im} \quad (k_l^{sm} = 0, \ \texttt{if} \ \ k_l^{sm} < 0) \end{aligned}$$

The **boundary conditions** are:

$$R(i, \boldsymbol{k}^{s}) = 0, \quad \texttt{if} \ \ \texttt{min}((\boldsymbol{k}^{sF}, \boldsymbol{k}^{sWF})) = 0 \tag{5.18}$$

$$R(i, \boldsymbol{k}^{s}) = 0, \quad \texttt{if} \ \ i < \texttt{max}(\boldsymbol{k}^{sG}) \tag{5.19}$$

$$R(i, \boldsymbol{k}^{s}) = 0, \quad \texttt{if} \ \ i = 0 \ \ \texttt{AND} \ \ \texttt{max}((\boldsymbol{k}^{sG}, \boldsymbol{k}^{sWG})) > 0 \tag{5.20}$$

$$R(i, \boldsymbol{k}^{s}) = 1, \quad \texttt{if} \ \ i = 0 \ \ \texttt{AND} \ \ \texttt{max}((\boldsymbol{k}^{sG}, \boldsymbol{k}^{sWG})) = 0 \tag{5.21}$$

It can be seen that the only difference between the boundary conditions above and those for systems with weighted-k-out-of-n structure and both G and F requirements is the second boundary condition in Equation (5.19), which is introduced due to the k-out-of-n:G requirements.

A general comment is that the evaluation processes for weighted-k-out-of-n requirements are not as efficient as those for k-out-of-n requirements. There-

fore, when modeling the system, it is preferable to model the requirements as k-out-of-n requirements.

5.3.3 Modeling and evaluation procedure of a unified k-out-of-n system

The modeling and reliability evaluation procedure for a unified k-out-of-n system is as follows:

(1). Build the **function diagram** of a component, and identify the **system requirements** on the performance indexes.

(2) Analyze the **possible states** of a components.

(3) Obtain the **component state distributions**.

(4) Evaluate **system reliability** using algorithms in Section 5.3.1 and 5.3.2.

5.3.4 Evaluation of component state distribution

This is the third step in the modeling and evaluation procedure for a unified k-out-of-n system.

As stated in Section 5.2.4.1, in the machine tool example shown in Figure 5.1, component state distributions are determined by the links, such as link L1, L2 and L3 in this example. The links describe how the performance indexes (functions and/or failure modes) occur and their dependencies. Each link is binary, and is associated with a probability of its occuring.

The component state distributions are calculated by simply enumerating all the links of in the function diagram of a component. This approach of calculating component state distribution is viable since typically there are not so many performance indexes in a component and the function diagram is not

too complex.

5.4 An example

Consider a machining system with components described in Section 5.2.4.1 and Figure 5.1, where component states as well as system states are defined. This is a system with k-out-of-n:G structure, as discussed in Section 5.3.1.1. Suppose there are 5 such machine tools in the machining system, and any machine is capable of doing two machining functions: turning and milling. In this example, we assume all the components are identically and independently distributed (i.i.d.).

(1) The function diagram is as shown in Figure 5.1. Suppose the requirements are $k_1 = 4$ and $k_2 = 3$, i.e., at least 4 machine tools with turning function are required to meet the turning demand, and at least 3 machine tools with milling function are required to meet the milling demand.

(2) A component has 4 possible states, as defined in Section 5.2.4.1.

(3) Suppose the reliability of the links (probabilities of working) are:

$$p_{L1} = 0.9,\ p_{L2} = 0.9,\ p_{L3} = 0.8.$$

From Figure 5.1, the relationship between the functions and the links can be described using boolean expressions as:

$$P1 = L1 \cdot L2, \quad P2 = L1 \cdot L3, \tag{5.22}$$

where $P1$ and $P2$ represent whether a component has the functions.

The enumerating method is used to determine the component state distribution. There are three links in the function diagram of a machine tool in this

example, and each link is binary. Thus, there are totally $2^3 = 8$ combinations of the links. For each combination, we can calculate the probability of the combination occuring, and the correponding component state. For example, let's consider the combination of links as follows: $L1 = 1$, $L2 = 0$ and $L3 = 1$. The combination of this combination occuring is

$$p_{L1} * (1 - p_{L2}) * p_{L3} = 0.9 * (1 - 0.9) * 0.8 = 0.072.$$

Based on the Equation (5.22), $P1 = 0$ and $P2 = 1$. That is, the machine tool can not perform the turning function but it can perform the milling function, and it is in state 2. Summing up the probabilities of all the combinations of links that make the component in state 2, we can obtain the probability of the component in state 2. And we can obtain the probabilities of the component in state 0, state 1 and state 3 in the same way. Eventually, we get the component state distributions as follows:

$$\boldsymbol{p}_i = (0.1180,\ 0.1620,\ 0.0720,\ 0.6480) \tag{5.23}$$

where the elements represent the probabilities of a component in state 0, 1 ,2 and 3, respectively.

(4) Since this system includes only k-out-of-n:G requirements, we can directly use the algorithm for this special case in Section 5.3.1.1, i.e., Equations (5.1), (5.2), (5.3), (5.4).

To evaluate the reliability of the system with respect to system state 3, or the probability of the system in system state 3, we use recursive function $R(5, \boldsymbol{k}^3)$, where the number "5" represents the number of components and $\boldsymbol{k}^3 = (k_1, k_2) = (4, 3)$. Using the proposed algorithm, we got $R(5, \boldsymbol{k}^3) = 0.6873$.

To evaluate the reliability of the system with respect to system state 2, or the probability of the system in system state 3 and state 2, we use recursive function $R(5, \boldsymbol{k}^2)$, where $\boldsymbol{k}^2 = (0, k_2) = (0, 3)$. Using the proposed algorithm, we got $R(5, \boldsymbol{k}^2) = 0.8624$. Thus the probability of the system in system state 2 is

$$R(5, \boldsymbol{k}^2) - R(5, \boldsymbol{k}^3) = 0.8624 - 0.6873 = 0.1751.$$

Similarly, to evaluate the reliability of the system with respect to system state 1, or the probability of the system in system state 3 and state 1, we use recursive function $R(5, \boldsymbol{k}^1)$, where $\boldsymbol{k}^1 = (k_1, 0) = (4, 0)$. Using the proposed algorithm, we got $R(5, \boldsymbol{k}^1) = 0.7576$. Thus the probability of the system in system state 1 is

$$R(5, \boldsymbol{k}^1) - R(5, \boldsymbol{k}^3) = 0.7576 - 0.6873 = 0.0703.$$

Finally, the probability of the system in state 0 is

$$1 - 0.6873 - 0.1751 - 0.0703 = 0.0673.$$

Summarizing the results above, we got the probabilities of the system in state 0, 1, 2, 3 are 0.0673, 0.0703, 0.1751, 0.6873, respectively. We also used enumerating method to verify the result, and the same result was obtained.

The computation time with respect to different number of components is also investigated for the probability of the system in system state 3. The cases with up to 50 components are investigated, and the results are listed in Table 5.2. The computation time does not go up dramatically with n, which illustrates the efficiency of the proposed algorithm.

Table 5.2: Computation time (seconds) with respect to the number of components

n	5	10	15	20	30	50
time (seconds)	0.02	0.08	0.15	0.24	0.27	0.41

5.5 Concluding remarks

It is a critical issue to evaluate the reliability of a complex system with a large number of components. Since it is very hard to develop an efficient method that be used for reliability evaluation of systems with any type of structure, it is thus very important to develop efficient reliability evaluation methods for systems with special structures. In this chapter, we have focused on systems with k-out-of-n structures.

The common characteristics of all the reported k-out-of-n systems in the literature are as follows: (1) a constituent component has some performance indexes (functions and/or failure modes); (2) for each of a group of selected performance indexes, the system has certain requirement on the combined effect of the components; and (3) the topological positions of the components do not matter. Based on these common characteristics, we have proposed the unified k-out-of-n system model, which is significant in that: (1) it provides a general model for k-out-of-n systems. All the reported binary k-out-of-n models, multi-state k-out-of-n models and weighted k-out-of-n models, are special cases of the unified k-out-of-n model; (2) it can deal with problems that current models are not able to deal with: a component has multiple performance indexes, and these performance indexes are dependent.

We have provided some simple examples in this chapter to illustrate the possible applications of the unified k-out-of-n system model. Complex real

world applications are yet to be identified and applied to.

Chapter 6

Reliability Evaluation of Multi-State Two-terminal Networks

6.1 Introduction

In Chapter 3, 4 and 5, we have discussed the reliability evaluation of multi-state systems with k-out-of-n structures. In this chapter, we discuss the reliability evaluation of another type of multi-state systems, the multi-state network systems. And our discussion will be focused on the reliability evaluation of multi-state two-terminal networks given all minimal path vectors. The materials in this chapter have been published in [106].

Many network systems, such as power generation and transmission systems, oil and gas supply systems, and communication systems, consist of components which can work at different levels of capacity. These systems are regarded as multi-state networks [46, 54, 55]. Reliability is an important index for evaluating the performance of these systems and for making decisions such as maintenance scheduling. We consider a network which satisfies the following assumptions: (1) all nodes are perfect, and (2) all links are directed and failure prone. In reliability evaluation of binary two-terminal networks, if the nodes

are not perfect, the node failure probabilities can be allocated to the links, and thus the network can be transformed into a network with perfect nodes. In reliability analysis of multi-state two-terminal networks discussed in this chapter, we assume that all nodes are perfect and study this simpler case first. Hopefully, results out of this work can be extended to deal with multi-state two-terminal networks with imperfect nodes, which might be transformed to multi-state two-terminal networks with perfect nodes.

The capacity of a link is an independent discrete random variable which may take non-negative integer values following a certain probability distribution. The term "components", which is usually used in multi-state system analysis [46, 55], are used to refer to these failure prone "links".

We limit our discussions to two-terminal reliability analysis. This is a classical network reliability problem with a broad range of practical applications, such as transportation networks and telecommunication networks [43, 79]. We are interested in the flow capacity from a single source node, s, to a single sink node, t. Under these assumptions, we can call the flow capacity from node s to node t the capacity or the state of the system, represented by $\phi(\mathbf{x})$. For such a network, we are interested in evaluating the probability that the system state is equal to or greater than d units, i.e. $\phi(\mathbf{x}) \geq d$, which can also be considered to be the reliability of the network once d is specified.

A general method for the multi-state network reliability evaluation is using minimal path (cut) vectors, as defined in Section 2.1.5. A minimal path vector to level d is called a d-MP for short. For the purpose of evaluating the probability of event $\phi(\mathbf{x}) \geq d$, where $\mathbf{x}$ denotes a component state vector, Xue reported an algorithm for generating all d-MPs [100]. Lin *et al.* proposed another algorithm for the same purpose, claiming it to be more efficient [54].

Using either algorithm, one can find all d-MPs. Let's suppose that there are L such d-MPs and, for simplicity, denote them as $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^L$. Then, the probability that $\phi(\mathbf{x}) \geq d$ can be calculated as follows:

$$\Pr(\phi(\mathbf{x}) \geq d) = \Pr\left(\{\mathbf{x} \geq \mathbf{z}^1\} \cup \{\mathbf{x} \geq \mathbf{z}^2\} \cup \cdots \cup \{\mathbf{x} \geq \mathbf{z}^L\}\right). \tag{6.1}$$

Given that all d-MPs have been found, the issue becomes how to evaluate the probability of the union of the events that the component state vector is greater than or equal to at least one of the d-MPs, as shown in Equation (6.1). Hudson and Kapur presented methods using the Inclusion-Exclusion (IE) principle and the Sum of Disjoint Products (SDP) principle to evaluate system reliability and reliability bounds for multi-state systems given all minimal path vectors or minimal cut vectors [39, 40, 41]. However, these methods are not systematic and not efficient. Aven proposed an algorithm based on state-space decomposition, which provides a systematic way of evaluating the union probability no matter how many d-MPs there are [1]. It has been proved to be much more efficient than the IE method [1], except for the situation when the number of d-MPs is much smaller than the number of components, which exists in very few real systems.

In this chapter, we propose an efficient recursive algorithm for evaluation of multi-state network reliability based on the SDP principle, and name it Recursive Sum of Disjoint Products (RSDP) algorithm. The basic idea is that, based on the SDP principle and a specially defined "maximum" operator, "$\oplus$", the probability of a union with L vectors can be calculated via calculating the probabilities of several unions with $L-1$ vectors or less. The correctness and efficiency of this algorithm will be investigated.

Acronyms:

SDP: Sum of Disjoint Products

RSDP: Recursive Sum of Disjoint Products

IE: Inclusion-Exclusion

MP: minimal path vector

d-MP: minimal path vector to system state d

Notation:

n: the number of components in the network

M_i: an integer value representing the maximum state or maximum capacity of component i, $M_i \geq 1$, $i = 1, 2, \ldots, N$

s: the source node

t: the sink node

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

x_i: a discrete random variable representing the state or the capacity of component i, x_i may take values 0, 1, 2, ..., M_i, $i = 1, 2, \ldots, N$

$\mathbf{x}$: $= (x_1, x_2, \ldots, x_N)$. We call $\mathbf{x}$ the component state vector. It represents the states of all failure prone components (i.e., all components).

$\phi(\mathbf{x})$: state of the system

p_{ij}: $= \Pr(x_i = j)$, $i = 1, 2, \ldots, N$, $j = 0, 1, 2, \ldots, M_i$. $\sum_{j=0}^{M_i} p_{ij} = 1$

P_{ij}: $= \Pr(x_i \geq j)$, $i = 1, 2, \ldots, N$, $j = 0, 1, 2, \ldots, M_i$.

$\mathbf{z}^i$: the ith minimal path vector of the considered multi-state network.

$\mathbf{y}^i$: a general vector with n elements and with index i.

$\mathbf{Y}^{j,i}$: a vector generated by the "$\oplus$" operator, $\mathbf{Y}^{j,i} = \mathbf{y}^j \oplus \mathbf{y}^i$

PrU(•): the recursive function of the RSDP algorithm

TM$_i$: the ith term in SDP calculation.

T_1: The CPU time by Aven's algorithm.

T_2: The CPU time by RSDP.

λ: the ratio $\frac{T_1}{T_2}$.

6.2 Some definitions

Some definitions that are to be used in the proposed RSDP algorithm are presented in this section.

Definition 6.13 *Event* $\{\boldsymbol{x} \geq \boldsymbol{y}\}$ *means* $x_i \geq y_i$ *for all* i, *where* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *are vectors with the same length.*

Consider a multi-state network with n independent components. Component i $(1 \leq i \leq n)$ has $M_i + 1$ discrete and mutually exclusive states 0, 1, ..., M_i. For a certain vector $\mathbf{y}$ in which y_i represents the state of component i, we have

$$\mathtt{Pr}(\mathbf{x} \geq \mathbf{y}) = \prod_{i=1}^{n} \mathtt{Pr}(x_i \geq y_i) = \prod_{i=1}^{n} P_{i,y_i}, \tag{6.2}$$

where P_{i,y_i} is the probability of component i in state y_i or above, $0 \leq y_i \leq M_i$. That is

$$P_{i,y_i} = \sum_{j=y_i}^{M_i} p_{i,j},$$

where M_i represents the highest state of component i, and $p_{i,j}$ is the probability of component i in state j.

Definition 6.14 *Event* $\{\boldsymbol{x} \not\geq \boldsymbol{y}\}$ *is the complement of event* $\{\boldsymbol{x} \geq \boldsymbol{y}\}$.

Definition 6.15 *A special "maximum" operator, "$\oplus$", is defined as:*

$$\boldsymbol{y}^1 \oplus \boldsymbol{y}^2 \equiv \left(max(y_j^1, y_j^2)\right), \quad 1 \leq j \leq n. \tag{6.3}$$

For example, if $\mathbf{y}^1 = (1,2,3,4)$, and $\mathbf{y}^2 = (4,3,2,1)$, we will have $\mathbf{y}^1 \oplus \mathbf{y}^2 = (4,3,3,4)$. The "$\oplus$" operator defined in equation (6.3) is designed to manipulate those d-MPs, and it plays an important role in the proposed RSDP algorithm.

6.3 Recursive sum of disjoint products algorithm

6.3.1 The proposed RSDP algorithm

In this section, we will propose a recursive algorithm based on the SDP principle for the evaluation of multi-state network reliability. We call this proposed algorithm Recursive sum of disjoint products (RSDP) algorithm.

Suppose we have three minimal path vectors $\mathbf{z}^1$, $\mathbf{z}^2$ and $\mathbf{z}^3$. From the SDP principle [46, 39], we have

$$\begin{aligned} \Pr\left(\{\mathbf{x} \geq \mathbf{z}^1\} \cup \{\mathbf{x} \geq \mathbf{z}^2\} \cup \{\mathbf{x} \geq \mathbf{z}^3\}\right) &= \Pr\left(\mathbf{x} \geq \mathbf{z}^1\right) + \Pr\left(\{\mathbf{x} \not\geq \mathbf{z}^1\}\{\mathbf{x} \geq \mathbf{z}^2\}\right) \\ &+ \Pr\left(\{\mathbf{x} \not\geq \mathbf{z}^1\}\{\mathbf{x} \not\geq \mathbf{z}^2\}\{\mathbf{x} \geq \mathbf{z}^3\}\right) \end{aligned} \tag{6.4}$$

The first term can be calculated directly using equation (6.2). The second term is the probability that events $\{\mathbf{x} \geq \mathbf{z}^2\}$ and $\{\mathbf{x} \not\geq \mathbf{z}^1\}$ occur simultaneously. To evaluate the second term, we can use the following basic probability formula

$$\Pr(\bar{A}B) = \Pr(B) - \Pr(AB), \tag{6.5}$$

where A and B are two arbitrary events. Applying equation (6.5), the second term in equation (6.4) can be written as

$$\begin{aligned} \texttt{Term2} &= \texttt{Pr}\left(\overline{\mathbf{x} \geq \mathbf{z}^1}\{\mathbf{x} \geq \mathbf{z}^2\}\right) \\ &= \texttt{Pr}\left(\mathbf{x} \geq \mathbf{z}^2\right) - \texttt{Pr}\left(\{\mathbf{x} \geq \mathbf{z}^1\}\{\mathbf{x} \geq \mathbf{z}^2\}\right) \\ &= \texttt{Pr}\left(\mathbf{x} \geq \mathbf{z}^2\right) - \texttt{Pr}\left(\mathbf{x} \geq (\mathbf{z}^1 \oplus \mathbf{z}^2)\right) \\ &= \texttt{Pr}\left(\mathbf{x} \geq \mathbf{z}^2\right) - \texttt{Pr}\left(\mathbf{x} \geq \mathbf{Y}^{1,2}\right) \end{aligned} \tag{6.6}$$

where $\mathbf{Y}^{1,2} = \mathbf{z}^1 \oplus \mathbf{z}^2$. As shown in equation (6.6), the probability of an event involving two vectors, $\mathbf{z}^1$ and $\mathbf{z}^2$, is calculated via the probability of an event involving only one vector, $\mathbf{z}^2$, and the probability of another event involving only one vector, $\mathbf{Y}^{1,2}$.

The third term in Equation (6.4) is the probability that events $\{\mathbf{x} \geq \mathbf{z}^3\}$, $\{\mathbf{x} \not\geq \mathbf{z}^1\}$ and $\{\mathbf{x} \not\geq \mathbf{z}^2\}$ occur simultaneously. Let $\mathbf{Y}^{1,3} = \mathbf{z}^1 \oplus \mathbf{z}^3$, and $\mathbf{Y}^{2,3} = \mathbf{z}^2 \oplus \mathbf{z}^3$. Similarly, we can find that the third term "Term3" can be represented as

$$\texttt{Term3} = \texttt{Pr}\left(\mathbf{x} \geq \mathbf{z}^3\right) - \texttt{Pr}\left(\{\mathbf{x} \geq \mathbf{Y}^{1,3}\} \bigcup \{\mathbf{x} \geq \mathbf{Y}^{2,3}\}\right) \tag{6.7}$$

As can be seen, the third term can be calculated via the probability of a union of two events. From this specific example, we find that a recursive algorithm based on the SDP principle is viable.

Now we will consider the general case. Suppose we have L general vectors $\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^L$. We define the **recursive function** as

$$\texttt{PrU}(\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^L) \equiv \texttt{Pr}\left(\bigcup_{i=1}^{L} \{\mathbf{x} \geq \mathbf{y}^i\}\right) \tag{6.8}$$

From the SDP principle, we develop the following **recursive algorithm**:

$$\texttt{PrU}(\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^L) = \sum_{i=1}^{L} \texttt{TM}_i, \tag{6.9}$$

where $\texttt{TM}_i$ is the ith term in the SDP calculation.

$$\begin{aligned} \texttt{TM}_1 &= \texttt{Pr}\left(\mathbf{x} \geq \mathbf{y}^1\right) && (6.10) \\ \texttt{TM}_i &= \texttt{Pr}\left(\mathbf{x} \geq \mathbf{y}^i\right) - \texttt{Pr}\left(\bigcup_{j=1}^{i-1} \{\mathbf{x} \geq \mathbf{Y}^{j,i}\}\right) \\ &= \texttt{Pr}\left(\mathbf{x} \geq \mathbf{y}^i\right) - \texttt{PrU}(\mathbf{Y}^{1,i}, \ldots, \mathbf{Y}^{i-1,i}), \quad \texttt{for } i \geq 2, && (6.11) \end{aligned}$$

where vector $\mathbf{Y}^{j,i} = \mathbf{y}^j \oplus \mathbf{y}^i$. The length of $\mathbf{Y}^{j,i}$ is the same as $\mathbf{y}^j$ and $\mathbf{y}^i$, and the value of an element of $\mathbf{Y}^{j,i}$ refers to the corresponding state of the corresponding component. Therefore, from Equations (6.9), (6.10) and (6.11), the $\texttt{PrU}(\bullet)$ function with L input vectors can be calculated via $\texttt{PrU}(\bullet)$ functions with $L-1$ input vectors or fewer.

The **boundary condition** is $L = 1$. In this case

$$\texttt{PrU}(\bullet) = TM_1 = \texttt{Pr}\left(\mathbf{x} \geq \mathbf{y}^1\right). \tag{6.12}$$

The simplifying procedure is used in RSDP to reduce whenever possible the number of input vectors in function $\texttt{PrU}(\bullet)$. That is, for any input vector, $\mathbf{y}^i$, if there is an input vector $\mathbf{y}^j$ $(j \neq i)$ satisfying $\mathbf{y}^i \geq \mathbf{y}^j$, $\mathbf{y}^i$ will be deleted from the set of input vectors. The reason is, under the assumption that $\mathbf{y}^i \geq \mathbf{y}^j$, we have $(\mathbf{x} \geq \mathbf{y}^i) \cup (\mathbf{x} \geq \mathbf{y}^j) = (\mathbf{x} \geq \mathbf{y}^j)$.

For a multi-state network with L d-MPs $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^L$, its reliability with

Table 6.1: State distributions of the components in the example network

State	0	1	2	3
Component 1	0.05	0.10	0.25	0.60
Component 2	0.10	0.30	0.60	-
Component 3	0.10	0.90	-	-
Component 4	0.10	0.90	-	-
Component 5	0.10	0.90	-	-
Component 6	0.05	0.25	0.70	-

respect to level d is

$$\Pr(\phi(\mathbf{x}) \geq d) = \mathtt{PrU}(\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^L).$$

6.3.2 An illustrative example

The detailed implementation of RSDP will be illustrated in this section, using the example given by Lin *et al.* [54]. The network system under investigation is a bridge network as shown in Fig. 6.1. This network has 6 components (links) represented by C1, C2, ..., and C6, respectively. The components have different numbers of possible states [54]. The state distributions of the components are listed in Table 6.1.

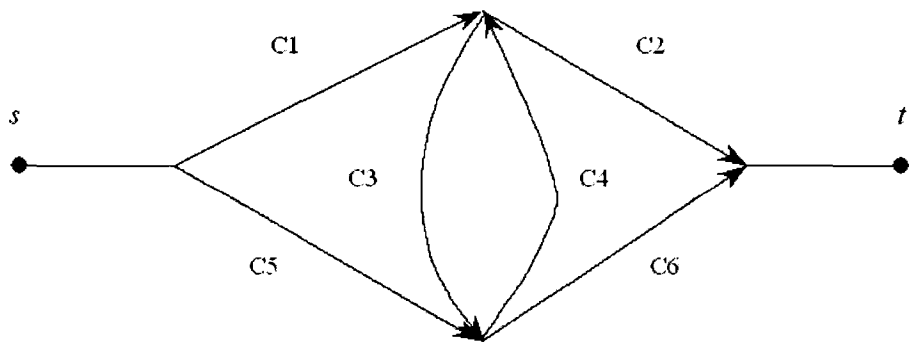

Figure 6.1: A multi-state network with 6 components

There are three 3-MPs:

$$\begin{aligned} \mathbf{z}^1 &= (3,\ 2,\ 1,\ 0,\ 0,\ 1) \\ \mathbf{z}^2 &= (2,\ 2,\ 0,\ 0,\ 1,\ 1) \\ \mathbf{z}^3 &= (2,\ 1,\ 1,\ 0,\ 1,\ 2). \end{aligned} \tag{6.13}$$

The proposed RSDP algorithm is used to calculate $\mathtt{Pr}(\phi(\mathbf{x}) \geq 3)$. From equation (6.9) of the RSDP algorithm, there will be three terms in this problem:

$$\begin{aligned} \mathtt{Pr}(\phi(\mathbf{x}) \geq 3) &= \Pr\left(\{\mathbf{x} \geq \mathbf{z}^1\} \cup \{\mathbf{x} \geq \mathbf{z}^2\} \cup \{\mathbf{x} \geq \mathbf{z}^3\}\right) \\ &= \mathtt{TM}_1 + \mathtt{TM}_2 + \mathtt{TM}_3. \end{aligned} \tag{6.14}$$

(1)

$$\begin{aligned} \mathtt{TM}_1 &= \mathtt{Pr}\left(\mathbf{x} \geq \mathbf{z}^1\right) \\ &= (0.6) \cdot (0.6) \cdot (0.9) \cdot (0.1 + 0.9) \cdot (0.1 + 0.9) \cdot (0.25 + 0.7) \\ &= 0.3078. \end{aligned}$$

(2) From equation (6.11), we have

$$\mathtt{TM}_2 = \mathtt{Pr}\left(\mathbf{x} \geq \mathbf{z}^2\right) - \mathtt{Pr}\left(\mathbf{x} \geq \mathbf{Y}^{1,2}\right), \tag{6.15}$$

where

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

Thus, we get $\mathtt{TM}_2 = 0.1590$.

(3) From equation (6.11)

$$\mathtt{TM}_3 = \mathtt{Pr}\left(\mathbf{x} \geq \mathbf{z}^3\right) - \mathtt{Pr}\left(\{\mathbf{x} \geq \mathbf{Y}^{1,3}\} \bigcup \{\mathbf{x} \geq \mathbf{Y}^{2,3}\}\right) \quad (6.16)$$

where

$$\begin{aligned} \mathbf{Y}^{1,3} &= \mathbf{z}^1 \oplus \mathbf{z}^3 = (3,\ 2,\ 1,\ 0,\ 1,\ 2), \textit{and} \\ \mathbf{Y}^{2,3} &= \mathbf{z}^2 \oplus \mathbf{z}^3 = (2,\ 2,\ 1,\ 0,\ 1,\ 2). \end{aligned}$$

Since $\mathbf{Y}^{1,3} \geq \mathbf{Y}^{2,3}$, based on "the simplifying procedure", $\mathbf{Y}^{1,3}$ will be removed. Thus we have

$$\mathtt{TM}_3 = \mathtt{Pr}\left(\mathbf{x} \geq \mathbf{y}^3\right) - \mathtt{Pr}\left(\mathbf{x} \geq \mathbf{Y}^{2,3}\right) = 0.1446. \quad (6.17)$$

Eventually we have

$$\mathtt{Pr}(\phi(\mathbf{x}) \geq 3) = \mathtt{TM}_1 + \mathtt{TM}_2 + \mathtt{TM}_3 = 0.6114.$$

This result agrees with the result in Lin *et al.* [54], and this has illustrated the correctness of the proposed RSDP algorithm.

In this example, the RSDP approach evaluates 5 probability terms. If the IE procedure is used, we need to evaluate 7 probability terms. The advantage of the proposed RSDP procedure over the IE principle is clear from the following:

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

6.4 Efficiency investigation of RSDP

Aven's algorithm [1] has been recognized as an efficient reliability evaluation algorithm for multi-state systems compared to conventional methods such as the IE method. The computation time with the IE method increases exponentially as the number of MPs, L, increases [1]. In this section, we will compare the efficiency of the proposed RSDP algorithm with that of Aven's algorithm. We wrote Aven's algorithm by following the procedure in Aven's paper [1] and the FORTRAN program in its appendix. The programs of both RSDP and Aven's algorithm have been developed with MATLAB 6.5, and were implemented on a computer with Pentium M 1.7GHz CPU and 512MB of RAM.

In terms of the efficiency of the two algorithms, we are interested in the required computation time with respect to different numbers of components, n, and different numbers of MPs, L. First we consider a hypothetical multi-state network system with 10 components. Each component has 10 states, from 0 to 9; and the state distributions of all the components are set to be the same. Specifically, the state distribution vector, $\boldsymbol{p}$, is set to be

$$\boldsymbol{p} = (0.05,\ 0.15,\ 0.1,\ 0.05,\ 0.15,\ 0.05,\ 0.15,\ 0.1,\ 0.05,\ 0.15)\,.$$

We randomly generate 50 vectors, $\mathbf{z}^1$ to $\mathbf{z}^{50}$, indicating the components' states, and ensure that there are no two vectors $\mathbf{z}^i$ and $\mathbf{z}^j$ satisfying $\mathbf{z}^i \geq \mathbf{z}^j$. That is, no vector is dominated by other vectors in this group [35]. As a result, these 50 vectors can be treated as all the MPs to a hypothetical system state, say state d. Thus the probability of the component state vector being not less than any of these hypothetical MPs can be calculated using RSDP or Aven's algorithm. First, assume that only the first 5 MPs, $\mathbf{z}^1$ to $\mathbf{z}^5$, are available.

RSDP and Aven's algorithm are used respectively to evaluate the probability of interest. Then we change L, the number of MPs, by assuming only the first 10, 15, 20, 25 , 30, 40 or 50 MPs are available, respectively, and calculate the same probability of interest. In all these cases, RSDP and Aven's algorithm lead to the same results, which further shows the correctness of RSDP.

We need to note that the 50 MPs used above are generated randomly. The computation time using RSDP or Aven's algorithm may be different if we use another group of 50 MPs. To make more sense in comparing the efficiency of RSDP and Aven's algorithm, we randomly generate 10 groups of MP vectors, with 50 MPs in each group. The average CPU time using these 10 groups of MP vectors is used to represent the CPU time with respect to a certain L. The CPU time (in seconds) using RSDP or Aven's algorithm is listed in Table 6.2, where T_1 and T_2 represent the CPU time by Aven's algorithm and by RSDP, respectively. Let $\lambda = \frac{T_1}{T_2}$ denote the ratio between the CPU time of Aven's algorithm and that of RSDP. That is, λ represents the advantage of RSDP over Aven's algorithm in terms of CPU time. If $0 < \lambda < 1$, RSDP is slower than Aven's algorithm; if $\lambda > 1$, RSDP is faster than Aven's algorithm. The bigger the λ is, the more advantageous RSDP is over Aven's algorithm. From Table 6.2, it appears that the computation time does not increase exponentially as L increases for either RSDP or Aven's algorithm. In this specific network system with 10 components, RSDP is a little bit faster when there are 15 MPs or less, while Aven's algorithm is faster when there are 20 MPs or more. From the trend of λ, Aven's algorithm becomes more advantageous with the increase of the number of MP vectors (L) for the network with 10 components.

Next, we investigate a multi-state network with only 5 components. The other settings, such as the number of states and the state distribution for each

component, remain the same as in the case of a system with 10 components. Similarly, we generate 10 groups of MP vectors with 50 MPs in each group, and investigate the efficiency of RSDP and Aven's algorithm. The results are shown in Table 6.3. We find that for the network with 5 components, Aven's algorithm is faster than RSDP. There is no clear trend for λ with respect to the number of d-MPs. We also conclude that for networks with a small number of components (less than 10 in this example), Aven's algorithm is more efficient than RSDP.

Now, we will investigate multi-state networks with more than 10 components. First, we consider a multi-state network with 15 components. All the other settings remain the same as in the case of system with 10 components. The results are shown in Table 6.4. We have also investigated the case of multi-state networks with 20, 30, and 40 components respectively in the same way, and the results are listed in Table 6.5, 6.6 and 6.7. We did not do the case for Aven's algorithm when there are 40 components and 20 d-MPs, because in this case the expected CPU time using Aven's algorithm is around 40 hours and thus determining the average CPU time would take quite a few days.

In the case of 15 components in Table 6.4, RSDP is faster than Aven's algorithm with respect to all L values investigated. From the trend of λ, the advantage of RSDP decreases with the increase of L; however, this trend does

Table 6.2: Efficiency comparison when the system has 10 components

Number of d-MPs (L)	5	10	15	20	25	30	40	50
CPU time by Aven's (T_1)	0.03	0.22	0.44	0.69	1.20	2.15	9.26	15.71
CPU time by RSDP (T_2)	0.01	0.09	0.29	0.83	1.62	2.74	16.475	23.95
Ratio $\lambda = \frac{T_1}{T_2}$	2.60	2.52	1.54	0.83	0.74	0.78	0.56	0.66

Table 6.3: Efficiency comparison when the system has 5 components

Number of d-MPs (L)	5	10	15	20	25	30	40	50
CPU time by Aven's (T_1)	0.008	0.023	0.055	0.085	0.13	0.20	0.25	0.29
CPU time by RSDP (T_2)	0.008	0.04	0.10	0.21	0.27	0.35	0.47	0.69
Ratio $\lambda = \frac{T_1}{T_2}$	1.00	0.56	0.55	0.40	0.48	0.56	0.53	0.41

Table 6.4: Efficiency comparison when the system has 15 components

Number of d-MPs (L)	5	10	15	20	25	30
CPU time by Aven's (T_1)	0.11	0.67	2.37	4.97	17.22	35.70
CPU time by RSDP (T_2)	0.01	0.10	0.35	1.39	6.40	17.95
Ratio $\lambda = \frac{T_1}{T_2}$	11.00	6.70	6.76	3.58	2.69	1.99

not exist in the cases of 20, 30 and 40 components, as shown in Table 6.5, 6.6 and 6.7. In fact, RSDP is much more efficient than Aven's algorithm in these cases. Specifically, RSDP is about 20 times faster than Aven's algorithm in the case of the system with 20 components, and hundreds or thousands of times faster than Aven's algorithm in the cases of the systems with 30 and 40 components. Thus, it seems that the efficiency of Aven's algorithm is much more sensitive to the number of components than that of RSDP. We can conclude that RSDP is more efficient than Aven's algorithm when the number of components of a system is not too small (say, when there are more than 15 components in the system), and the advantage of RSDP becomes stronger with the increase of the number of components in the system.

In Figure 6.2, we present ratio λ with respect to different number of components and different number of minimal path vectors L in a graphical manner. The vertical axis represents the ratio λ in the logarithm format. Again, it is very obvious that the advantage of RSDP over Aven's method becomes more

significant with the increase of the number of components in the system.

Table 6.5: Efficiency comparison when the system has 20 components

Number of d-MPs (L)	5	8	10	15	20
CPU time by Aven's (T_1)	0.31	0.97	3.27	33.55	114.53
CPU time by RSDP (T_2)	0.01	0.06	0.18	0.95	6.42
Ratio $\lambda = \frac{T_1}{T_2}$	31.27	16.17	18.17	35.44	17.84

Table 6.6: Efficiency comparison when the system has 30 components

Number of d-MPs (L)	5	8	10	15	20
CPU time by Aven's (T_1)	0.93	17.28	75.49	992	15,646
CPU time by RSDP (T_2)	0.01	0.06	0.19	1.92	20.00
Ratio $\lambda = \frac{T_1}{T_2}$	93.00	288.00	397.29	518.02	782.30

Table 6.7: Efficiency comparison when the system has 40 components

Number of d-MPs (L)	5	8	10	15	20
CPU time by Aven's (T_1)	2.58	80.2	670	8,698	-
CPU time by RSDP (T_2)	0.01	0.06	0.22	2.25	37.80
Ratio $\lambda = \frac{T_1}{T_2}$	258.00	1,336.67	3,045.45	3,865.96	-

6.5 Concluding remarks

In this chapter, we developed the RSDP algorithm, an efficient recursive algorithm based on the SDP principle for reliability evaluation of multi-state two-terminal networks given all the minimal path vectors. Based on the SDP principle and a specially defined "maximum" operator, "$\oplus$", RSDP can calculate the probability of a union with L vectors via calculating the probabilities of several unions with $L-1$ vectors or fewer. The implementation of RSDP

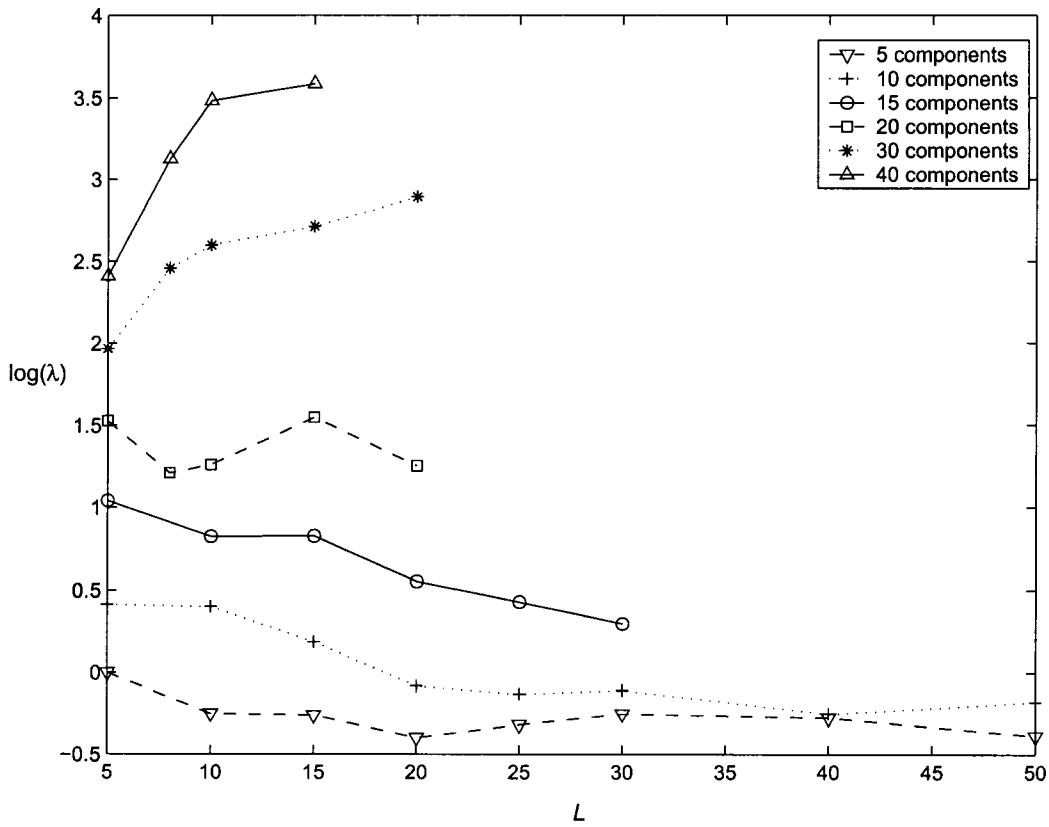

Figure 6.2: Ratio λ with respect to different number of components and different L

has been illustrated using a simple example. The efficiency of this algorithm has been investigated by comparing it with Aven's algorithm [1] which is recognized as efficient. It has been found that RSDP is more efficient than Aven's algorithm when the number of components of a system is not too small (say, greater than 15), and it can efficiently deal with the reliability evaluation of complex systems with a large number of components. RSDP provides us with an efficient, systematic and simple approach for evaluating multi-state network reliability when all d-MPs are given.

The RSDP algorithm might be extended to develop more efficient methods for reliability evaluation of more complex multi-state network systems, such as multi-state all-terminal networks.

From Chapter 3 to Chapter 6, we have discussed reliability modeling and evaluation of multi-state systems, focusing on multi-state k-out-of-n systems and multi-state two-terminal networks. It is important to efficiently and accurately assess system reliability. On top of system reliability analysis, we can do optimal reliability design to improve system reliability. In next chapter, we will discuss how to improve reliability and cost measures of multi-state systems through optimal reliability design.

Chapter 7

Optimal Design of Multi-State Series-parallel Systems

In this chapter, we will discuss how to improve system reliability and cost effectiveness through optimal design. We focus on the optimal design of multi-state series-parallel systems, which has been intensively studied in the literature.

In the first section of this chapter, we focus on how to deal with multiple conflicting design objectives, such as system reliability and system cost, which are typically involved in reliability based optimization of multi-state systems. An approach based on physical programming and genetic algorithms is proposed in this chapter for this purpose. The materials in the first section of this chapter has been documented in paper [88].

In the second section of this chapter, we will investigate how to improve the optimal design of multi-state series-parallel systems by extending the redundancy optimization of multi-state series-parallel systems to the joint reliability-redundancy optimization of multi-state series-parallel systems. That is, in addition to redundancies, component state distributions are treated as design variables as well. The materials in the second section of this chapter has been documented in paper [91].

7.1 Optimal system design considering multiple design objectives

In practical situations involving reliability optimization, there often exist mutually conflicting goals such as maximizing system utility and minimizing system cost and weight [45]. Current multi-state optimization methods typically treat one goal as the objective function of the optimization model and the other goals as constraints. Levitin et al [51] studied the redundancy optimization problem for series-parallel multi-state systems, and the model they used aimed at minimizing the system cost under the constraint of system availability. Levitin and Lisnianski [50] presented a structure optimization approach for multi-state systems with two failure modes. They proposed two optimization problem formulations: one aims at maximizing system availability and the other aims at providing the maximal proximity of the expected system performance to the desired levels for both the open mode and the close mode, while satisfying the system cost and reliability requirements. Ramirez-Marquez and Coit [78] proposed a heuristic approach for solving the redundancy allocation for multi-state series-parallel systems. Liu et al [56] presented a neural network approximation approach for optimal design of continuous-state parallel-series systems, so as to reduce the computational complexity in the utility evaluation of continuous-state systems. Their model was built to maximize system utility subject to system cost constraints.

However, it is very difficult to specify in advance the constraint values for the goals used as constraints. After a solution is obtained, we often need to modify these constraint values to find a better tradeoff between goals such as system utility and system cost. But finding the most appropriate constraint values is a trial and error process, it is time-consuming, and there is no clear guidance as to how to converge to the right set of constraint values. Partic-

ularly, when there are many constraint values that need to be specified, it is almost impossible to find the most appropriate values for these constraints.

There has been intensive research on redundancy allocation of binary systems considering multiple objectives. Sakawa [83] presented a multi-objective formulation to maximize reliability and minimize cost for system structure optimization by using the surrogate worth trade-off method. Inagaki et al [42] proposed to maximize reliability and minimize cost and weight by using an interactive optimization method. Park [72] used fuzzy logic theory to analyze a multi-objective reliability apportionment problem for a two-component series system. Dhingra [15] and Rao and Dhingra [76] studied the reliability and redundancy allocation problem for a four-stage and a five-stage over-speed protection system, using crisp and fuzzy multi-objective optimization approaches, respectively. Huang [29] proposed an approach for fuzzy multi-objective optimization decision-making involving a series reliability system. Ravi et al [77] modeled the problem of optimizing the reliability of a complex system as a fuzzy multi-objective optimization problem [30]. Huang et al [31] applied a proposed interactive physical programming approach to reliability-redundancy optimization of a binary-state system. However, this approach is too complicated to be used conveniently in practical problems, and the improvement on the solution's quality is trivial compared to using the original physical programming approach. A sophisticated intelligent interactive multi-objective optimization method was later used adopted for the reliability optimization of binary systems [32]. Taboada *et al.* proposed two practical approaches for multi-objective optimization with the application to system reliability optimization [87].

In this section, we present a multi-objective optimization model for redun-

dancy allocation for multi-state series-parallel systems, which optimizes the goals of system utility, system cost and system weight simultaneously. The physical programming method is used as an effective approach to optimize the system structure within this model's framework. Physical Programming eliminates the typical iterative process involving the adjustment of the physically meaningless weights, which is required by virtually all other multi-objective optimization methods, and thus substantially reduces the computational intensities. The DMs' preferences are specified individually on each goal through physically meaningful values, which makes the physical programming method easy to use and advantageous in dealing with a large number of objectives. Physical programming has proved its effectiveness in addressing a wide array of multi-objective problem [61, 62, 63, 64, 74]. In the present application, we apply physical programming to the redundancy allocation for multi-state systems. As will be shown, physical programming offers a flexible and effective approach for the optimal design of multi-state system.

Genetic algorithm (GA) is a very powerful optimization approach [14], and it is used to solve the proposed physical programming based optimization model due to the following three reasons: (1) the design variables, the number of components of each subsystems, are integer variables, (2) the objective functions in the physical programming based optimization model do not have nice mathematical properties, and thus traditional optimization approaches are not suitable in this case, (3) genetic algorithm has good global optimization performance. An example is used to illustrate the flexibility and effectiveness of the proposed physical programming approach over the approaches that are traditionally used in binary and multi-state redundancy allocation problems, e.g., the single-objective method and the fuzzy optimization method.

Acronyms

- DM: Decision Maker
- OVO: One vs. Others

Nomenclature

- $\bar{g}_i$: Class function of objective i
- g_{ij}: Boundary value of preference ranges for objective i. j=1, 2, ...,5
- N: Number of subsystems (stage)
- S_i: Subsystem (stage) i , $1 \leq i \leq N$
- n_i: Number of components in S_i
- h_i: Component version in S_i
- H_i: The number of types of components available for S_i
- M: The maximum state level of the components and the system
- x_{ij}: The state of component j of subsystem i
- p_{ik}: Probability of a component of subsystem i in state k
- $\boldsymbol{x}$: A vector representing the states of all components in the multi-state system
- $\phi(\boldsymbol{x})$: State of the system. $\phi(\boldsymbol{x}) = 0,\ 1,\ ...,\ M$
- U: System utility

- C: System cost
- W: System weight
- s: State s of component or system
- u_s: The utility when system is in state s
- c_i: Cost function of component in Si
- w_i: Weight function of component in Si
- $\boldsymbol{n} = (n_1, n_2, ...n_N)$
- $\boldsymbol{h} = (h_1, h_2, ...h_N)$
- $\bar{g}_U$: Class function of system utility
- $\bar{g}_C$: Class function of system cost
- $\bar{g}_W$: Class function of system weight
- U_0: System utility constraint value
- C_0: System cost constraint value
- W_0: System weight constraint value
- $\mu_{\tilde{U}}$: Fuzzy membership function of system utility
- $\mu_{\tilde{C}}$: Fuzzy membership function of system cost
- $\mu_{\tilde{W}}$: Fuzzy membership function of system weight

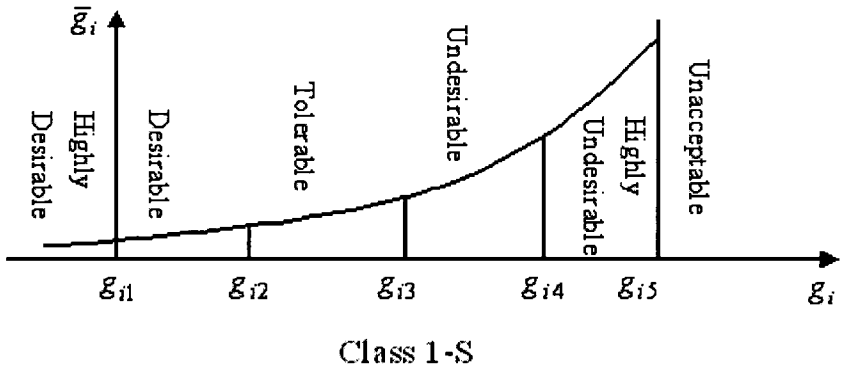

Figure 7.1: Qualitative meaning of soft class function

7.1.1 Physical programming synopsis

Physical Programming [61] is a multi-objective optimization tool that explicitly incorporates the DM's preferences on each design metric into the optimization process. Within the physical programming procedure, the design metrics are classified into four classes: smaller is better (i.e., minimization), larger is better (i.e., maximization), value is better, and range is better. There are two so-called class functions, one soft and one hard, with respect to each class [61]. The hard class functions are used to represent the constraints, while the soft class functions become additive constituent components of the aggregate objective function (to be minimized) of the optimization model. Consider for example the case of class-1 soft class function (class 1-S), the qualitative meaning of the preference function is depicted in Fig. 1. The value of the design metric, g_i , is on the horizontal axis, and the corresponding class function, $\bar{g}_i$, is on the vertical axis. A lower value of the preference function is better than a higher value thereof.

Physical programming allows the DM to express ranges of differing levels of preference with respect to each design metric with more flexibility and speci-

ficity than by simply declaring minimize, maximize or equal to. For Class 1-S, shown in Figure 7.1, the ranges are defined as follows.

Highly desirable range: $g_i \leq g_{i1}$,

Desirable range: $g_{i1} \leq g_i \leq g_{i2}$,

Tolerable range: $g_{i2} \leq g_i \leq g_{i3}$,

Undesirable range: $g_{i3} \leq g_i \leq g_{i4}$,

Highly undesirable range: $g_{i4} \leq g_i \leq g_{i5}$,

Unacceptable range: $g_i \geq g_{i5}$.

The parameters g_{i1} through g_{i5} are physically meaningful constants associated with each design metric i . What the DM needs to do in the physical programming framework is just to specify the values of the parameters g_{i1} , g_{i2} , g_{i3} , g_{i4} , and g_{i5} for each design metric i, and the class function can be completely determined by these parameters. Refer to Messac [61] for the details on how to build class functions from these boundary parameters.

The range limits define the intra-criteria preference, while the "One vs. Others" criteria rule (OVO rule) describe the inter-criteria preference. Suppose there are two options: (1) full reduction for *one* criterion across a given preference range, say, the tolerable range; (2) full reduction for all the other criteria across the next better range, say, the desirable range. The OVO rule decides that option (1) is preferred over option (2). For example, assume that we have four criteria to be minimized, criterion 1 to 4. We say that the reduction of criterion 1 from the right boundary to the left boundary of the tolerable range is preferred over the reductions of criterion 2, 3, and 4 all from the right boundary to the left boundary of the desirable ranges. The OVO rule is built into the generated class function of each criterion.

Class functions also transform design metrics with disparate units and phys-

ical meaning into a dimensionless scale. The aggregate objective function is built by combining all the soft class functions. In redundancy allocation problems, there are only Class 1-S functions (to be minimized) and Class 2-S functions (to be maximized), thus the physical programming problem model has the following form [61]:

$$\begin{aligned} \min_{x} \ & g(x) = \log_{10}\left\{\tfrac{1}{n_{sc}}\textstyle\sum_{i=1}^{n_{sc}} \bar{g}_i[g_i(x)]\right\} \\ \text{s.t.} \quad & g_i(x) \leq g_{i5} \qquad \text{(for class } 1-\text{S)} \\ & g_i(x) \geq g_{i5} \qquad \text{(for class } 2-\text{S)} \\ & x_{jm} \leq x_j \leq x_{jM} \end{aligned} \tag{7.1}$$

where x_{jm} and x_{jM} represent corresponding minimum and maximum values, n_{sc} is the number of the soft design metrics that the problem comprises.

The physical programming approach has the following characteristics: (1) it eliminates the iterative weight-adjusting process, thus substantially reduces the computational intensity; (3) The DM only needs to specify desirability ranges for each design metric, not those meaningless weights, which makes this approach very friendly to users; (3) The DM's preferences are specified on each design metric individually, therefore, physical programming is suitable to deal with a larger number of design objectives. While in the weight-based methods, there's no reasonable way to specify the weights for the objectives in advance.

7.1.2 Physical programming based multi-state system optimization model and solution approach

In this section, we will present the approach to evaluate the system utility of a multi-state series-parallel system, formulate the multi-state redundancy al-

location problem model based on physical programming, and give the solution approach using GA for the physical programming based optimization model.

7.1.2.1 System utility evaluation of multi-state series-parallel system

As shown in Figure 2.1, a multi-state series-parallel system consists of N subsystems, S_1 to S_N, connected in series. Each subsystem, say S_i, has n_i identical components connected in parallel.

To design such a system, we need to select the type of components to use for each subsystem and determine the number of components of the selected type [13, 51]. The following assumptions are used:

(1). For each subsystem S_i, there are H_i types of components available in the market. For a type h_i $(1 \leq h_i \leq H_i)$ component, its state probability distribution, cost and weight are specified.

(2). The states of the components in a subsystem are identically and independently distributed.

(3). The component and the system may be in $M+1$ possible states, namely, 0, 1, 2, ..., M.

According to multi-state system definition of Barlow and Wu [3], the state of a parallel system is defined to be the state of the best component in the system, and the state of a series system is defined to be the state of the worst component in the system. Hence, the system state of the series-parallel shown in Figure 2.1 is

$$\phi(x) = \min_{1 \leq i \leq N} \max_{1 \leq j \leq n_i} x_{ij} \tag{7.2}$$

And the probability that the system is in state s or above ($s = 0, 1, ..., M$) is

$$\Pr\left(\phi(x) \geq s\right) = \prod_{i=1}^{N}\left[1 - \left(1 - \sum_{k=s}^{M} p_{ik}\right)^{n_i}\right] \tag{7.3}$$

where $p_{ik} = \Pr\left(x_{ij} = k\right)$ for any j .

The index of system utility is used to measure the performance of a multi-state series-parallel system[2]

$$U = \sum_{s=0}^{M} u_s \cdot \Pr\left(\phi(x) = s\right) \tag{7.4}$$

where U is the expected system utility, and u_s is the utility when system is in state s .

For given h_1 , h_2 ,... , h_N and n_1 , n_2 ,... , n_N values, we need to determine the system cost and system weight, the other two design objectives. We use the system cost and system weight formulations suggested by Dhingra [15]. The total cost of the system is expressed as

$$C = \sum_{i=1}^{N} c_i(h_i)\left[n_i + \exp\left(n_i/4\right)\right]. \tag{7.5}$$

And the system weight takes the form:

$$W = \sum_{i=1}^{N} w_i(h_i) n_i \exp\left(n_i/4\right) \tag{7.6}$$

7.1.2.2 Physical programming based redundancy allocation model

In the redundancy allocation problem for multi-state series-parallel system, the design variables are the component version vector $\boldsymbol{h}$ and component number

vector $\boldsymbol{n}$

$$\boldsymbol{h} = (h_1, h_2, ...h_N), \boldsymbol{n} = (n_1, n_2, ...n_N) \tag{7.7}$$

The design metrics under consideration are system utility, system cost and system weight. The system utility, which is to be maximized, is described using class-2S class function in the physical programming approach framework. The system cost and system weight, which are to be minimized, are both described using class-1S class function.

The multi-objective optimization model of the multi-state system is formulated as

$$\begin{aligned}
&\min_{n,\,h}\ g(\boldsymbol{n}, \boldsymbol{h}) = \log_{10}\left\{\tfrac{1}{3}\left[\bar{g}_U\left(U(\boldsymbol{n}, \boldsymbol{h})\right) + \bar{g}_C\left(C(\boldsymbol{n}, \boldsymbol{h})\right) + \bar{g}_W\left(W(\boldsymbol{n}, \boldsymbol{h})\right)\right]\right\} \\
&\text{s.t.} \\
&\quad U(\boldsymbol{n}, \boldsymbol{h}) \geq U_0 \\
&\quad C(\boldsymbol{n}, \boldsymbol{h}) \leq C_0 \\
&\quad W(\boldsymbol{n}, \boldsymbol{h}) \leq W_0 \\
&\quad 0 < h_i \leq H_i, \quad i = 1, 2, ..., N \\
&\quad 0 < n_i, \qquad\quad i = 1, 2, ..., N \\
&\quad n_i,\ h_i \text{ are integers}
\end{aligned} \tag{7.8}$$

where $g(\boldsymbol{n}, \boldsymbol{h})$ is the aggregate objective function, $\bar{g}_U$, $\bar{g}_C$ and $\bar{g}_W$ are the class functions of system utility, system cost and system weight, respectively, H_i is the number of types for component C_i , and U_0 , C_0 and W_0 are the constraint values, which are equal to the boundaries of the acceptable ranges of the corresponding design metrics.

7.1.2.3 Genetic algorithm as a solution approach

GA is a very powerful optimization approach with two key advantages. (1) Flexibility in modeling the problem. GA has no strict mathematical requirements, such as derivative requirement, on the objective functions and constraints. The only requirement is that the objective functions and constraints can be evaluated in some way. GA is also suitable for dealing with those problems including discrete design variables. (2) Global optimization ability. GA has been recognized as one of the most effective approaches in searching for the global optimal solution.

The procedure of GA is as follows.

(1). Initialization. Set the size of population and the length of the chromosome. Set $k = 0$, and generate the initial population $P(0)$.

(2). Evaluation. Calculate the fitness value of each chromosome of the current population $P(k)$. Save the chromosome $B(k)$ with the best fitness value.

(3). Select. Select chromosomes from the current population based on their fitness values to form a new population $P(k+1)$.

(4). Cross. One point crossover is used to $P(k+1)$.

(5). Mutate. Implement even mutation on $P(k+1)$.

(6). Duplication. Use $B(k)$ to replace the first chromosome in $P(k+1)$.

(7). If the maximal iteration is reached, terminate the procedure and output the result. Otherwise, set $k = k+1$, and go to step (2).

7.1.2.4 Advantages of the proposed physical programming based multi-state system optimization approach

The redundancy allocation problem of multi-state system is formulated as multi-objective optimization problem in this section, because there are multiple objectives such as utility, cost and weight under consideration. The proposed multi-objective optimization model provides us a systematic framework to deal with these multiple objectives. Previous studies used single-objective optimization methods to solve such problems, with one objective as the optimization criterion and the other objectives as constraints. The single-objective optimization method is easy to understand, but it is difficult to specify these constraint values in advance. After a solution is obtained, it is often needed to modify these constraint values to find a better tradeoff between these goals. But finding the most appropriate constraint values is a trial and error process, it is time-consuming, and there is no guarantee that a satisfactory solution will be obtained in the end.

Multi-objective optimization methods have been applied to redundancy optimization of binary-state systems, and a popular one among these methods is fuzzy optimization method [15, 76, 77]. Fuzzy optimization method allows DM to express his or her preferences on different objectives using fuzzy membership functions. However, fuzzy membership functions are not flexible enough to clearly express DM's preferences, and there is no internal mechanism to control the tradeoff among different objectives.

Physical programming, however, provides more flexibility for the DM to specify his or her preferences on each objective using class functions. The DM can specify different levels of preference on each objective easily. In addition, the OVO rule of physical programming provides the internal mechanism to

control tradeoff among different objectives. In addition to optimizing each individual objective, physical programming will drive the optimal values of different objectives to preference ranges close to one another.

7.1.3 An example

An example is used to illustrate the redundancy allocation procedure for a multi-state series-parallel system using physical programming based multi-objective optimization approach, and demonstrate the advantages of the physical programming approach over single-objective method and the fuzzy optimization method.

A multi-state series-parallel system with four subsystems is considered. The state space of the component and the system is $\{0, 1, 2, 3\}$. Suppose we have different versions of components for each subsystem, with their characteristics listed in Table 7.1, which gives component cost $c_i(h_i)$, component weight $w_i(h_i)$, and the state distribution for each component, p_{i0} to p_{i3} . The system utility u_s with respect to the corresponding system state s is shown in Table 7.2.

Three methods, single-objective optimization method, fuzzy optimization method and the physical programming approach, are used to optimize this multi-state system so as to maximize system performance utility and minimize system cost and system weight simultaneously. Comparative studies are made to show the advantages of the physical programming approach over the other two methods.

GA is used to solve the formulated optimization models for all the three methods. The GA parameter settings we used in this example are as follows. The decimal encoding is used. The population size is chosen to be 100. The

Table 7.1: Characteristics of available components

S_i	$A(h_i)$	p_{i0}	p_{i1}	p_{i2}	p_{i3}	$c_i(h_i)$	$w_i(h_i)$
1	1	0.100	0.450	0.250	0.200	0.545	7
	2	0.060	0.450	0.200	0.290	0.826	3
	3	0.045	0.400	0.150	0.405	0.975	10
	4	0.038	0.350	0.160	0.452	1.080	8
2	1	0.050	0.450	0.300	0.200	0.550	12
	2	0.040	0.400	0.300	0.260	0.630	5
	3	0.040	0.300	0.320	0.340	0.740	8
	4	0.030	0.250	0.250	0.470	0.900	10
	5	0.025	0.215	0.180	0.580	1.150	12
3	1	0.145	0.625	0.130	0.100	0.250	10
	2	0.110	0.400	0.250	0.240	0.380	12
	3	0.065	0.450	0.230	0.255	0.494	13
	4	0.080	0.300	0.300	0.320	0.625	15
	5	0.050	0.250	0.250	0.450	0.790	13
	6	0.038	0.235	0.240	0.487	0.875	17
4	1	0.115	0.535	0.200	0.150	0.545	10
	2	0.074	0.550	0.186	0.190	0.620	15
	3	0.045	0.440	0.215	0.300	0.780	12
	4	0.035	0.330	0.250	0.385	1.120	14

Table 7.2: System utility with respect to each system state

s	0	1	2	3
u_s	0.0	0.5	0.8	1.0

chromosome length is 12: the first 4 digits are used to represent the component version variables h_1 to h_4 , and the rest 8 digits are used to represent the numbers of components n_1 to n_4 , each with 2 digits. We use the roulette-wheel selection scheme, one-point cross operator with cross rate of 0.25, and even mutation operator with mutate rate of 0.1.

In any iteration of GA, we have a population of 100 candidate solutions to be evaluated. Suppose the single aggregate objective function to be minimized is f . $f^{\max}$ and $f^{\min}$ are the maximum and minimum value of the population in current iteration. The fitness value for a candidate solution with aggregate objective function value f is

$$F = \frac{f^{\max} - f + 0.3 \cdot \left(f^{\max} - f^{\min}\right)}{(1+0.3) \cdot (f^{\max} - f^{\min})}$$

where the number 0.3 is the so-called "selection pressure" [99]. The "selection pressure" factor in the formula above is used to adjust the fitness value of a candidate solution based on its aggregate objective function value. When the "selection pressure" is bigger, the probability for those candidate solutions with relatively large aggregate objective function values will become higher, while that for those with relatively small aggregate objective function values will become lower. If the "selection pressure" is set to be zero, the probability for the candidate solution with the largest aggregate objective function value will be zero.

7.1.3.1 Single-objective optimization method

First, we use single-optimization method to get the optimal redundancy allocation scheme. Here, system utility is used as the objective, and system cost

Table 7.3: Optimization results using different methods

	Single-objective optimization	Fuzzy optimization	Physical programming
U	0.9654	0.9492	0.9245
C	38.7021	32.2833	27.9569
W	985.8467	699.2584	548.8717
h	(4, 5, 6, 4)	(4, 5, 6, 4)	(4, 5, 5, 4)
n	(6, 5, 4, 6)	(5, 4, 4, 5)	(4, 3, 4, 5)

and system weight are used as constraints in the optimization model:

$$\begin{aligned} &\max_{n,\,h} \; U(\boldsymbol{n}, \boldsymbol{h}) \\ &\text{s.t.} \\ &\quad C(\boldsymbol{n}, \boldsymbol{h}) \le C_0 \\ &\quad W(\boldsymbol{n}, \boldsymbol{h}) \le W_0 \\ &\quad 0 < h_i \le H_i, \quad i = 1, 2, ..., N \\ &\quad 0 < n_i, \qquad\quad i = 1, 2, ..., N \\ &\quad n_i,\ h_i \text{ are integers} \end{aligned} \tag{7.9}$$

where the utility constraint value and the weight constraint value are chosen as $C_0 = 45$ and $W_0 = 1000$.

Solve the constrained single-objective optimization problem formulated in (7.9), and we get the optimization results shown in Table 7.3. After investigating the optimization result, the DM may think that the utility goal is better than enough, and he wants to improve the cost and weight goals by partly sacrificing the utility goal. But it is hard to specify the new constraint values for cost and weight design metrics. The constraint-adjusting process is a trial-and-error process, and it is quite time-consuming.

7.1.3.2 Fuzzy optimization method

Like Dhingra's work [15], we use the logarithm sigmoid function to define the fuzzy membership functions for the design metrics of system utility, cost and weight, since the logarithm sigmoid function is a popular nonlinear function to define fuzzy membership functions. The standard logarithm sigmoid function is

$$f(x) = 1 \Big/ \left(1 + e^{-x}\right) \tag{7.10}$$

We have $f(5) = 0.9933$, $f(-5) = 0.0067$. Hence, we use the range [-5, 5] of the standard logarithm sigmoid function to define the fuzzy membership functions for the three design metrics (shown in Figure 7.2):

$$\mu_{\tilde{U}}(x) = \begin{cases} 0, & x < 0.9 \\ \frac{f[100(x-0.9)-5]-f(-5)}{f(5)-f(-5)}, & 0.9 \leq x \leq 1 \end{cases} \tag{7.11}$$

$$\mu_{\tilde{C}}(x) = \begin{cases} 0, & x > 45 \\ \frac{f[(x-25)/2-5]-f(5)}{f(-5)-f(5)}, & 25 < x \leq 45 \\ 1, & x \leq 25 \end{cases} \tag{7.12}$$

$$\mu_{\tilde{W}}(x) = \begin{cases} 0, & x > 1000 \\ \frac{f[(x-400)/40-5]-f(5)}{f(-5)-f(5)}, & 400 < x \leq 1000 \\ 1, & x \leq 1000 \end{cases} \tag{7.13}$$

There are two critical values for each fuzzy membership function. For instance, for the cost objective, the two critical values are 25 and 45, which are

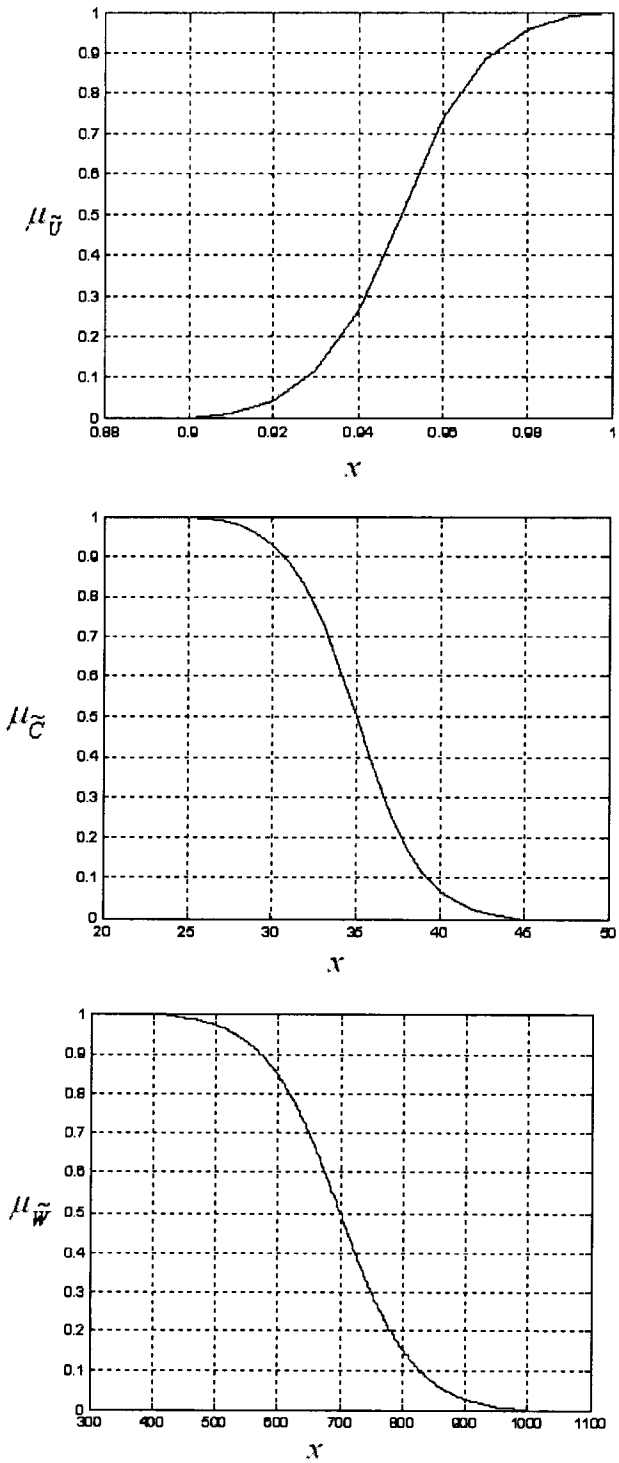

Figure 7.2: Fuzzy membership functions

the starting point and the ending point of the sigmoid part of the membership function, respectively. To ensure a fair comparison, for the cost and weight objectives, we select the critical points with respect to membership function value 0 as the constraint values used in the single-optimization method. The critical point for the utility objective with respect to membership function value 0 is set to be 0.9.

The aggregate objective function is defined as

$$\mu_{\tilde{D}} = \min\left(\mu_{\tilde{U}}(U),\ \mu_{\tilde{C}}(C),\ \mu_{\tilde{W}}(U)\right) \tag{7.14}$$

where the fuzzy set $\tilde{D}$ is the intersection set of fuzzy sets $\tilde{U}$, $\tilde{C}$ and $\tilde{W}$. And the fuzzy multi-objective optimization model is formulated as

$$\begin{aligned}
&\max_{n,\,h}\ \mu_{\tilde{D}} \\
&\text{s.t.} \\
&\quad U(\boldsymbol{n},\boldsymbol{h}) \geq U_0 \\
&\quad C(\boldsymbol{n},\boldsymbol{h}) \leq C_0 \\
&\quad W(\boldsymbol{n},\boldsymbol{h}) \leq W_0 \\
&\quad 0 < h_i \leq H_i, \quad i = 1,2,...,N \\
&\quad 0 < r_i, \qquad\quad i = 1,2,...,N \\
&\quad r_i,\ h_i \text{ are integers}
\end{aligned} \tag{7.15}$$

where the constraint values U_0 , C_0 and W_0 are equal to the critical points for the utility objective with respect to membership function value 0.

Solving the optimization above, we get the optimization results shown in Table 7.3. Compared to the results obtained using single-objective optimization method, the cost and weight objectives are improved, while the utility objective deteriorates. The fuzzy optimization method aims at optimizing the

Table 7.4: Physical programming class functions setting

	g_{i1}	g_{i2}	g_{i3}	g_{i4}	g_{i5}
Utility	0.99	0.98	0.95	0.92	0.9
Cost	15	20	25	30	45
Weight	400	500	600	800	1000

three objectives simultaneously, and finding a compromising solution.

7.1.3.3 Physical programming approach

In the physical programming framework, the utility objective has Class-2S class function (larger is better), while the cost and weight objectives have Class-1S class functions (smaller is better). The class functions settings for the three objectives are shown in Table 7.4. In order to get a result with better system cost performance, we enforce more strict requirements on the cost objective through the class function setting.

For the purpose of a fair comparison, the constraint values in the physical programming optimization model formulated in (7.8) are set to be the same as those in fuzzy optimization models, and they are also equal to the g_{i5} values for the three objectives.

Solving the physical programming optimization model, we get the optimization results shown in Table 7.3. The cost objective is improved greatly. The optimal utility and cost values both fall into the undesirable ranges, while the optimal weight value falls into the tolerable range. The optimal system costs with both signle-objective optimization approach and fuzzy optimization approach fall into the highly undesirable range, which is what the designers highly do not want. From the results shown in Table 7.3, the physical programming approach can generate the optimal results that best meet designers'

preferences on the three design objectives.

From the optimization procedures and results, we can find that the physical programming method has two advantages over the fuzzy optimization method:

(1). The class function of physical programming provides more flexibility for the DM to specify his or her preferences on each objective. The DM can specify different levels of preference on each objective. By modifying the preference ranges of the cost objective, we can get a greatly improved optimal cost value. Using fuzzy optimization, however, the DM can only specify two critical points for each objective. There are different kinds of fuzzy membership functions available, e.g. sigmoid membership function and linear membership function, but it won't provide more flexibility, either.

(2). The class function provides the preference inside a single objective, and the OVO rule of physical programming provides the preferences among different objectives. Therefore, besides optimizing each individual objective, physical programming will drive the optimal values of different objectives to preference ranges close to one another. The class functions and the OVO rule lead to optimal solution that best satisfies the DM's preferences on different objectives.

(3). A problem of using fuzzy optimization approach is that maybe the resulted optimal cost and weight are good while the utility objective is totally unsatisfying. Of course we can adjust the fuzzy membership functions when encountering this problem, but we can not ensure that the optimal results will be satisfying next time. When using the physical programming approach, however, with class functions accurately describing the DM's preferences on each objective and with the OVO rule as the inter-criteria preference, we can ensure to get satisfying optimal results with respect to each design criterion.

7.1.4 Conclusions

In this section, we present a multi-objective optimization approach for redundancy allocation for multi-state series-parallel systems. This approach seeks to maximize system performance utility while minimizing system cost and system weight simultaneously. Physical programming is used as an effective approach to optimize the system structure within this multi-objective optimization framework. The physical programming approach offers a flexible and effective way to address the conflicting nature of these different objectives. Genetic algorithm is used in solving the proposed physical programming based optimization model. The example illustrates the flexibility and effectiveness of the proposed physical programming approach over the single-objective method and the fuzzy optimization method.

The multi-objective optimization approach for multi-state series-paralle systems presneted in this section can be extended to deal with multi-objective optimization problems for other types of multi-state systems.

7.2 Joint reliability-redundancy allocation for multi-state series-parallel systems

In the previous section, we discussed how to improve reliability based design from the standpoint of design objectives, that is, how to deal with multiple conflicting design objectives. In this section, however, we will look at how to improve reliability based design from the standpoints of design variables.

In binary-state system design, there are basically two options to improve the reliability of the system: to increase the component reliability, or to provide redundancy at various stages [45]. The reliability-redundancy allocation problem was first introduced by Misra and Ljubojevic [65]. They considered

the problem of simultaneously determining optimal component reliabilities and optimal redundancy levels for a series-parallel system subject to a cost constraint. The component cost was assumed to be an exponential function of component reliability at each stage. Tillman et al [96] assumed a different cost-reliability relationship, and, in addition to cost constraint, included weight and volume constraints in their optimization model. Mathematically, the reliability-redundancy allocation problem is a mixed integer nonlinear programming problem. Misra and Ljubojevic and Tillman et al used heuristic approaches to solve it [65, 96]. Other solution approaches include the branch-and bound technique [44], XKL method [98], surrogate constraints method [26], and evolutionary algorithms [75].

In multi-state systems, the concept corresponding to the concept of reliability in binary system is state distribution. State distribution of a component refers to the probabilities of the component in different possible states. Similarly, there can also be two options to improve the system utility of a multi-state series-parallel system: (1) to provide redundancy at each stage, (2) to improve the component state distribution, that is, make a component in states with respect to higher utilities with higher probabilities.

Current studies on optimal design of multi-state series-parallel systems mainly focus on the problem of determining the optimal redundancy for each stage. Levitin et al [51] assumed that there were different versions of components for selection for each stage, and proposed a redundancy optimization model for multi-state series-parallel systems to determine the optimal component versions and redundancies for various stages. The problem was later extended to including both redundancy and maintenance optimization [48]. Ramirez-Marquez and Coit [78] proposed a heuristic approach for solving the

redundancy allocation problem formulated by Levitin et al. [51]. Liu et al [56] presented a neural network approximation approach for redundancy optimization of continuous-state parallel-series systems. Their model only had component redundancies as design variables.

The multi-state system structure optimizations reviewed above for multi-state systems are only partial optimization. Component state distributions should also be considered to be design variables. The option of selecting different versions of components provides more flexibility than just using redundancy. But this option totally depends on the products available on the market, and the available versions are always limited. By determining the optimal values of component state distributions, we can have our design optimization built into the manufacturing or scheduling process, so as to have more flexibility and get better optimization results.

In this work, we present an optimization model for multi-state series-parallel system to jointly determine the optimal component state distribution and optimal redundancy for each stage. The reason why component state distribution can be used as a controllable design variable is presented. The relationship between component state distribution and component cost is discussed based on an assumption on the treatments on the component. The physical programming-based optimization model is presented. An example is used to illustrate the optimization model and its solution approach. The materials in this section has been documented in paper [91].

7.2.1 Problem formulation

Acronyms

DM: Decision Maker

Notation

M: The maximum state level of the components and the system

N: Number of subsystems (stage)

S_i: Subsystem (stage) i , $1 \leq i \leq N$

n_i: Number of components in subsystem S_i

p_{ij}: Probability of component i in state j

$\boldsymbol{x}$: A vector representing the states of all components in the multi-state system

$\phi(\boldsymbol{x})$: State of the system. $\phi(\boldsymbol{x}) = 0,\ 1,\ ...,\ M$

U: System utility

C: System cost

u_s: The utility when system is in state s

r_{ik}: Defined term, $r_{ik} = p_{ik} \Big/ \sum_{l=0}^{k} p_{il},\ \ k = 1,\ 2,\ ...,\ M$

c_i: Cost function of component in S_i

α_{ik} , β_{ik}: Characteristic constants with respect to state k in S_i

t: The mission time

f: Aggregate objective function in physical programming-based model

$\bar{g}_U$: Class function of system utility

$\bar{g}_C$: Class function of system cost

U_0: System utility constraint value

C_0: System cost constraint value

g_{ij}: Boundary value of preference ranges for objective i. j=1, 2, . . . ,5

X: Design variable vector

h_i: Component version for stage i

Assumptions

(1) The components in a subsystem are identical and independently distributed.

(2) The components and the system may be in $M+1$ possible states, namely, 0, 1, 2, ..., M.

(3) The multi-state series parallel systems under consideration are coherent systems.

The structure of a multi-state series-parallel system is given in Figure 2.1 in Chapter 2. A multi-state series-parallel system has N subsystems connected in series, and each subsystem i has n_i identical and independently distributed components connected in parallel. The probability of component i in state j is p_{ij} .

7.2.1.1 Design variables

The design variables in the reliability-redundancy allocation problem for multi-state series-parallel system are component state distributions p_{ij} $(i = 1, 2, ..., N,\ j = 1, 2, ..., M)$ and redundancies n_i $(i = 1, 2, ..., N)$. It is apparent that redundancy is controllable, so that it can be used as design variable. We will justify in this part that component state distribution is also a controllable design variable.

In the binary-state case, the reliability of a component is its probability of working, and it has been used as a design variable in reliability-redundancy allocation problems of binary-state systems. In the case of multi-state system, let us consider a three-state system where the components and system have three states {0, 1, 2} . We have the following two statements: (a) The proba-

bility of a component in state 1 or 2 can be regarded as the reliability of this component if the word "working" means that the component's state is greater or equal to 1. (b) Similarly, the probability of a component in state 2 can be regarded as the reliability of this component if the word "working" means that the component's state is greater than or equal to 2. Therefore, just like the binary case, the state distribution of each component can be used as a design variable.

If we want to get a component with state distribution $\{p_0, p_1, p_2\}$, we can first ensure the component's reliability is p_2 under the reliability meaning of (b), and then ensure the component's reliability is $p_1 + p_2$ under the reliability meaning of (a).

Let's investigate an example of how the component state distribution is controlled. Suppose that there is a three-state component, and two treatments can be used to influence the state distribution of the component. Treatment 1: will increase the probability of the component in state 1, but will not influence the probability of the component in state 2; Treatment 2: will increase the probability of the component in state 2. Therefore, for this three-state component, using the two treatments on the component, we can control the state distribution of the component. That is, the state distribution of this component can be regarded as a controllable design variable.

7.2.1.2 System utility evaluation

System utility is one of the most widely used performance measures for multi-state systems [2]. There is a utility value related to each possible system state, and system utility is the expected utility of the multi-state system. The probability that the system state of a multi-state series-parallel system is in

state s ($s = 0, 1, ..., M$) or above can be determined using Equation (7.3), and the system utility can be determined using Equation (7.4).

7.2.1.3 Formulation of system cost

In the binary-state reliability-redundancy allocation problem, Misra and Ljubojevic [65] assumed that there is an exponential relationship between component cost and component reliability. Tillman et al [96] assumed another cost-reliability relationship, which is smoother than the one proposed by Misra and Ljubojevic.

The cost of subsystem i given by Tillman et al [96] is

$$C_i = c_i(r_i)\left[n_i + \exp\left(\frac{n_i}{4}\right)\right], \tag{7.16}$$

in which

$$c_i(r_i) = \alpha_i\left(\frac{-t}{\ln r_i}\right)^{\beta_i} \tag{7.17}$$

where $c_i(r_i)$ is the cost-reliability relationship function for a component in subsystem i, α_i and β_i are constants representing the inherent characteristics of components in subsystem i, and t is the required mission time.

Now let's investigate the possible cost formulation in the case of multi-state system, where the system and components may be in $M+1$ states. The state distribution of a component in subsystem i is denoted by $\{p_{i0}, p_{i1}, ..., p_{iM}\}$. The cost formulation of the component is based on the following assumption:

Assumption: there are M treatments that can influence the component's state distribution, and treatment k will increase the probability of the component in state k, but will not influence the probability of the component in the states above k.

Let

$$r_{ik} = \frac{p_{ik}}{\sum_{l=0}^{k} p_{il}}, \quad k = 1,\ 2,\ ...,\ M \tag{7.18}$$

Here r_{ik} can be considered to be the binary-state reliability of component i under treatment k. We define the cost of the component as

$$c_i(p_{i1}, p_{i2}, ..., p_{iM}) = \sum_{k=1}^{M} \alpha_{ik} \left(\frac{-t}{\ln r_{ik}}\right)^{\beta_{ik}} \tag{7.19}$$

where α_{ik} and β_{ik} are characteristic constants with respect to state k, $0 < \alpha_{i1} < \alpha_{i2} < ... < \alpha_{iM}$ and $1 \leq \beta_{i1} < \beta_{i2} < ... < \beta_{iM}$, and t is the mission time. The system cost is

$$C = \sum_{i=1}^{N} c_i(p_{i1}, p_{i2}, ..., p_{iM}) \left[n_i + \exp\left(n_i/4\right)\right] \tag{7.20}$$

For each subsystem, the additional cost $c_i(p_{i1}, p_{i2}, ..., p_{iM}) \exp\left(n_i/4\right)$ is included to represent the cost for interconnecting parallel components.

7.2.1.4 Characteristics of the optimization problem

Two objectives, system utility and system cost, are considered in our optimization model. Component state distributions p_{ij} $(i = 1, 2, ..., N, \quad j = 1, 2, ..., M)$ and redundancies n_i ($i = 1, 2, ..., N$) are to be determined so as to maximize system utility and minimize system cost. Note that $p_{i0} = 1 - \sum_{k=1}^{M} p_{ik}$. This problem is formulated mathematically as a mixed integer multi-objective optimization problem, in which the continuous variables represent the component state distributions and the integer variables represent the redundancies.

Available multi-objective optimization approaches include surrogate worth trade-off method [83], fuzzy optimization method [76], physical programming [61], etc. The fuzzy optimization approach and the physical programming ap-

proach have been describeed in Chpater 7. Compared to other multi-objective optimization approaches, physical programming provides a better way to specify designers' preferences on different objectives, and it is easier to be used in practical problems. In this work, the physical programming approach is used to model and solve this reliability-redundancy optimization problem with two objectives.

7.2.1.5 Physical programming-based optimization problem formulation

The physical programming approach has been briefly discussed in Section 7.1.1. In the work, we only consider system utility and system cost as design objectives, since the focus of this work is the joint reliability redundancy optimization. The physical programming-based optimization model for the multi-state reliability allocation problem is formulated as follows:

$$
\begin{aligned}
&\min f = \log_{10}\left\{\tfrac{1}{2}\left[\bar{g}_U(U) + \bar{g}_C(C)\right]\right\} \\
&\text{s.t.} \\
&\quad U \geq U_0 \\
&\quad C \leq C_0 \\
&\quad 0 \leq p_{ij} \leq 1, \quad i = 1,2,...,N,\ j = 1,2,...,M \\
&\quad \textstyle\sum_{j=1}^{M} p_{ij} \leq 1, \quad i = 1,2,...,N \\
&\quad 0 < n_i, \qquad i = 1,2,...,N \\
&\quad n_i \text{ are integers}
\end{aligned}
\tag{7.21}
$$

where f is the aggregate objective function; $\bar{g}_U$ and $\bar{g}_C$ are the class functions of system utility and system cost, and they are functions of system utility U and system cost C , which can be calculated using the formula presented in this section; $\bar{g}_U$ is Class-2S class function, and $\bar{g}_C$ is Class-1S class function; U_0

Table 7.5: System utility with respect to each system state

s	0	1	2
u_s	0.0	0.5	1.0

and C_0 are constraint values, they are equal to the boundaries of the acceptable ranges of the corresponding objectives. Refer to Messac [61] for detailed descriptions on the physical programming approach.

The physical programming-based model presented in equation (7.21) is a mixed integer single-objective optimization model. Algorithms such as branch-and-bound, and generalized Benders decomposition and outer approximation [23] have been used to solve this kind of problems. However, the most effective algorithm to solve mixed integer optimization problems, we believe, is genetic algorithm (GA). The encoding method of GA enables it to directly represent continuous design variables and discrete design variables as well, which makes the solution process much simpler. In addition, GA has very good global optimization capability. Therefore, GA is used to solve the physical programming based model in this work.

7.2.2 An example

A 3-stage multi-state series-parallel system is used to illustrate the proposed reliability-redundancy allocation approach. The three stages (subsystems) are connected in series, and each subsystem i has n_i identical and independently distributed components connected in parallel. The system is a three-state system, where the system and components can be in three states 0, 1 and 2. The system utility u_s with respect to the corresponding system state s is shown in Table 7.5.

Table 7.6: Characteristic constants for the system

Stage i	α_{i1}	α_{i2}	β_{i1}	β_{i2}
1	1.510-5	4.010-5	1.2	1.5
2	0.910-5	3.210-5		
3	5.210-5	9.010-5		

Table 7.7: Physical programming class functions setting

	g_{i1}	g_{i2}	g_{i3}	g_{i4}	g_{i5}
Utility	0.99	0.97	0.92	0.85	0.75
Cost	30	50	100	150	200

The cost of component is assumed to follow the relationship given in equation (7.19). The characteristic constants used in this example are presented in Table 7.6. The mission time is set to be $t = 1000$.

In the physical programming framework, the utility objective has Class-2S class function (larger is better), while the cost objective has Class-1S class functions (smaller is better). The class function settings for the three objectives are shown in Table 7.7, where g_{i1} to g_{i5} represent the boundaries of different desirable ranges for the utility and cost objectives (Messac, 1996). The constraint values U_0 and C_0 are equal to the boundary values g_{i5} of the corresponding objectives. There are 9 design variables in this problem:

$$X = \{p_{11},\ p_{12},\ p_{21},\ p_{22},\ p_{31},\ p_{32},\ n_1,\ n_2,\ n_3\} \tag{7.22}$$

GA is used to solve the formulated single-objective nonlinear optimization model shown in equation (7.21). In this problem, the population size is chosen to be 100. The decimal encoding is used and the chromosome length is set to be 15. We use the roulette-wheel selection scheme, one-point cross operator

Table 7.8: Optimization results for the reliability-redundancy allocation problem

Stage i	State distribution		n_i	System utility	System cost	f
	p_{i1}	p_{i2}				
1	0.2030	0.4200	8	0.9734	89.4761	-0.1581
2	0.2109	0.4300	8			
3	0.2100	0.4000	7			

with cross rate of 0.25, and even mutation operator with mutation rate of 0.1.

The results obtained from the GA algorithm are listed in Table 7.8, where f denotes the aggregate objective function value in the physical programming-based optimization model. The optimal system utility value is 0.9734, which falls into the highly desirable range. The optimal system cost value is 89.4761, which falls into the desirable range. Such a result is satisfactory considering the DM's preferences on these two objectives.

If there are only limited versions of components with specific state distribution for each stage, the result we get will not be as optimal. Suppose that there are four different versions of components available for each stage, as shown in Table 7.9, where h_i denotes the component version for stage i . We can select component from the available versions for each stage and determine the optimal redundancy values. We use the same characteristic constants setting and the same approach to evaluate the system cost as those in the reliability-redundancy allocation problem above. The physical programming class functions setting and the system utility with respect to each system state are also set to be the same.

After solving the redundancy optimization problem, we get the results shown in Table 7.10. Compared with the results obtained by reliability-redundancy allocation optimization in Table 7.8, we can find that the opti-

Table 7.9: Characteristics of available components

Stage i	h_i	p_{i1}	p_{i2}
1	1	0.30	0.52
	2	0.25	0.35
	3	0.12	0.45
	4	0.10	0.60
2	1	0.21	0.64
	2	0.25	0.33
	3	0.30	0.48
	4	0.14	0.40
3	1	0.28	0.40
	2	0.21	0.50
	3	0.15	0.55
	4	0.30	0.15

Table 7.10: Optimization results for the redundancy allocation problem

Stage i	H	State distribution		n_i	System utility	System cost	f
		p_{i1}	p_{i2}				
1	3	0.1200	0.4500	7	0.9721	89.5769	-0.1533
2	3	0.3000	0.4800	7			
3	1	0.2800	0.4000	7			

mized system utility and system cost are both a little bit worse. The aggregate objective function value is larger, which is caused by the additional constraints. The state distributions of the selected components will not happen to be the same as the optimal state distribution obtained in the joint optimization of state distributions and redundancies presented above.

7.2.3 Conclusions

This work identifies that there are two options to improve the system utility of a multi-state series-parallel system: (1) to provide redundancy at each stage,

(2) to improve the component state distributions. We present an optimization model for multi-state series-parallel system to attempt to jointly determine the optimal component state distribution and optimal redundancy for each stage. This work explains how component state distribution might be used as controllable design variable, and presents the relationship between component state distribution and component cost based on an assumption on the treatments on the component. The example illustrates the optimization model and its solution approach, and the reason why the proposed reliability-redundancy allocation model is superior to the current redundancy allocation models.

7.3 Concluding remarks

In this chapter, we have discussed how to improve the reliability based optimal design of multi-state series-parallel systems. We develop a physical programming and genetic algorithms based approach for dealing with multiple conflicting design objectives, such as system reliability and system cost, which are typically involved in reliability based optimization of multi-state systems. It has been illustrated that the proposed approach is more effective in capturing designers' preferences on different design objectives.

We have also in this chapter discussed how to extend the redundancy optimization of multi-state series-parallel systems to the joint reliability-redundancy optimization of multi-state series-parallel systems. That is, in addition to redundancies, component state distributions are treated as design variables as well. The optimization model is developed, and a numerical example is used to illustrate the superiority of the proposed model over current redundancy allocation models. Practical applications of the proposed joint reliability-redundancy optimization needs to be futher explored. Hopefully, what have been done in

this chapter can provide basic approaches for joint reliability-redundancy optimization of multi-state systems, and inspire people to develop more effective and practical approaches for this purpose.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Reliability is one of the most critical performance measures of today's complex systems, and has been emphasized more and more by academia, industry and government. Reliability of a system needs to be evaluated accurately, and it can be achieved through optimal reliability design.

In traditional binary reliability framework, both systems and components can only take two possible states: completely working and totally failed. However, engineering systems typically have multiple partial failure states in addition to the above-mentioned completely working and totally failed states. Reliability analysis considering multiple possible states is known as multi-state reliability analysis, and systems under consideration are known as multi-state systems. Multi-state reliability analysis recognizes the multiple possible states of engineering systems. It can map component performances to system performances more accurately, and be able to answer more questions such as the probabilities of a system in different possible states.

In conclusion, this research work makes significant contributions to multi-state system reliability theory, including reliability modeling, evaluation and

optimal design of multi-state systems.

(1) Efficient methods have been developed for reliability evaluation of multi-state systems with k-out-of-n structures. Since it is hard to find a method that can be used to efficiently evaluate the reliability of systems with any type of system structures, it is thus very important to develop methods for efficient reliability evaluation of systems with special structures. We have studied multi-state systems with k-out-of-n structures and developed more efficient reliability evaluation methods. We have developed exact reliability evaluation methods as well as a reliability bounding approach for the generalized multi-state k-out-of-n system model defined by Huang et al (2000). We have presented another special multi-state k-out-of-n system model with higher flexibility, compared to Huang's model (2000), together with its reliability evaluation algorithm. Finally, we have proposed a unified k-out-of-n system model, which can treat the binary k-out-of-n system models, multi-state k-out-of-n models and weighted k-out-of-n models as special cases, and developed efficient methods for its reliability evaluation.

(2) An efficient method has been developed for reliability evaluation of multi-state two-terminal networks given all minimal path vectors. The proposed method is based on the recursive computation principle and the Sum of Disjoint Products principle, and is called the Recursive Sum of Disjoint Products (RSDP) method. It is more efficient than reported methods including the Inclusion-Exclusion (IE) method and Aven's method (1985).

(3) The recursive computation principle is the common principle in the system reliability evaluation algorithms proposed in this work. Three basic elements in a recursive algorithm have been identified

and discussed: recursive function, updating procedure and boundary conditions. The system reliability evaluation algorithms developed in this work for multi-state k-out-of-n systems and multi-state two-terminal networks are all recursive algorithms.

(4) An effective method has been developed for dealing with multiple objectives involved in the optimal design of multi-state series-parallel systems. The method is based on physical programming and genetic algorithms, and it has been illustrated to be more effective in capturing designers' preferences on different design objectives than the reported methods in the literature such as single-objective method and fuzzy optimization method.

(5) A joint reliability and redundancy optimization method has been developed for multi-state series-parallel systems. In addition to redundancies, component state distributions are treated as design variables as well. The method has been illustrated to be superior to current redundancy allocation methods.

With efficient reliability evaluation methods and effective reliability based design approaches for multi-state engineering systems, the research results out of this work would provide useful tools for achieving highly reliable and cost effective engineering systems.

8.2 Future work

Based on the multi-state system reliability modeling, evaluation and design methods presented in this research work, there are some interesting topics for future study.

8.2.1 Application study of multi-state system reliability

In the area of multi-state system reliability, most of the reported research studies are focused on theoretical study. The contributions of this thesis work are also mainly on the theoretical side. More application study should be carried on.

The fact is on one hand, a lot of practical reliability related problems are multi-state reliability problems, such as those for power transmission and generation systems, transportation systems, communication systems, etc. These problems are currently dealt with using less efficient methods such as simulation methods. On the other hand, there are a lot of research going on in the area of multi-state system reliability and a lot of challenging problems have been given better solutions, such as the reliability evaluations methods proposed in this work for multi-state k-out-of-n systems. Thus, more application study of multi-state system reliability would be very valuable to fill the gap. The most immediate step would be to identify practical multi-state systems with k-out-of-n structures, and apply the methods developed in this work for the reliability evaluation of such systems. I would also like to explore large scale practical applications of multi-state network reliability methods. Practical engineering systems that I would like to investigate further include transportation systems particularly road transportation systems and railway transportation systems, manufacturing systems, and power generation and transmission systems.

8.2.2 Study of more complex multi-state systems

We might be able to develop more effective methods for more complex multi-state systems, based on the methods developed in this thesis work. For exam-

ple, we have developed an efficient method for evaluating reliability of multi-state two-terminal networks. The method might be able to be extended to the case of multi-state all-terminal networks, which is a more general and practical case.

In this thesis work, we have focused on multi-state systems with explicitly specified system structures, such as k-out-of-n structures and two-terminal network structure. Many multi-state systems do not have such a special structure. The structure function, describing the relationship between system performance and component performances, might be a general function. Or the system model might be a simulation based model. As an example, let us consider the road transportation system within the city of Edmonton, Alberta, Canada. A road section might be in different states, depending on the road conditions. The system level performance measure is the satisfaction level of the residents on the road transportation system, which is dependent on the conditions of the road sections in the system. This is a complex relationship and can not be described using a simple system structure. Thus, this road transportation system is a multi-state system with no special structures. Effective reliability modeling and evaluation methods are yet to be developed for such complex cases of multi-state system.

8.2.3 Integrated design and maintenance optimization

In Section 7.2, we present an approach for joint reliability and redundancy optimization of multi-state series-parallel systems. In addition to considering system reliability at the design stage, maintenance is an important way to improve system reliability when the system is in the operation stage. There are reported researches studies in separate reliability based optimal design

and maintenance optimization. However, in the reliability based optimization of systems such as power generation systems, we would be able to generate better optimization results, in terms of improving system reliability and lowering system cost, if we conduct the integrated design and maintenance optimization. Specifically, the integrated optimization model for a series-parallel system might be built by using system reliability and system cost as design objectives and using the following variables as design variables: the number of components in each subsystem, the versions of components in each subsystem, maintenance intervals, and maintenance resources allocation.

8.2.4 Convergence of multi-state system reliability analysis and reliability based design optimization

Multi-state system reliability has been intensively studied in the field of reliability engineering, mainly focusing on systems with special structures, such as network systems, k-out-of-n systems and consecutively connected systems. The developed reliability evaluation algorithms aims at efficiently evaluating systems with a large number of components. In the area of mechanical engineering, however, a lot of research studies on reliability based design optimization (RBDO) have been conducted [18, 17, 68]. RBDO has been mainly focused on systems with general structures. And reported RBDO approaches typically suffer from the limitation of only being able to deal with a small number of uncertainty sources, or components if we use the term in multi-state system reliability analysis. However, the goals for multi-state system reliability in the reliability engineering area and RBDO in mechanical engineering area are the same, that is, to develop effective and efficient tools for system reliability evaluation and design optimization. There is a good po-

tential to develop more effective and efficient tools based on research results developed in these two areas. Specifically, we might be able to apply results developed in one area to the other to achieve better solutions, and we might also be able to develop efficient reliability evaluation approaches for systems with general system structure and a large number of components.

BIBLIOGRAPHY

[1] T. Aven. Reliability evaluation of multistate systems with multistate components. *IEEE Transactions on Reliability*, 34: 473-479, 1985.

[2] T. Aven. On performance-measures for multistate monotone systems. *Reliability Engineering & System Safety*, 41(3):259-266, 1993.

[3] R.E. Barlow and A.S. Wu. Coherent systems with multistate components. *Mathematics of Operations Research*, 3(4): 275-281, 1978.

[4] R.E. Barlow and K.D. Heidtmann. Computing k-out-of-n system reliability. *IEEE Transactions on Reliability*, 33(4): 322-323, 1984.

[5] N.C. Beaulieu. On the generalized multinomial distribution, optimal multinomial detectors, and generalized weighted partial decision detectors. *IEEE Transactions on Communications* 39: 193-194, 1991.

[6] R. Billinton and R. Allan. *Reliability evaluation of power systems.* Plenum Press, New York, 1996.

[7] H.W. Block and T.H Savits. A decomposition for multistate monotone systems. *Journal of Applied Probability*, 19(2): 391-402, 1982.

[8] R.A. Boedigheimer and K.C. Kapur. Customer-driven reliability models for multi-state coherent systems. *IEEE Transactions on Reliability*, 43(1): 46-50, 1994.

[9] R.D. Brunelle and K.C. Kapur. Continuous State Space System Reliability: An Interpolation Approach. *IEEE Transactions on Reliability*, 47(2): 181-187, 1998.

[10] R.D. Brunelle and K.C. Kapur. Review and classification of reliability measures for multistate and continuum models. *IIE Transactions*, 31(12): 1171-1180, 1999.

[11] J.D. Campbell and A.K.S. Jardine. *Maintenance Excellence: Optimizing Equipment Life-cycle Decisions*, New York: M. Dekker, 2001.

[12] Y. Chen and Q.Y. Yang. Reliability of two-stage weighted-k-out-of-n systems with components in common. *IEEE Transactions on Reliability*, 54(3): 431-440, 2005.

[13] D.W. Coit and A.E. Smith. Solving the redundancy allocation problem us-ing a combined neural network/genetic algorithm approach. Computers & Operations Research, 23(6): 515-526, 1996.

[14] L. Davis. *Handbook of genetic algorithms.* New York : Van Nostrand Reinhold, 1991.

[15] A.K. Dhingra. Optimal apportionment of reliability and redundancy in series systems under multiple objectives. *IEEE Transactions On Reliability*, 41: 576-582, 1992.

[16] J. Doucette, M. Clouqueur, W. D. Grover. On the Availability and Capacity Requirements of Shared Backup Path-Protected Mesh Networks. *Optical Networks Magazine, Special Issue on Engineering the Next Generation Optical Internet*, 4(6):29-44, 2003.

[17] X. Du, A. Sudjianto and W. Chen. An Integrated Framework for Optimization under Uncertainty Using Inverse Reliability Strategy. *ASME Journal of Mechanical Design*, 126(4): 561-764, 2004.

[18] X. Du and W. Chen. Sequential Optimization and Reliability Assessment for Probabilistic Design. *ASME Journal of Mechanical Design*, 126(2): 225-233, 2004.

[19] E. El-Neweihi, F. Proschan and J. Sethuraman. Multi-state coherent system. *Journal of Applied Probability*, 15: 675-688, 1978.

[20] E. El-Neweihi, F. Proschan. Degradable Systems - A Survey of Multistate System-Theory. *Communications in Statistics-Theory and Methods*, 13(4): 405-432, 1984.

[21] E. El-Neweihi, F. Proschan and J. Sethuraman. Optimal allocation of multistate components. In: *Handbook of Statistics, Vol.7: Quality Control and Reliability.* Edited by P.R.Krishnaiah, C.R.Rao. North-Holland, Amsterdam, pp.427-432, 1988.

[22] E.A. Elsayed. *Reliability Engineering*, Addison Wesley Longman, New York, 1996.

[23] C. Floudas. *Nonlinear and mixed-integer optimization: fundamental and applications.* Oxford University Press, New York, 1995.

[24] W. Griffith. Multistate Reliability Models. *Journal of Applied Probability*, 17: 735-744, 1980.

[25] U. Gurler and A. Kaya. A maintenance policy for a system with multistate components: an approximate solution. *Reliability Engineering & System Safety*, 76: 117-127, 2002.

[26] M. Hikita, Y. Nakagawa, K. Nakashima and H. Narihisa. Reliability optimization of systems by a surrogate-constraints algorithm. *IEEE Transactions on Reliability*, 41(3): 473-480, 1992.

[27] W.M. Hirsch, M. Meisner and C. Boll. Cannibalization in Multicomponent Systems and Theory of Reliability. *Naval Research Logistics*, 15(3): 331-360, 1968.

[28] H.Z. Huang. Reliability evaluation of a hydraulic truck crane using field data with fuzziness. *Microelectronics and Reliability*, 36(10): 1531-1536, 1996.

[29] H.Z. Huang. Fuzzy multi-objective optimization decision-making of reliability of series system. *Microelectronics and Reliability*, 37(3): 447-449, 1997.

[30] H.Z. Huang, Y.K. Gu, X. Du. An interactive fuzzy multi-objective optimization method for engineering design. *Engineering Applications of Artificial Intelligence*, 19(5): 451-460, 2006.

[31] H.Z. Huang, Z. Tian and Y. Gu. Reliability and redundancy apportionment optimization using interactive physical programming. *International Journal of Reliability, Quality And Safety Engineering*, 11: 213-222, 2004.

[32] H.Z. Huang, Z. Tian and M.J. Zuo. Intelligent interactive multiobjective optimization method and its application to reliability optimization. *IIE Transactions*, 37(11): 983-993, 2005.

[33] J. Huang. *Multi-state system reliability analysis*, Ph.D. Thesis, University of Alberta, 2001.

[34] J. Huang and M.J. Zuo. Multi-state k-out-of-n system model and its application. In *Proceedings of the 2000 Annual Reliability and Maintainability Symposium*", pp. 264-268, 2000.

[35] J. Huang and M.J. Zuo. Dominant multi-state systems. *IEEE Transactions on Reliability*, 53(3): 362-368, 2004.

[36] J. Huang, M.J. Zuo and Z. Fang. Multi-state consecutive-k-out-of-n systems. *IIE Transactions*, 35(6): 527-534, 2003.

[37] J. Huang, M.J. Zuo and Y.H. Wu. Generalized multi-state k-out-of-n:G systems. *IEEE Transactions on Reliability*, 49(1): 105-111, 2000.

[38] J.C. Hudson and K.C. Kapur. Reliability Theory for Multistate Systems with Multistate Components. *Microelectronics and Reliability*, 22(1): 1-7, 1982.

[39] J.C. Hudson and K.C. Kapur. Reliability analysis for multistate systems with multistate components. *IIE Transactions*, 15, 127-135, 1983.

[40] J.C. Hudson and K.C. Kapur. Modules in coherent multistate systems. *IEEE Transactions on Reliability*, 32: 183-185, 1983.

[41] J.C. Hudson and K.C. Kapur. Reliability bounds for multistate systems with multistate components. *Operations Research*, 33(1): 153-160, 1985.

[42] T. Inagaki, K. Inoue and H. Akashi. Interactive optimization of system reliability under multiple objectives. *IEEE Transaction on Reliability*, 27: 264-267, 1978.

[43] S. Kuo, S. Lu, and F. Yeh. Determining terminal pair reliability based on edge expansion diagrams using OBDD. *IEEE Transactions on Reliability*, 48: 234-246, 1999.

[44] W. Kuo, H. Lin, Z. Xu and W. Zhang. Reliability optimization with the Lagrange multiplier and branch-and-bound technique. *IEEE Transactions on Reliability*, 36(5): 624-630, 1987.

[45] W. Kuo, V.R. Prasad, F.A. Tillman and C.L. Hwang. *Optimal reliability design: fundamentals and applications.* Cambridge: Cambridge University Press, 2001.

[46] W. Kuo and M.J. Zuo. *Optimal Reliability Modeling: Principles and Applications.* John Wiley & Sons, New York, 2003.

[47] G. Levitin. *Universal Generating Function in Reliability Analysis and Optimization.* Springer-Verlag, 2005.

[48] G. Levitin and A. Lisnianski. Joint redundancy and maintenance optimization for multistate series-parallel systems. *Reliability Engineering & System Safety*, 64(1): 33-42, 1999.

[49] G. Levitin and A. Lisnianski. Optimal multistage modernization of power system subject t oreliability and capacity requirements. *Electrical Power Systems Research*, 50: 183-190, 1999.

[50] G. Levitin and A. Lisnianski. Structure optimization of multi-state system with two failure modes. *Reliability Engineering & System Safety*, 72: 75-89, 2001.

[51] G. Levitin, A. Lisnianski, H. Ben Haim and D. Elmakis. Redundancy optimization for series-parallel multistate systems. *IEEE Transactions on Reliability*, 47(2): 165-172, 1998.

[52] W. Li, M.J. Zuo. Reliability evaluation of multi-state weighted k-out-of-n systems. *Reliability Engineering and System Safety*, 2006. Accepted for Publication.

[53] H.T. Liao, E.A. Elsayed and L.Y. Chan. Maintenance of continuously monitored degrading systems. *European Journal of Operational Research*, 175(2): 821-835, 2006.

[54] J.S. Lin, C.C. Jan and J. Yuan. On reliability evaluation of a capacitated-flow network in terms of minimal pathsets. *Networks*, 25: 131-138, 1995.

[55] A. Lisnianski and G. Levitin. *Multi-state System Reliability: Assessment, Optimization and Applications.* World Scientific, Singapore, 2003.

[56] P.X. Liu, M.J. Zuo and M. Q-H Meng. A neural network approach to optimal design of continuous-state parallel-series systems. *Computers and Operations Research*, 30: 339-352, 2003.

[57] Y. Liu and K.C. Kapur. Reliability Measures for Dynamic Multi-State Non-repairable Systems and Their Applications for System Performance Evaluation. *IIE Transactions*, 38: 511-520, 2006.

[58] L. Lu and J. Jiang. The unavailability, spurious operation and cost analysis for standby k-out-of-n systems. *Reliability Engineering and System Safety*, 2005. Accepted for Publication.

[59] F. Meng. More on optimal allocation of components in coherent systems. *Journal of Applied Probability*, 33: 548-556, 1996.

[60] M.C. Meng. Comparing two reliability upper bounds for multistate systems. *Reliability Engineering and System Safety*, 87(1): 31-36, 2005.

[61] A. Messac. Physical Programming: Effective Optimization for Computational Design. *AIAA Journal*, 34(1): 149-158, 1996.

[62] A. Messac and B. Wilson. Physical Programming for Computational Control. *AIAA Journal*, 36: 219-226, 1998.

[63] A. Messac, M. Martinez and T. Simpson. Introduction of a Product Family Penalty Function using Physical Programming. *ASME Journal of Mechanical Design*, 124: 164-172, 2002.

[64] A. Messac and A. Ismail-Yahaya. Multiobjective Robust Design using Physical Programming. *Structural and Multidisciplinary Optimization, Journal of the International Society of Structural and Multidisciplinary Optimization (ISSMO)*, 23: 357-371, 2002.

[65] K.B. Misra and M.D. Ljubojevic. Optimal reliability design of a system: a new look. *IEEE Transactions on Reliability*, 22(5): 255-258, 1973.

[66] M. Modarres, M. Kaminskiy and V. Krivtsov. *Reliability Engineering and Risk Analysis: A Practical Guide*, New York: Marcel Dekker, 1999.

[67] B. Natvig. Two suggestions of how to define a multi-state coherent system. *Applied Probability*, 14: 391-402, 1982.

[68] E. Nikolaidis, D.M. Ghiocel and S. Singhal. *Engineering Design Reliability Handbook* , CRC Press, 2004.

[69] L. Nordmann and H. Pham. Weighted voting systems. *IEEE Transactions on Reliability*, 48: 42-49, 1999.

[70] L. Nordmann and H. Pham. Reliability of decision making in human-organizations. *IEEE Transactions on Systems Man and Cybernetics Part A-Systems and Humans*, 27: 543-549, 1997.

[71] P.D.T. O'Connor. *Practical Reliability Engineering*, John Wiley & Sons, 2002.

[72] K.S. Park. Fuzzy apportionment of system reliability. *IEEE Transaction on Reliability*, 36: 129-132, 1987.

[73] C.S. Park, R. Pelot, K. Porteous and M.J. Zuo. *Contemporary Engineering Economics: A Canadian Perspective*, Second Edition, Addison Wesley Longman, Toronto, ON, 2001.

[74] M. Patel, K.E. Lewis, A. Maria and A. Messac. System Design through Subsystem Selection using Physical Programming. *AIAA Journal*, 41: 1089-1096, 2003.

[75] V.R. Prasad and W. Kuo. Reliability optimization of coherent systems. *IEEE Transactions on Reliability*, 49(3): 323-330, 2000.

[76] S.S. Rao and A.K. Dhingra. Reliability and redundancy apportionment using crisp and fuzzy multiobjective optimization approaches. *Reliability Engineering and System Safety*, 37: 253-261, 1992.

[77] V. Ravi, P.J. Reddy and H.J. Zimmermann. Fuzzy global optimization of complex system reliability. *IEEE Transactions on Fuzzy System*, 8: 241-248, 2000.

[78] J.E. Ramirez-Marquez and D.W. Coit. A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems. *Reliability Engineering & System Safety*, 83 (3): 341-349, 2004.

[79] J.E. Ramirez-Marquez and D.W. Coit. A Monte-Carlo simulation approach for approximating multi-state two-terminal reliability. *Reliability Engineering & System Safety*, 87: 253-264, 2005.

[80] J.E. Ramirez-Marquez, D.W. Coit and M. Tortorella. A Generalized Multistate-Based Path Vector Approach for Multistate Two-Terminal Reliability. *IIE Transactions*, 38(6):477-488, 2006.

[81] S. Ross. Multivalued State Component Systems. *Annals Of Probability*, 7:379-383, 1979.

[82] A.M. Rushdi. Utilization of symmetric switching functions in the computation of k-out-of-n system reliability. *Microelectronics and Reliability*, 26: 973-987, 1986.

[83] M. Sakawa. Multiobjective optimization by the surrogate worth trade-off method. *IEEE Transaction on Reliability*, 27: 311-314, 1978.

[84] S. Satitsatian and K.C. Kapur. An algorithm for lower reliability bounds of multistate two-terminal networks. *IEEE Transactions on Reliability*, 55(2): 199-206, 2006.

[85] J.H. Saleh and K. Marais. Highlights from the early (and pre-) history of reliability engineering. *Reliability Engineering & System Safety*, 91(2): 249-256, 2006.

[86] C. Srivaree-Ratana, A. Konak and A.E. Smith. Estimation of all-terminal network reliability using an artificial neural network. *Computers & Operations Research*, 29(7): 849-868, 2002.

[87] H. Taboada, F. Baheranwala, D. Coit and N. Wattanapongsakorn. Practical Solutions for Multi-Objective Optimization: An Application to System Reliability Design Problems. *Reliability Engineering & System Safety*, 92(3): 314-322, 2007.

[88] Z. Tian and M.J. Zuo. Redundancy allocation for multi-state systems using physical programming and genetic algorithms. *Reliability Engineering & System Safety*, 91(9): 1049-1056, 2006.

[89] Z. Tian and M.J. Zuo. A unified model of k-out-of-n systems and its reliability evaluation. *Proceedings of the 2007 Industrial Engineering Research Conference*, Nashville, Tennessee, USA, May 19-23, 2007. pp. 1770-1775.

[90] Z. Tian, R. CM Yam, M.J. Zuo and H. Huang. Reliability bounds for multi-state k-out-of-n systems. *IEEE Transactions on Reliability*, 2006. Accepted for Publication.

[91] Z. Tian, M.J. Zuo and H. Huang. Reliability-redundancy allocation for multi-state series-parallel systems. *IEEE Transactions on Reliability*, 2004. Submitted.

[92] Z. Tian, M.J. Zuo and H. Huang. Reliability-Redundancy Allocation for Multi-State Series-Parallel Systems. *Proceedings of the 2005 European Safety & Reliability Conference (ESREL05)*, Tri City, Poland, June 20-23, 2005. pp. 1925-1930.

[93] Z. Tian, M.J. Zuo, and R. CM Yam. Performance evaluation of generalized multi-state k-out-of-n systems with independent components. *Proceedings of the Fourth International Conference on Quality and Reliability*, Beijing, China, August 9-11, 2005. pp. 515-520.

[94] Z. Tian, M.J. Zuo and R. CM Yam. Performance Evaluation for Generalized Multi-State k-out-of-n Systems. *Proceedings of the 2005 Industrial Engineering Research Conference*, Atlanta, USA, May 14-18, 2005.

[95] Z. Tian, M.J. Zuo, R. C.M. Yam. The Multi-State k-out-of-n Systems and Their Performance Evaluation. *IIE Transactions*, 2006. Submitted.

[96] F.A. Tillman, C.L. Hwang and W. Kuo. Determining component reliability and redundancy for optimum system reliability. *IEEE Transactions on Reliability*, 26(3): 162-165, 1977.

[97] J.S. Wu and RJ Chen. An algorithm for computing the reliability of a weighted-k-out-of-n system. *IEEE Transactions on Reliability*, 43: 327-328, 1994.

[98] Z. Xu, W. Kuo and H. Lin. Optimization limits in improving system reliability. *IEEE Transactions on Reliability*, 39(1): 51-60, 1990.

[99] G. Xuan and R. Cheng. *Genetic algorithm and engineering design.* Beijing: Science Press, 2000.

[100] J. Xue. On multistate system analysis. *IEEE Transactions on Reliability*, 34: 329-337, 1985.

[101] X.Y. Zang , H.R. Sun and K.S. Trivedi. A BDD-based algorithm for reliability analysis of phased-mission systems. *IEEE Transactions on Reliability*, 48(1): 50-60, 1999.

[102] M.J. Zuo, R.Y. Jiang RY, R.C.M. Yam. Approaches for reliability modeling of continuous-state devices. *IEEE Transactions on Reliability*, 48(1): 9-18, 1999.

[103] M.J. Zuo and M. Liang. Reliability of multistate consecutively-connected systems. *Reliability Engineering & System Safety*, 44: 173-176, 1994.

[104] M.J. Zuo, B. Liu and D. Murthy. Replacement-repair policy for multistate deteriorating products under warranty. *European Journal of Operational Research*, 123: 519-530, 2000.

[105] M.J. Zuo and Z. Tian. Performance evaluation for generalized multi-state k-out-of-n systems. *IEEE Transactions on Reliability*, 55(2): 319-327, 2006.

[106] M.J. Zuo and Z. Tian. An efficient method for reliability evaluation of multi-state networks given all minimal path vectors. *IIE Transactions*, 2006. Accepted for Publication.