1-1-2011

Novel models and algorithms for systems reliability modeling and optimization

Dingzhou Cao
Wayne State University,

NOVEL MODELS AND ALGORITHMS FOR SYSTEMS RELIABILITY MODELING AND OPTIMIZATION

by

DINGZHOU CAO

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirement

for the degree of

DOCTOR OF PHILOSOPHY

2011

MAJOR: INDUSTRIAL ENGINEERING

Approved by:

Advisor	Date
Co-Advisor	Date

DEDICATION

To my father, Bangjiang Cao &

my mother, Zhaoying Wen

for letting me pursue my dream

for so long, so far away from home

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Ratna Babu Chinnam, for supporting me over the years, and for his able guidance, continuous encouragement and helps. His constructive feedback, criticism, and moral support have been a great source of inspiration to me academically and in my personal development.

I am also grateful to my co-advisor Dr. Alper Murat, for his help and guidance in developing research ideas, and suggesting changes. I would like to thank other committee members, Dr. Kenneth Chelst, Dr. Leslie Monplasisir and Dr. Dmitry E. Tananko, for their ideas and suggestions for improvement.

I express my sincere thanks to my girlfriend, Yu Sun, for her continuous support and help these years. I would like to thank my all other friends in the department for their help and motivation.

TABLE OF CONTENTS

LIST OF TABLES

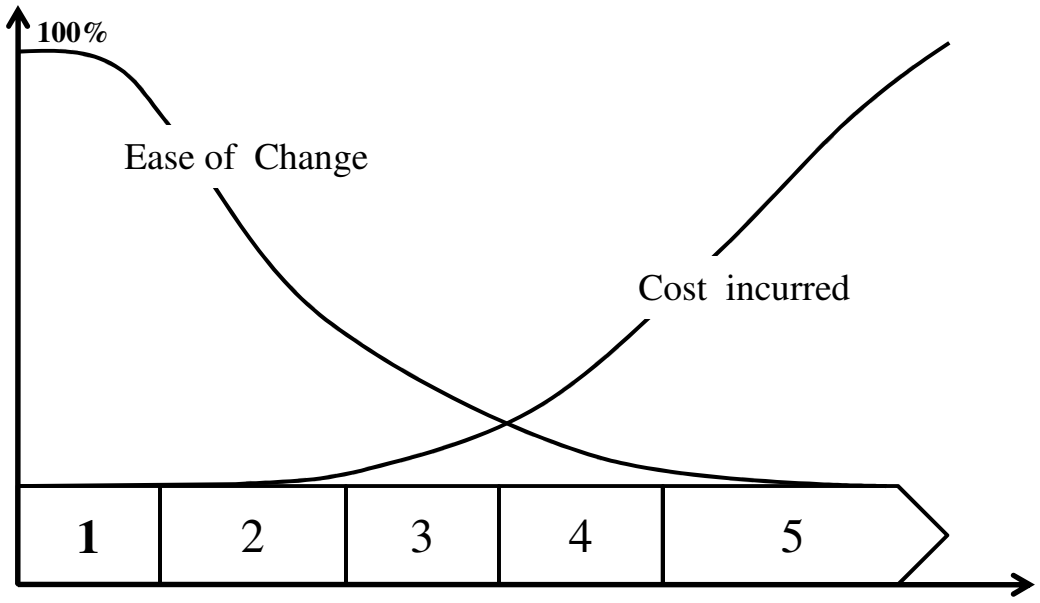
LIST OF FIGURES

Chapter 1 Introduction

1.1 Research Motivation

Over the last couple of decades, globalization and other factors have significantly changed the business environment. In today's competitive marketplace, firms are facing significant on-going challenges. The customers are not only concerned about the price of and quality the product, but also about the life cycle cost associated with product maintenance, recovery and disposal. Hence, lowering the life cycle cost (LCC) of the product is an important target for companies, which will increase the product's value and attractiveness to the customer. Reliability, Availability and Maintainability (RAM) greatly influences the life cycle cost of the product. The more reliable and maintainable the product is, the lower is its life cycle cost. It is widely recognized that the costs associated with RAM over the product's life cycle can outweigh other costs in many sectors. For space programs, operational support costs typically comprise 60% to 80% of total program cost (Montgomery 1996) ; for the supersonic fighter aircraft or subsonic transport/bomber, the cost (excluding fuel) of engines constitutes at least two-thirds of the total engine life-cycle cost(Nelson 1977). Consideration of RAM characteristics early in the product design stages can reduce the LCC. The typical product life-cycle can be divided into five phases (John, Loy et al. 2003): the need analysis & specification phase, the conceptual & preliminary design phase, the detail design & development phase, the manufacturing, production & construction phase, the system operational, support, and disposal phase. The ease of change of a design decreases rapidly as the design progresses

in time. Incurred cost also is also low at the beginning of the life cycle, but increases rapidly in the detail design and manufacturing phases, as shown in Figure 1.1.



Life cycle phases: 1: need analysis & specification, 2: conceptual & preliminary, 3: detail design & development, 4: manufacturing, production & construction, 5: system operation, support & disposal

Figure 1.1: Cost incurred, ease of change for various life cycle phases (John, Loy et al. 2003)

When the maintainability and reliability issues are addressed in the early phases, there will be fewer design iterations and design changes in the detail design phase. This is because potential issues are investigated more thoroughly and resolved in the design process. The earlier the design flaws are detected, the lower the costs and associated efforts needed to eliminate them. Furthermore, the cost drivers for operation and maintenance can also be identified in the early phase of the product design. Once cost drivers are identified, the corrective measures can be taken to avoid the induced maintenance costs by choosing another design alternative or by choosing better materials, etc.

In order to make the product more attractive to the customer and achieve the most economic life cycle cost, one needs to account for the effect of reliability/availability on LCC upfront in the design stage and choose the best alternative that reduces the total cost. Hence, developing proper models to accurately estimate the reliability/availability and maintainability aspects of a design upfront in product development is a key requirement for proper analysis and management of life-cycle costs.

1.2 Research Objective

The accurate estimation and management of the system reliability/availability during the early stages of new product development is not only critical for managing product development and manufacturing costs but also for reducing the LCC. In this regard, the overall objective of this research study is to develop an integrated framework for "design for reliability" (DFR) during upfront product development by truly treating reliability as a design parameter. The aim here is to develop the theory, methods, and tools necessary for:

1) Accurate assessment of system reliability/availability in the product development stages.
2) Optimization of the design to meet system reliability/availability targets.

In modeling the system reliability and availability, we aim to address the limitations of existing methods, in particular the Markov chains method and the Dynamic Bayesian Network approach, by incorporating a Continuous Time Bayesian Network (CTBN) framework for more effective modeling of sub-system/component interactions, dependencies, and various repair policies. We also propose a multi-object optimization

scheme to aid the designer in obtaining optimal design(s) with respect to system reliability/availability targets and other system design requirements. In particular, the optimization scheme would entail optimal selection of sub-system and component alternatives by taking multiple criteria into consideration. The theory, methods, and tools to be developed are extensively tested and validated using simulation test-bed data and actual case studies from industry examples.

1.3 Organization of Thesis

Chapter 2 presents the review of literature. Section 1 gives the concepts, definitions and assumptions in Reliability, Maintainability and Availability modeling of the reparable systems. Section 2 gives an overview of the different quantitative RAM modeling methods currently used and their limitations. Section 3 presents the relevant research on system design optimization analysis, with a focus on optimization methods.

Chapter 3 proposes a new CTBNs RAM modeling formalism. The proposed approach addresses the issue of estimating the reliability (availability) for repairable system, which has dynamic interactions among components or is subject to different repair policies. The mapping from dynamic gates of Dynamic Fault trees to CTBNs is discussed and the modeling of various repair policies in CTBNs is investigated. Three case examples from the literature and industry applications are discussed to demonstrate the approach.

Chapter 4 presents an integrated framework for system design optimization analysis. The integrated framework is based on CTBNs and NSGA-II. The main functionality of this framework is to find the optimal system configuration by taking

multiple requirements (reliability/availability, cost and weight et al) into consideration. The RAM modeling formalism (proposed in Chapter 3) is employed to estimate the system reliability (availability) and the system design problem is treated as a black-box multi-objective optimization problem. We used NSGA-II to solve this multi-objective optimization problem. Finally, a ground vehicle system design case study example is presented to demonstrate the proposed approach.

Chapter 5 proposes a modified adaptive ϵ-constraint method which can identify all Pareto-optimal solutions for the linear multi-objective optimization problem. The existing adaptive ϵ-constraint method is improved by adding a checking scheme to avoid solving for the duplicate solutions, and thus speed up the algorithm significantly. The proposed method is applied to two typical system design problems: The Redundancy Allocation Problem (RAP) for Series-Parallel systems and the configuration selection problem.

Chapter 6 concludes the thesis by summarizing contributions of the research and discussion of future research directions.

Chapter 2 RAM Modeling and System Design Optimization: Literature Review

In this chapter, the concepts, definitions and assumptions of repairable systems are given in Section 2. Section 3 gives an overview of the different quantitative RAM modeling methods being used currently. Section 4 presents the relevant research on system design optimization analysis, including the system metric (reliability/availability) estimation approaches and the system optimization methods.

2.1 Repairable Systems

The repairable system is a system which would be subject to repair actions (including the replacement) to bring it back to operating state when it fails. It is defined as contrast to non-repairable system, which is defined as that cannot be repaired or do not necessary to be repaired when fails. For example, a vehicle is a repairable system. When the transmission of the vehicle is down, what you need to do is just to repair the transmission or replace a new one instead of dumping the whole vehicle. As for non-repairable system, such as the rocket, if the launch fails, the rocket will explode or burn in the sky; it cannot be repaired. Other example such as chips in in the computer, if the CPU fails, we will replace it with a new one; nobody would repair a chip. However, in a computer system, if the chip fails and it is replaced by a new one, the chip is non-repairable; but the computer system can be considered as a reparable system in system level. Sometimes, a component is totally replaced by a new one also can be considered as repairable in system level.

2.1.1 System Dynamic Behaviors

In this study, dynamic behaviors refer to the functional dependences or interactions among components or subsystems in the system. Most common dynamic behaviors are already summarized and modeled by Dynamic Fault trees (Dugan 2000; Dugan, Sullivan et al. 2000), which is an extension of traditional fault trees to handle failure sequence (or temporal) and functional dependencies. The basic dynamic gates include the PEDP gate, the WSP gate and the PAND gate. We will describe these dynamic gates in details in later chapters. In this study, we mainly focus on proposing the CTBNs to model these dynamic gates.

2.1.2 Perfect Repair

For a repairable system, in component level, the components are subjected to failure if not periodically and properly maintained. If failed, they have to be repaired. The type of repair conducted defines the performance of the component. Perfect repair (Mettas and Zhao 2004) brings the component to an as good as new state; while minimal repair makes the component as good as it was immediately before failure. In this study, perfect repair is considered. We assume that when component failed, the repair action will restore its operating state to be as good as new. In the background of this study, we also assume that all failure time and repair time of component are following exponential distribution. The failure rate for each component can be estimated from various available data, like the field data, test data and similar product data et al.

2.1.3 Repair Policies

In system level, a repair process can have several characteristics. The repair target can be a single component or a subsystem consists of a set of components. In the latter case, the repair completion time can be the same for all components in that subsystem, or the time to repair component may be different from one to another. When repairing a subsystem, all components are repaired in that subsystem or just a minimal subset allowing the subsystem to be operative again. The repair crews can be limited or unlimited. The order to repair components can be specified, the more critical a component is, the higher repair priority it will have. Furthermore, the activation of repair process of a component or subsystem can be a failure of the same component or subsystem, etc.

In this study, several repair policies are considered and the time to repair a component or a subsystem follows exponential distribution with the repair rate of that component or subsystem.

(1). Subsystem repair policy. The repair of a subsystem is activated by the failure of the subsystem itself, as soon as it occurs. There are 3 kinds of subsystem repair policies (Raiteri, Fraceschinis et al. 2004; Portinale, Raiteri et al. 2010):

- *Subsystem Global Repair policy* (SGR): all the components in the subsystem are under repair and the repair is completed at the same time for all the components. The time to repair follows exponential distribution with a repair rate.

- *Subsystem Local Repair policy* (SLR): for a subsystem, all its components are influenced by the repair process, but the time to repair each of them may be different from one to another (each component has its own repair rate).
- *Minimal Subsystem Local Repair policy* (SLR-min): the extension of SLR, where the repair of the subsystem components is interrupted as soon as the subsystem is available again, i. e. when a minimal subset of components necessary to recover the system, is repaired. In this case, some of subsystem components may not be repaired.

(2). Component repair policy. The repair of a single component is activated by the failure of the component itself, as soon as it occurs. There are also 2 kinds of component repair policies.

- *Component Repair policy* (CR): concerns the repair of a single component. The time to repair the component can be different from one to another.
- *Component Repair policy with limited repair crews* (CR-limit). The only different of this policy and the CR policy is that, for a system or subsystem, the number of repair crews is limited.

If there is not redundancy in the system, all subsystem repair policies mentioned above will become component repair policy. In a system without any redundancy, if one component fails, then it will bring down the whole system. Thus in this kind of system, subsystem repair policies are identical to component repair policies. Furthermore, without redundancy, even if a single component fails, then it will bring down the whole system;

all components are equally important. Thus there is not minimal subset of components can recover the system. The "SLR-min" policy is not practical in this kind of system.

2.1.4 System Metrics

For a repairable system, beside reliability, the availability is another more important system metric need to be considered. Availability is defined as the probability that the system is operational at a given time, t (i.e. has not failed or it has been restored after failure). Availability is a performance criterion for repairable systems that accounts for both the reliability and maintainability properties of a system. The definition of availability is somewhat flexible and there are a number of different classifications of availability. In this study, we focus on two kinds of availabilities (Reliasoft 2011), the instantaneous (point) availability and the mean availability. For the three case examples in Chapter 4, we estimated the point availability in the first example (the Cardiac system) for evaluating the proposed method, by comparing its result with results of other methods taken from literatures; for the ground vehicle and the fleet of vehicles example, the mean availability is a more practical metric for this kind of application. We calculate the mean availability in these two case examples.

Instantaneous (point) Availability $\mathbf{A(t)}$

Instantaneous (point) availability is the probability that a system will be operational at any random time t. This is very similar to the reliability function in that it gives a probability that a system will function at the given time t. Unlike reliability, the instantaneous availability measure incorporates maintainability information. At any given time t, the system will be operational if the following conditions are met:

The item functions properly from 0 to t with probability $R(t)$ or it functions properly since the last repair at time $u, 0 < u < t$, with probability:

$$\int_0^t R(t-u)m(u)du$$

with $m(u)$ being the renewal density function of the system.

Then the point availability is the summation of these two probabilities, or:

$$A(t) = R(t) + \int_0^t R(t-u)m(u)du$$

Mean Availability $\overline{\mathbf{A(t)}}$

The mean availability is the proportion of time during a mission or time period that the system is available for use. It represents the mean value of the instantaneous availability function over a period $[0\ T]$ and is given by:

$$\overline{\boldsymbol{A(t)}} = \frac{\mathbf{1}}{\boldsymbol{t}} \int_{\mathbf{0}}^{\boldsymbol{t}} \boldsymbol{A(u)du}$$

The mean availability also can be express in a simple way. For a time period $[0\ T]$, t_{up} is the system up time, the total amount of time the system is available for use. The mean availability can be estimated by:

$$\overline{\boldsymbol{A(t)}} = \frac{\boldsymbol{t_{up}}}{\boldsymbol{T}}$$

2.2 RAM Modeling Methods: Literature Review

The RAM modeling methods can be broadly divided into two main categories:

- Analytical techniques.
- Simulation techniques.

Analytical techniques basically describe the system model in term of mathematical equations. The techniques can produce exact or approximate[1] analytical solutions to the mathematical equations. Example of such solutions includes the closed-form solution to a set of ordinary differential equations, known as the Chapman-Kolmogorov equations for Markov chains.

Simulations techniques are essentially a way to generate sample paths (i.e. runs), and statistically estimated the measures of interest from those sample paths. Simulation is a versatile tool that can be applied to any real-world system. However, these techniques are computationally heavy (i.e. usually we need a high number of runs to get an acceptable result). In reliability analysis, the Monte Carlo based Petri Net method (Portinale and C. 2009) is a stochastic simulation technique.

In this study, we only focus on analytical techniques. From a system standpoint, the analytical techniques can divide them into two classes: reliability modeling methods for non-repairable system and RAM modeling methods for repairable systems. In this subsection, we discuss them separately.

Reliability Modeling Methods for Non-repairable Systems

There are numerous methods available for reliability modeling; among them, the Fault tree and its extension Dynamic Fault trees (DFT)(Dugan, Bavuso et al. 1992),

[1] From a practical standpoint, some mathematical equations are usually solved using a particular numerical algorithm, and therefore computing an approximate solution.

solved by Markov chains (MCs) (Dugan, Bavuso et al. 1993; Dugan 2000), are the basic reliability modeling approaches. However, in practical application, MC-based DFT methods have a space state explosion problem(Boudali and Dugan 2005), which leads to long computational times and/or intractable solutions.

Recently, a number of studies have attempted to use the Bayesian Network (BN) and its extensions to model and analyze the reliability of complex system. By introducing new modeling features (multi-state variables, noisy gates, common cause failures et al.), BNs have been shown to increase both the modeling capabilities and analysis power of combinational based models. Works on system safety and Bayesian Networks, developed by Bobbio (Bobbio, Portinale et al. 2001), explain how to map static Fault trees into Bayesian Networks and show the obvious superiority of BNs over MC-based FTs in terms of modeling and analysis capabilities.

However, one limitation of conventional BNs is that, it cannot capture dynamic behaviors among subsystems or components. Two kinds of BN extensions are studied to make up this shortfall. The first one is the interval-based approach Discrete-Time Bayesian Networks (DTBNs)(Boudali and Dugan 2005). In this paper, Boudali proposes an interval-based DTBN for reliability modeling and analysis. The main character of DTBNs is that the time line is divided into finite number of intervals and all of these intervals are the state space for each node. The state of a node locating in an interval represents the failure event of the node that occurs in that interval. By using this BN time representation, it is possible to account for the temporal dependencies among the system components and event-dependent failure behaviors, avoiding the state space explosion problem of MC-based approach. The authors also show how to map DFT gates into

DTBNs. In the paper(Boudali and Dugan 2006), the authors make a further extension of DTBNs and present another formulism of BN named Continuous-Time Bayesian Networks (CTBNs, although this work is different from the CTBNs defined in this thesis, the authors use the same acronym)[2]. Unlike DTBNs, the time line in CTBNs is divided into infinite number of intervals, which means that each interval is a time point. The state of a node locating in a time point represents that the failure event of the node occurs in this point. Unfortunately, there is no general-distribution BN inference engine for CTBNs so that the authors derive a closed-from solution for it. The authors also demonstrate how to map DFT gates into CTBNs. The limitation of CTBNs and DTBNs is that they are only applied to non-reparable systems; they are not capable to model reparable systems. The other extension of BNs to cope with system dynamic behaviors is the instance-based Dynamic Bayesian Networks (DBNs) (Weber and Jouffe 2003; Montani, Portinale et al. 2005; Portinale, Raiteri et al. 2010). In DBNs, the time is divided into successive time slices, and a node is associated with each time slice. The BN model is obtained by generating a BN for a specific time slice and repeating the same structure for all other slices, thus covering the whole time range of interest. Weber (Weber and Jouffe 2003) describes a study which is dedicated to show how to use DBNs to account for dependencies among components in reliability analysis. He also compares the performance of MCs with DBNs in reliability modeling. In the paper (Montani, Portinale et al. 2005), Montani proposes to characterize dynamic gates within the DBN framework, by translating DFTs into the corresponding DBN model.

[2] From the next paragraph of this paper, the acronym CTBN refers to the definition of Continuous Time Bayesian Network framework from Nodelman, U. (2007). Continuous Time Bayesian Networks. PhD Dissertation, STANFORD UNIVERSITY.

RAM Modeling Methods for Repairable Systems

In all studies mentioned above, non-reparable systems are considered. In order to account for reparable systems, some extensions of Fault Trees are presented. In the paper (Bobbio and Raiteri 2004), the authors propose the Dynamic Repairable Parametric Fault Trees (DRPFTs), which implement DFTs in compact parametric form. By adding Repair Boxes (RB), the DFT is extended to include dependencies arising from the repair process. However, the RB semantics is investigated only in the special case of its application to a dynamic gate. In the work(Raiteri, Fraceschinis et al. 2004), another extension of FTs called Repairable Fault Trees (RFTs) is proposed; more generalized repair features are introduced. Nevertheless, one limitation is that, both methods have to be solved by Petri-net-based method eventually, and the RB introduced by DRPFTs/RFTs requires a solution in the state spaces, which may be computationally very expensive. Thus in paper (Raiteri, Fraceschinis et al. 2004), the authors seek for solutions by implementing the modular multi-solution process. Namely classifying the RFT modules into two categories, the sub-trees without RB actions are solved with the standard combinatorial method and the sub-trees with RB actions are solved by Petri-net-based method. In spite of the increased expressive power, RFTs feature the drawback of a lower solving efficiency, due to the required state-based analysis. In the paper (Portinale, Raiteri et al. 2010), Portinale shows how to take into account similar repair policies aforementioned during the reliability analysis based on DBNs formalism. However, due to the limitation of DBNs, all repair policies investigated in this paper assume unlimited repair crews, which limit its application in real world.

In order to model repair actions with limited crews, one must have a center process which can assign and reclaim repair crews. The time-slice-based modeling method DBNs are out of capability for this situation. Continuous Time Bayesian Networks (CTBNs), firstly presented by Nodelman (Nodelman 2007), are based on Bayesian Networks, but with a continuous time representation of the temporal evolution. They inherit all the advantages of BNs and are kind of event-based modeling methods, which are more flexible to model repair process under different repair policies. The technical report (Portinale and C. 2009) is the only work found to apply CTBNs in reliability modeling. In the technical report, the authors present an extension to CTBNs called Generalized CTBNs (GCTBNs) by adding immediate nodes. They claim that these immediate nodes allow capturing the logical/probabilistic interaction among the model's variables. Actually, in this thesis (Section 3.2.2, Chapter 3), we will demonstrate it is also possible to model this scenario just using the original definition of CTBNs. The authors also outline a semantic model of GCTBN based on the formalism of Generalized Stochastic Petri Nets (GSPN). In this study, we propose a CTBN framework for reliability modeling, and extend it to an RAM modeling with considering various repair policies, including the repair policy with limited repair crews. We validate the proposed approach by applying it to an example taken from literature and show that the modeling capability of CTBNs outperforms that of DBNs, which indicates the CTBN framework is a good alternative for RAM modeling and analysis.

2.3 System Design Optimization: Literature Review

The design process of a reliable system is by nature, iterative. Traditional approaches to the design process of a reliable system follow the system requirement

analysis, preliminary design, detail design, evaluation and redesign phases until a final acceptable design is obtained. However, to achieve a shorter time-to-market, system reliability concerns should be addressed at the design stage ("design for reliability"). The design requirements have to consider reliability (availability), cost, weight, physical size, power consumption, etc. The system design optimization problem can be formula as to select components or redundancy-level to optimize some objective functions, given system-level constraints on reliability (availability), cost, and/or weight.

The system design optimization problem consists of two parts: the system metrics estimation and the optimization methods. The common system metrics are system reliability (availability), cost and weight et al. The optimization methods in system design optimization problem refer to multi-objective optimization methods or single-objective optimization methods. In this study, we focus on multi-objective optimization methods.

Among these system metrics, reliability (availability) estimation is the most important system metric. For different systems, the way to calculate reliability (availability) is total different. For example, in a simple series or parallel system, the reliability (availability) can be estimated by close-form mathematical formulas; however, for a complex system with dynamic behaviors or various kind of repair policies, there are not close-form mathematical formulas for its reliability (availability), the Markov chain, Dynamic Bayesian Networks or CTBNs are the methods for the reliability (availability) estimation in this kind of systems. From optimization standpoint, we put this kind of non-close-form objective function problem as black-box optimization problem. The heuristics methods (GA, Simulated annealing and Tabu search et. al) are the most efficiency way to solve the black-box optimization problem.

Thus, in the proposed system design optimization framework, we treat the system design optimization problem based on the difference of systems. For simple structure (series/parallel) systems with close-form reliability (availability) mathematical formulas, we propose a modified adaptive ϵ-constraint method to identify all Pareto-optimal solutions. For systems with dynamic behaviors and various repair policies, we employ CTBNs to estimate its reliability (availability) and treat it as a black-box optimization problem; and NSGA-II is used to solve for its Pareto solutions.

2.3.1 System Metrics Estimation Methods

A large number of models and solution methods have been proposed to solve the system design optimization problem especially the reliability optimization problem (Redundancy Allocation Problem, RAP), such as dynamic programming, Lagrangean multiplier (Misra 1972), heuristic approach (Ramirez-Marquez and Coit 2004) and integer programming (Sharma and Misra 1990). As for the system involves active and cold-standby redundancy, Tavakkoli-Moghaddam (Tavakkoi-Moghaddam, Safari et al. 2008) proposed a genetic algorithm for a redundancy allocation problem of the Series-Parallel system with active and cold-standby redundancies. The similar jobs are done by Coit in these papers (Coit and Smith 1996; Coit 2001). However, the above-mentioned methodologies are not applicable to the kind of complex system with dynamic behaviors; they mainly focus on simple system structure: series, parallel or k-out-of-n with/without simple active or cold-standby redundancy strategy. The reliability of these simple structure (series/parallel) systems can be estimated by close-form mathematical formulas. There are few works about reliability (availability) optimization of complicated systems with dynamic behaviors. The only one work related to this is done by Ren (Ren and

Dugan 1998). In her paper, she described a methodology for embedding a GA into an existing fault-tree methodology to determine the heuristic optimal design configuration of a reliable system. She used DFTs to model interactive actions between events. However, she applied MCs-based method DIFtree SOLVER to solve the DFTs, which will have state explosion problem if the system is too large. As we discussed in previous subsection, the CTBN is a flexible tool for reliability (availability) estimation. It doesn't have the state explosion problem mentioned above, but it is more practical to model the dynamic failure behaviors between components and various repair policies. In this study, we will employ CTBNs to calculate system reliability (availability) of complex systems with dynamic behaviors for the proposed system design optimization framework.

2.3.2 Optimization Methods

In the past several decades, there have been a number of studies and approaches to the RAP. Roughly, they can be grouped into three categories: single objective optimization with constraints, aggregated objective function methods for multi-objective optimization, and Pareto-based ranking methods for multi-objective optimization.

Single objective optimization methods

The first set of methods treat the RAP as a single objective optimization problem (maximizing system reliability or minimizing cost) with constraints. Various single-objective optimization approaches have been used to solve such formulations, including dynamic programing (Bellman and Dreyfus 1958; Fyffe, Hines et al. 1968; Misra 1971), integer programming (Bulfin and Liu 1985; Misra and Sharma 1991; Billionnet 2008), meta-heuristics (Painton and Campbell 1995; Coit and Smith 1996; Ravi, Murty et al.

1997; Kulturel-Konak, Smith et al. 2003; Liang and Smith 2004; Rashika and Manju 2006), mixed integer and non-linear programming (Tillman, Hwang et al. 1977) and column generation method (Zia and Coit 2010).

Multi-objective Optimization: Aggregation Objective Function Methods

These single-objective optimization techniques have their own advantages on RAP. However, in practical applications, some researchers have realized that it should take multiple considerations into account when determining the redundancy allocation of the system. For example, they want to obtain a system with high reliability and at the same time, they still want the design cost of the system to be low. The aggregation objective function method is implemented to solve this problem. They weight sum the multiple objective functions into a single objective function, and solve the new objective function via single-objective optimization approaches. These studies (Dhingra 1992; Busacca, Marseguerra et al. 2001; Marseguerra, Zio et al. 2004; Zafiropoulos and Dialynas 2004; Tian and Zuo 2006) are belong to this category. In the paper (Dhingra 1992), the author presented a multi-objective reliability apportionment problem. The problem is a multi-objective, nonlinear, mixed-integer mathematical programming problem and is solved by sequential unconstrained minimization techniques in conjunction with heuristic algorithms. The series system considered in this study is with time-dependent reliability. The study (Zafiropoulos and Dialynas 2004) provided an efficient computational method to obtain the optimal system structure of electronic devices by using a single or a multi-objective simulated annealing algorithm based optimization approach. Studies (Marseguerra, Zio et al. 2004; Zafiropoulos and Dialynas 2004; Tian and Zuo 2006) are multi-criteria formulations using genetic algorithm.

Approach in paper (Tian and Zuo 2006) is based on GA and Monte Carlo simulation; while in paper (Marseguerra, Zio et al. 2004) GA and physical programming are combined to solve the RAP.

Multi-objective Optimization: Pareto-based ranking methods

The above multiple objectives studies have taken important steps towards finding more effective and efficient approaches for RAP. However, in order to obtain promising results, how to aggregate multiple objectives into a single one is a sophisticated work; besides, the aggregation of multiple objectives may eliminate the possibility of identifying more non-dominated solutions. To cope with these drawbacks, people come to some other multi-objective optimization approaches. Multi-objective optimization refers to the solution of problems with two or more objective to be satisfied simultaneously. Unlike single-objective optimization problem, the multi-objective optimization problems usually have a set of solutions, which called Pareto-optimal solutions or non-dominated solutions. There have been some studies in this field (Salazar, Rocco et al. 2006; Taboada, Baheranwala et al. 2007; Kulturel-Konak, Coit et al. 2008; Taboada, Espiritu et al. 2008). In the paper (Taboada, Baheranwala et al. 2007), the authors formulated the redundancy allocation problem as a tri-objective problem (maximize reliability, minimize cost and weight) and solve this problem using Non-dominated Sorting Genetic Algorithm (NSGA-II) (Deb, Pratap et al. 2002). An improving version NSGA-II was presented so called MOMS-GA (Taboada, Espiritu et al. 2008) to solve the tri-objective redundancy allocation problem in multi-state systems. In the paper (Kulturel-Konak, Coit et al. 2008), Tabu search approaches with Monte-Carlo simulation method are employed to solve a bi-objective (reliability and cost) redundancy

allocation problem. In this paper (Kulturel-Konak, Coit et al. 2008), a problem specific MOEA is employed to solve the continuous reliability optimization problems where the reliability of the components are variables to be optimized.

The meta-heuristic based multi-objective optimization approaches mentioned above are very popular on RAP now. However, meta-heuristic based approaches have several limitations: they do not guarantee that Pareto points are optimal; they may not identify all the Pareto-optimal points and they may have computation burden problem when the population size is large. Thus, in this study, from optimization standpoint, in our proposed system design optimization framework, two cases are discussed. 1) For systems with dynamic behavior and various repair policies, CTBNs is used to estimate the system reliability (availability). The system design optimization is a black-box type problem and the meta-heuristic based methods are best option for this kind of problem. Thus in our framework, NSGA-II is employed to identify the Pareto solutions for this problem. 2) For simple structure (series/parallel) systems with close-form reliability (availability) mathematical formulas, a modified adaptive ϵ-constraint method is proposed to identify all Pareto-optimal solutions for this problem. Compared with meta-heuristic based method NSGA-II, the modified adaptive ϵ-constraint method could identify all Pareto-optimal solutions, and guarantees that the identified solutions are Pareto-optimal.

The ϵ-constraint method (Chankong and Haimes 1983) is a traditional method to generate all non-dominated solutions for multi-objective optimization problems. It works by choosing one of the objective functions as the only objective and handling the remaining objective functions in the form of constraints. Through systematic variation of

the objective constraint bounds, all non-dominated solutions can be obtained. To improve the computational efficiency of the ϵ-constraint method, an improved version of the ϵ-constraint method (Ozlen and Azizoglu 2009) was proposed. Instead of using a fixed ϵ, the improved method updates the ϵ value based on the location of the previous solution. It uses an adaptive ϵ value in each iteration, which explains its name, the adaptive ϵ-constraint method. This improves the efficiency of algorithm significantly. However, there are two limitations ton this method. Firstly, the adaptive ϵ-constraint method is limited to linear objective functions and requires all coefficients of the objective function terms and the decision variables to be integers. Secondly, this method identifies duplicate solutions, affecting the efficiency of the algorithm. In this thesis, we propose efficient solutions for addressing these two limitations.

In the RAP problems, the system reliability objective function is not a linear integer function. In order to obtain a linear function, we make some mathematical transformations to the reliability objective function. In addition, to cope with the non-integral nature of the non-integer reliability objective function, we make necessary modifications to the adaptive ϵ-constraint method so that it can account for at least one non-integer linear objection function. To avoid solving for the duplicated solutions, two search refinement strategies are added to the adaptive ϵ-constraint method which can help to reduce the number of IPs solving solved significantly. Furthermore, a decomposition scheme is employed to improve the algorithm efficiency. We decompose the original problem into several sub-problems and solve each sub-problem for Pareto-optimal solution sets with the modified adaptive ϵ-constraint method. By sequentially combining

and filtering each pair of Pareto-optimal solution sets from a pair of subsystems, finally we can obtain the Pareto-optimal solutions set for the original problem.

2.4 Summary

We first introduce the concepts, definitions and assumptions of repairable systems. Then we review the different quantitative RAM analysis methods being used currently and present the relevant research on system design optimization methods. Furthermore, the limitations of existing methods are highlighted and discussed.

Chapter 3 RAM Modeling using Continuous Time Bayesian Networks

As we discuss in Section 2.3 of Chapter 2, in RAM modeling, the existing methods have limitations. Such as the traditional method Markov chains have state explosion problem when the system is too larger; while the Dynamic Bayesian Networks are incapable to model some repair policies especially the one with limited repair crews. The CTBNs, firstly presented by Nodelman (Nodelman 2007), are based on Bayesian Networks, but with a continuous time representation of the temporal evolution. They inherit all the advantages of BNs and are kind of event-based modeling methods which are more flexible to model repair process under different repair policies. In this chapter, we propose CTBN formalism for RAM modeling. The proposed method doesn't have the state explosion problem. And within the flexible modeling ability, it is capable to cope with various repair policies. The proposed approach is applied to three case examples, one is taken from the literature and another two are original from practical problems of our industrial partners. The experiment results are promising, which indicate that the presented method is a good alternative for existing RAM modeling methods.

3.1 Introduction of CTBNs

In this subsection, we provide the background material about the Continuous Time Bayesian Networks. We first introduce a continuous-time, finite-state, homogenous Markov process, and then describe the Continuous Time Bayesian Networks, which use a graphical representation to model multi-variable continuous-time stochastic process.

3.1.1 Continuous Time Markov Processes

Let X be a continuous-time, finite-state, homogenous Markov process. X has n states $\{x_1, x_2, \dots, x_n\}$. $X(t)$ is the (finite) state of the system at time t. The collection of random variables $\{X(t) | t \in R^+\}$ composes the process. X satisfies the Markov assumptions.

The initial distribution $P_X^0 = P(x(0))$ is a multinomial distribution over n states of X. The transient behavior of X is described by the initial distribution P_X^0 and the transition model which is often represented by the intensity matrix

$$Q_x = \begin{bmatrix} -q_{x_1} & q_{x_1x_2} & \cdots & q_{x_1x_n} \\ q_{x_2x_1} & -q_{x_2} & \cdots & q_{x_2x_n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{x_nx_1} & q_{x_nx_2} & \cdots & -q_{x_n} \end{bmatrix}$$

where $q_{x_ix_j}$is the intensity with which Xtransitions from $x_i$to $x_j$and $q_{x_i} = \sum_{i \neq j} q_{x_ix_j}$. The diagonal element q_i and the off-diagonal elements q_{ij}define the instantaneous transition probabilities of X.

The intensity matrix Q_x is time invariant. Given Q_x, the transient behavior of X can be described as the following: X stays in state x_i for an amount of time t and transitions to statex_j. t is exponentially distributed with parameter q_{x_i}. The expected time of transitioning is $1/q_x$. Upon transitioning, the probability that X transitions from state x_i to x_j is $\theta_{x_ix_j} = q_{x_ix_j}/q_{x_i}$. For example, assume that we want to model the changes of weather$W(t)$ which has three values ($Val\big(W(t)\big) = \{w_0 = sunny, w_1 = rainy, w_2 = cloudy\}$). We could represent the behavior of $W(t)$ using the intensity matrix

$$Q_W = \begin{bmatrix} -1.6 & 1.3 & .3 \\ .7 & -1 & .3 \\ .8 & 1.2 & -2 \end{bmatrix}$$

If we set the time unit to one day, this means that we expect the weather would change in $\frac{1}{2} = .5$ day if currently it is cloudy. When the intensity is changing, with probability $\frac{.8}{2} = .4$ the new value will be sunny and with probability $\frac{1.2}{2} = .6$ the new value will be rainy.

To model a multi-variable system, we first combine all variables into a single join variable by enumerating all possible states of the variables. If the system has N variables $X_i (i = 1, \dots, N)$, and each variable contain S_i states, the total number of states of the join process is $n = \prod_{i=1}^{N} S_i$ and the size of the intensity matrix for the join process is n by n. As the number of variables increases, the size of the intensity matrix grows exponentially.

3.1.2 Continuous Time Bayesian Networks

A continuous-time Markov process suffers from state space explosion when handling large dynamic systems. A structured representation is needed to deal with multi-variable dynamic systems.

In order to decompose a multi-variable dynamic system, we introduce a conditional intensity matrix (CIM) to describe the dynamics of local variables in a system. Let X be all the variables of the dynamic system we are trying to model. Let $x \in X$ be one variable in the system and $U \subset X$ be a set of other variables. The conditional intensity matrix $Q_{x|U}$ for variable x is defined as a set of intensity matrices $Q_{x|u}$, one for each

instantiation u of the variable set U. The evolution of X depends instantaneously on the values of the variables in U. Using a CIM, we can model each local variable as an inhomogeneous Markov process, whose intensities are a function of the current values of a set of other variables

Example 3.1.2.1 Let us expand the weather example in the previous subsection to a dynamic system with more than one node. Assume that we want to consider the effect of the weather on the people's outdoor exercise intensity, which has two values ($Val(E(t)) = \{e_0 = light, e_1 = normal\}$). We can model the dynamics of each local variable separately by utilizing the dependencies among the variables. Therefore, the dynamics of the exercise intensity can be described using three CIMs.

$$Q_{E|w_o} = \begin{bmatrix} -.1 & .1 \\ .8 & -.8 \end{bmatrix} \; Q_{E|w_1} = \begin{bmatrix} -.4 & .4 \\ 1.2 & -1.2 \end{bmatrix},$$

$$Q_{E|w_2} = \begin{bmatrix} -.3 & .3 \\ 1 & -1 \end{bmatrix}$$

The behavior of variable $E(t)$ is now represented as an inhomogeneous Markov process, whose intensities depend on the current value of $W(t)$. When $W(t) = w_0$, the behavior of $E(t)$ is descried using $Q_{E|w_o}$. When $W(t) = w_1$, it is described using $Q_{E|w_1}$. When $W(t) = w_2$, it is described using $Q_{E|w_2}$.

The way the Markov chain models the dynamic of this system is total different. We first have to list all the possible combinations of the joint variable: $\{(w_0, e_0), (w_0, e_1), (w_1, e_0), (w_1, e_1), (w_2, e_0), (w_2, e_1)\}$. We then write the transition intensity of each pair of values into the join intensity matrix. It is a 6 by 6 matrix.

Definition 3.1.2.1 A continuous time Bayesian Networks N over X consists of two components: an initial distribution P_X^0, specified as a Bayesian Networks B over X, and a continuous transition model, specified using a directed (possibly cyclic) graph G whose nodes are $x \in X$. Let U_x denote the parents of x in G. Each variable $x \in X$ is associated with a conditional intensity matrix $Q_{x|U_x}$.

Example 3.1.2.2 Assume we want to model how a person improves his health status via doing outdoor exercise. When the weather is good, or his schedule is not tight, he may exercise more. Increasing exercise intensity tends to make him more energetic and healthy, which will allow him to work more efficiently. Such a dynamic system contains four variables: Weather, Exercise, Schedule, and Body status. Each variable changes in continuous time and its changing rate depends on the current value of some other variables.

We can use a CTBN to represent such behavior. The dependencies of these four variables are depicted using a graphical structure. As shown in Figure 3.1. The quantitative transient dynamics for each variable are represented using CIMs. Let's assume all the four variables are binary. Let B(t) be the person's body status $(Val(B(t)) = \{b_0 = health, b_1 = sick\})$, E(t) be the exercise intensity $(Val(E(t)) = \{e_0 = light, e_1 = normal\})$, S(t) be his daily schedule $(Val(S(t)) = \{s_0 = loose, s_1 = tight\})$, and $W(t)$ be the weather $(Val(W(t)) = \{w_0 = sunny, w_1 = raining\})$. The conditional intensity matrices for the four variables can be specified as

Table 3.1: CIM for node Weather

State	$W(w_0)$	$W(w_1)$
$W(w_0)$	-.5	.5
$W(w_1)$	.5	-.5

Table 3.2: CIM for node Exercise

$\{W(w_0), S(s_0)\}$	State	$E(e_0)$	$E(e_1)$
	$E(e_0)$	-2	2
	$E(e_1)$	.1	-.1

$\{W(w_1), S(s_0)\}$	State	$E(e_0)$	$E(e_1)$
	$E(e_0)$	-.6	.6
	$E(e_1)$	.3	-.3

$\{W(w_0), S(s_1)\}$	State	$E(e_0)$	$E(e_1)$
	$E(e_0)$	-.5	.5
	$E(e_1)$	1.1	-1.1

$\{W(w_1), S(s_1)\}$	State	$E(e_0)$	$E(e_1)$
	$E(e_0)$	-1	1
	$E(e_1)$	.1	-.1

Table 3.3: CIM for node Schedule

$B(b_0)$	State	$S(s_0)$	$S(s_1)$
	$S(s_0)$	-.1	.1
	$S(s_1)$	.5	-.5

$B(b_1)$	State	$S(s_0)$	$S(s_1)$
	$S(s_0)$	-.5	.5
	$S(s_1)$	.8	-.8

Table 3.4: CIM for node Body status

$E(e_0)$	State	$B(b_0)$	$B(b_1)$
	$B(b_0)$	-.2	.2
	$B(b_1)$	5	-5

$E(e_1)$	State	$B(b_0)$	$B(b_1)$
	$B(b_0)$	-.1	.1
	$B(b_1)$	10	-10

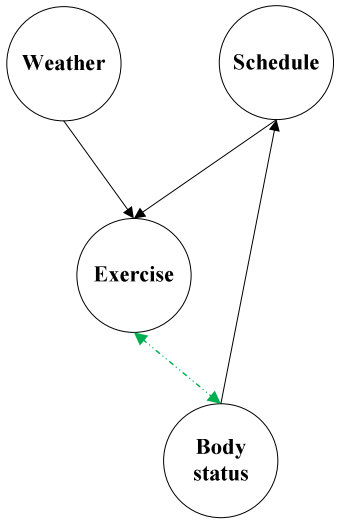


Figure 3.1: Healthy improvement example

If the person is in loose schedule, and the weather is sunny, the dynamic of his exercise intensity is determined by the intensity matrix $Q_{E|w_{0,},s_{o},}$. If the time unit is one month, we can expect he will do normal exercise in $\frac{1}{2} = .5$ month, conditioned on the fact that he is currently doing light exercise, his schedule is loose and the weather is good. The model contains double direction links, the dash line, indicating that whether a person is doing normal exercise depends on his body status; while the body status also impacts the exercise intensity of the person. For example, if he is sick, we would not do any exercise.

3.1.3 Inference in CTBNs

Given a CTBN model, we would like to use it to answer queries conditioned on observations, which is named evidence here. Evidence for a CTBN is usually a partial trajectory, in which some values or transitions are missing for some variable during some time intervals. It is also possible that the evidence has no observation; we name it as empty evidence. In this study, we mainly focus on two kinds of queries. The first one is

to query the marginal distribution of some variable at a particular time, such as the distribution of weight at $t = 10$, which can answer the kind of question like what is the probability of the person is overweight at the tenth month. In reliability analysis, this inference enables us to query the reliability or point availability of a system at a particular time. The second kind of inference is to query the expected total amount of time that a variable spends in a state in a period of time like $[0\ T]$. For example, this inference can answer the question like what is the expected total month the person is overweight for the first ten months. In reliability analysis, we can apply this to calculate the expected total amount of operational time for a repairable system so as to estimate its mean availability. The reliability or availability estimation using a CTBN is a filtering task with empty evidence for a particular node.

Inference algorithms

There are various inference algorithms available in the literature, such as the exact inference method (Nodelman, Koller et al. 2005), the expectation propagation based method (Nodelman, Koller et al. 2005; Saria, Nodelman et al. 2007), the mean field variational based method (Cohn, El-Hay et al. 2009) and the sampling based method (Fan and Shelton 2008). Next, we will give a brief introduction of the basic inference algorithm in CTBNs, the exact inference. A more complete treatment of this method can be found in Nodelman' paper (Nodelman, Koller et al. 2005).

Assume that there is a partially observed trajectory σ for a CTBN from time 0 to T. The evidence σ is divided into N interval $[t_i, t_{i+1})(i = 0, \dots, N-1)$ according to the observed transition times. That is, each interval contains a constant observation of the CTBN. We set $t_0 = 0$ and $t_N = T$.

To perform exact inference, the intensity matrix Q for the join homogeneous Markov process is generated using the amalgamation method. Then the evidence is incorporated into Q as following. The intensity matrix Q is reduced to Q_i for each interval $[t_i, t_{i+1})$ by zeroing out the rows and columns of Q which represent states that are inconsistent with the evidence. And Q_{ij} represents the matrix Q with all elements zeroed out except the off-diagonal elements that represent the intensities of transitioning from non-zero rows in Q_i to non-zero columns in Q_j. $\text{Exp}(Q_i(t_{i+1} - t_i))$ is the transition matrix for interval $[t_i, t_{i+1})$ and $Q_{i,i+1}$ coresponds to the transition probability density between two consecutive intervals at time t_{i+1}.

Next, the forward-backward algorithm for Markov processes is used to answer queries. Let $\alpha_t = p(X_t, \sigma_{[0,t]})$ and $\beta_t = p(\sigma_{[t,T)}|X_t)$ be the forward and backward probability vector respectively, $\sigma_{[t_i,t_j]}$ be the trajectory during interval $[t_i, t_j]$, α_0 be the initial distribution P_X^0 over the state at time 0 and β_T be a vector of ones. The forward and backward distribution vectors for each interval can be calculated recursively:

$$\alpha_{t_{i+1}} = \alpha_{t_i} \exp\big(Q_i(t_{i+1} - t_i)\big)\, Q_{i,i+1}$$

$$\beta_{t_i} = Q_{i-1,i} \exp\big(Q_i(t_{i+1} - t_i)\big)\, \beta_{t_{i+1}}$$

The distribution over the state of the CTBN at time $t \in [t_i, t_{i+1})$ given the evidence $\sigma_{[0,T)}$ can be computed as

$$P\big(X_t = K, \sigma_{[0,T)}\big) = \alpha_{t_i} \exp\big(Q_i(t - t_i)\big)\, \Delta_{k,k} \exp\big(Q_i(t_{i+1} - t)\big)\, \beta_{t_{i+1}}$$

where $\Delta_{i,j}$ is an $n \times n$ matrix of zeros except for a single one in position i, j. The expected total amount of time that the join Markov process spends in state j during interval $[t_i, t_{i+1})$ given the evidence is a kind of expected sufficient statistics (Nodelman, Koller et al. 2005) for CTBNs, which can be calculated as:

$$\frac{1}{C}\int_{t_i}^{t_{i+1}} \alpha_{t_i} \exp\big(Q_i(t-t_i)\big)\,\Delta_{j,j}\exp\big(Q_i(t_{i+1}-t)\big)\,\beta_{t_{i+1}}\,dt$$

where C is the normalization factor to guarantee that the summation of the total time the process spends on each state during interval $[t_i, t_{i+1})$ is $t_{i+1} - t_i$.

3.2 FT(DFT) Gates and CTBN Modeling

Fault trees (FTs) are one of the most popular techniques for reliability analysis of large, complex systems. The two most commonly used gates in a FT are the AND gate and OR gate. Dynamic Fault trees are extensions of FTs, aimed at increasing the modeling power of FTs by including new primitive gates, able to accommodate complex kind of dependencies. DFTs introduce four basic (dynamic) gates: the warm spare gate (WSP), the sequence enforcing gate (SEQ), the probabilistic dependency gate (PEDP) and the priority AND gate (PAND). In the rest of this section, we propose to characterize (Dynamic) Fault trees gates within the CTBNs framework, by translating all the basic gates into the corresponding CTBNs models. We adopt the following convention. Given a generic binary component K we denote with $K = 1$ the component failure and with $K = 0$ the component working. In the common hypothesis, component failures and repairs are exponentially distributed with failure rate λ_K and μ_k.

3.2.1 AND Gate and OR Gate

The AND gate represents that the output event occurs if all input events occur; the OR gate represents that the output event occurs if at least one of the input events occurs. The corresponding CTBN representations for AND gate and OR gate are shown in Figure 3.2. In the CTBN modeling, node A, B and C all have 2 states: state 1 means the corresponding component fails and state 0 means it is working. Their initial states are 0, which means that they are all working at the beginning. CIMs for node A and B are listed in the left part of Table 3.5. For node A/B, its initial state is 0, the expected time of transitioning from state 0 to state 1 is $\frac{1}{\lambda_A}$ or $\frac{1}{\lambda_B}$. However, when it enters state 1, it will stay in that state forever, because the expected time of transitioning from state 1 to state 0 is infinity (1/0). In reliability, this means that the component is non-repairable. This is very easy to be extended to model reparable case. For instance, suppose component A and B both are reparable and subject to a CR policy with failure rate and repair rate μ_A and μ_B respectively. When they are in state 1, they will return back to state 0 with expected time $\frac{1}{\mu_A}$ and $\frac{1}{\mu_B}$ respectively. The detailed CIM for the reparable case is in the right part of Table 3.5. For node C with OR gate, its initial state is 0. When node A and B are in state 0, node C always stays in state 0 (the expected transitioning time is infinity). However, when at least one of A or B is in state 1, it transits into state 1 immediately (the expected transitioning time is $0 = \frac{1}{\infty}$). The similar logic is applied to node C with AND gate.

Table 3.5: : CIMS for node A and B of OR gate

Node A, non-reparable	State	A(0)	A(1)		Node A, reparable	State	A(0)	A(1)
	A(0)	$-\lambda_A$	λ_A			A(0)	$-\lambda_A$	λ_A
	A(1)	0	0			A(1)	μ_A	$-\mu_A$
Node B, non-reparable	State	B(0)	B(1)		Node B, reparable	State	B(0)	B(1)
	B(0)	$-\lambda_B$	λ_B			B(0)	$-\lambda_B$	λ_B
	B(1)	0	0			B(1)	μ_B	$-\mu_B$

Table 3.6: CIMS for node C of OR gate

{A(0),B(0)}	State	B(0)	B(1)
	B(0)	0	0
	B(1)	∞	$-\infty$
{A(0),B(1)} {A(1),B(0)} {A(1),B(1)}	State	B(0)	B(1)
	B(0)	$-\infty$	∞
	B(1)	0	0

Table 3.7: CIMS for node C of AND gate

{A(0),B(0)} {A(0),B(1)} {A(1),B(0)}	State	B(0)	B(1)
	B(0)	0	0
	B(1)	∞	$-\infty$
{A(1),B(1)}	State	B(0)	B(1)
	B(0)	$-\infty$	∞
	B(1)	0	0

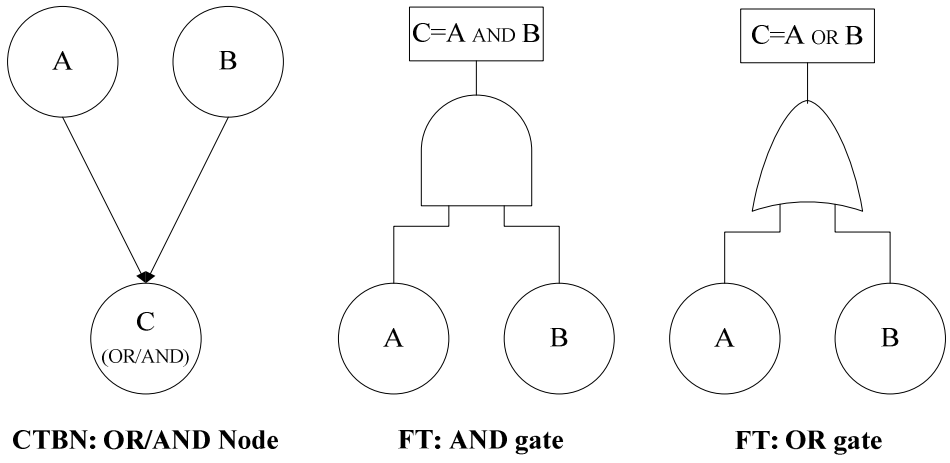


Figure 3.2: The OR and AND gate in FT and CTBN representation

3.2.2 WSP Gate

The WSP is a dynamic gate modeling one or more main components that can be replaced by one or more backups (spares). Spares can fail while they are dormant, but the failure rate of the unpowered spare is αλ with 0≤α≤1 called the dormancy factor and λ is the failure rate of an active spare. Spares are called "hot" if α=1 and "cold" if α=0. When the main component fails, the switch will activate spare component to substitute the main one with probability p_1; it fails to activate spare component with probability $p_2 (p_2 = 1 - p_1)$. As an example, let us consider a situation where a single component P can be substituted by a spare S. If any component (either main or spare) is failed or the switch fails to active spare component, the gate produces a fault. The corresponding CTBN nodes to this gate are shown in Figure 3.3. Node P has 3 states: 0, 11 and 12. State 0 means P is working; State 11 means P fails in failure modal 1, which means that the switch activates the spare component successfully; State 12 means P fails in failure modal 2, which means that the switch fails to activate the spare component. Node S and Node WSP both have 2 states: 0 and 1. State 0 means it is working, while state 1 means it fails. The CIMs for each node with different scenarios are listed in Table 3.8-3.10. For node P, its initial state is 0, the expected time it goes out this state is $\frac{1}{\lambda_P}$. The probability it enters state 11 is $\frac{p_1 \lambda_P}{\lambda_P} = p_1$ and the probability it enters state 12 is $\frac{p_2 \lambda_P}{\lambda_P} = p_2$. If it transits into state 11 or state 12, it will stay in that state forever. For node S, its initial state is 0. However, the state transition depends on its parent P. If P is in state 0, then S is dormant and evolves with failure rate $\alpha\lambda_s$; If P is in state 11, which means P fails and the switch activates S successfully, it will evolve with failure rate λ_s; If P is in state 12, which

means P fails and the switch fails to activate S, then S will stay in dormant state. S is non-reparable; once it enters the failure state, it will not go out. For node WSP, it will stay in state 0 if P is working or P is in state 11 and S is in state 0; it will transit into state 1 immediately if P is in state 12 or P is in state 11 and S is in state 1.

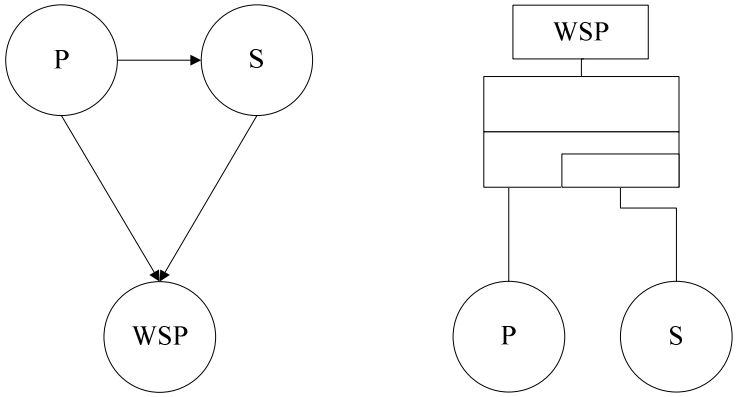


Figure 3.3: The WSP gate in FT and CTBN representation

Table 3.8: CIM for node P of WSP gate

State	P(0)	P(11)	P(12)
P(0)	$-\lambda_P$	$p_1\lambda_P$	$p_2\lambda_P$
P(11)	0	0	0
P(12)	0	0	0

Table 3.9: CIM for node S of WSP gate

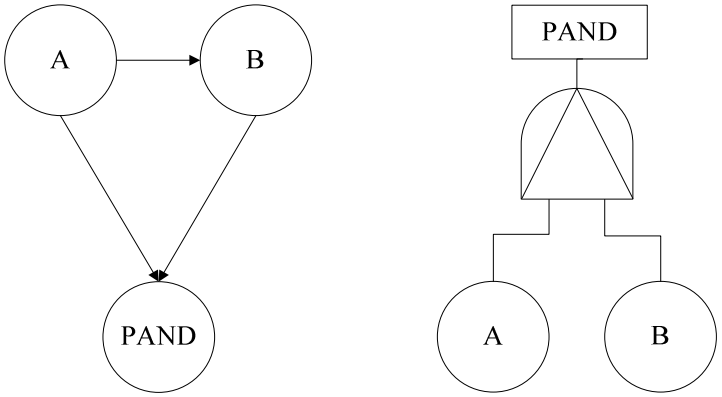
P(0)	State	S(0)	S(1)
	S(0)	$-\alpha\lambda_s$	$\alpha\lambda_s$
	S(1)	0	0
P(11)	State	S(0)	S(1)
	S(0)	$-\lambda_s$	$-\lambda_s$
	S(1)	0	0
P(12)	State	S(0)	S(1)
	S(0)	$-\alpha\lambda_s$	$\alpha\lambda_s$
	S(1)	0	0

Table 3.10: CIM for node WSP

{P(0), S(0)} {P(0), S(1)} {P(11), S(0)}	State	WSP(0)	WSP(21)
	WSP(0)	0	0
	WSP(21)	∞	$-\infty$
{P(11), S(1)} {P(12), S(0)} {P(12), S(1)}	State	WSP(0)	WSP(21)
	WSP(0)	$-\infty$	∞
	WSP(21)	0	0

3.2.3 PAND Gate

PAND gate reaches a failure state if and only if all of its input components have failed in a pre-assigned order. As an example, let us consider a situation where a PAND gate consists of 2 components A, B. If A fails before B, then a failure occurs; otherwise, there is not a failure happening. Node B has 3 states: state 0 means B is working; state 11 means A fails before B; state 12 means A fails after B. Node A and node PAND have 2 states 0 and 1. State 0 means they are working while state 1 means they fail. CIM of node A is the same as that in Table 3.5. CIM of node B and node PAND are listed in Table 3.11 and Table 3.12 respectively. For node B, if A is in state 0, the only way B can go is to state 12, which means B fails before A. However, if A is in state 1, obviously, A fails before B, B then will enter state 11. As for node PAND, it will keep in state 0 unless A and B both are in state 1, which indicates a failure.



CTBN: PAND Node **FT: PAND gate**

Figure 3.4: The PAND gate in FT and CTBN representation

Table 3.11: CIM for node B of PAND gate

A(0)	State	B(0)	B(11)	B(12)
	B(0)	$-\lambda_B$	0	λ_B
	B(11)	0	0	0
	B(12)	0	0	0
A(1)	State	B(0)	B(11)	B(12)
	B(0)	$-\lambda_B$	λ_B	0
	B(11)	0	0	0
	B(12)	0	0	0

Table 3.12: CIM for node PAND

{A(1), B(11)}	State	PAND(0)	PAND(1)
	PAND(0)	$-\infty$	∞
	PAND(1)	0	0
Others	State	PAND(0)	PAND(1)
	PAND(0)	0	0
	PAND(1)	0	0

3.2.4 PDEP Gate

In the PDEP gate, one trigger event T causes other dependent components to become unusable or inaccessible with probability $p_{dep} \leq 1$. As an example, let us consider a situation where the failure of a trigger component T will cause both

component A and B to fail with probability p_{dep}. The corresponding CTBN representation is shown in Fig 3.5. Node T has 3 states: 0, 11 and 12. State 0 means it is working; state 11 means it fails in failure modal 1 and this failure will cause both A and B fail; state 12 means it fails in failure modal 2 and this failure has no effect on A and B. Node A and B both have 2 states: 0 and 1. State 0 means it is working while state 1 means it fails. For node T, its initial state is 0 and it will transit to state 11 with probability p_{dep} or transit to state 12 with probability $1 - p_{dep}$. For node A, if T is working or T's failure has not impact on it (with probability p_{dep}), it will evolve with failure rate λ_A; otherwise, it will transit to failure state (state 1) immediately. Node B has the same CIMs as Node A has. The discussion above assumes that A and B are non-reparable. If A and B are reparable, since trigger event T can cause both A and B to fail, there are two cases in reality: case 1: they need repair action or case 2: they do not. Case 1 means that if A and B fail (due to trigger event T or fail by themselves), it takes $\frac{1}{\mu_A}$ or $\frac{1}{\mu_B}$ to repair them. Case 2 is that if A and B fail due to T, they do not need repair action. They are in good condition; they stop working just because T is malfunction. Once T is back to normal, they will continue to operate. However, if their failures are not caused by T, they fail by themselves, they still need repair action. For example, suppose T is power supply, A and B are two generators. If power supply (T) is down, A and B are down. However, they do not need repair action because they are in good condition. Once the power supply continues, they will continue to work again. It is easy to model case 1: if A or B are in state 1, they will transit back to state 0 with repair rate μ_A or μ_B. For case 2, where one more state is needed for A and B, they have 3 states: state 0 means they are working; state 11 means they fail in failure modal 1 and this failure is caused by T, they

do not need repair action in this failure; state 12 means they fail in failure modal 2 and this failure is caused by themselves, so this failure needs repair action. For detail CIMs, please refer to Table 3.14 and Table 3.15. In the Literature (Portinale, Raiteri et al. 2010), only case 1 is considered. In order to make a comparison with the results from the literature, for the rest of this chapter, unless specified, we refer to case 1.

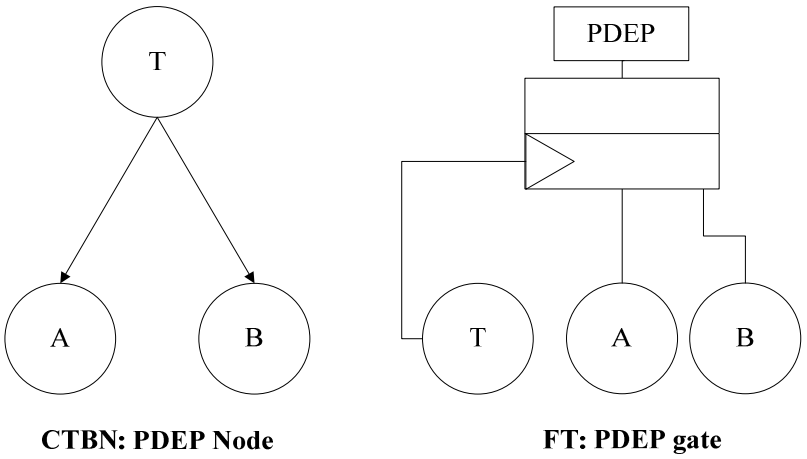


Figure 3.5: The PEDP gate in FT and CTBN representation

Table 3.13: CIM for node T of PDEP gate

State	T(0)	T(11)	T(12)
T(0)	$-\lambda_T$	$p_{dep} * \lambda_T$	$(1 - p_{dep}) * \lambda_T$
T(11)	0	0	0
T(12)	0	0	0

Table 3.14: CIM for node A of PDEP gate, non-reparable and reparable case 1

Non-reparable					Reparable, Case 1			
T(0) or T(12)	State	A(0)	A(1)		T(0) or T(12)	State	A(0)	A(1)
	A(0)	$-\lambda_A$	λ_A			A(0)	$-\lambda_A$	λ_A
	A(1)	0	0			A(1)	μ_A	$-\mu_A$
T(11)	State	A(0)	A(1)		T(11)	State	A(0)	A(1)
	A(0)	-∞	∞			A(0)	-∞	∞
	A(1)	0	0			A(1)	μ_A	$-\mu_A$

Table 3.15: CIM for node A of PDEP gate, reparable case 2

Reparable, Case 2				
T(0)	State	A(0)	A(11)	A(12)
	A(0)	$-\lambda_A$	0	λ_A
	A(11)	∞	-∞	0
	A(12)	μ_A	0	$-\mu_A$
T(11)	State	A(0)	A(11)	A(12)
	A(0)	-∞	∞	0
	A(11)	0	0	0
	A(12)	0	0	0
T(12)	State	A(0)	A(11)	A(12)
	A(0)	$-\lambda_A$	0	λ_A
	A(11)	0	0	0
	A(12)	μ_A	0	$-\mu_A$

3.3 Multi-state Interaction and CTBN Modeling

Multi-state interaction is similar to the WSP gate. The (Dynamic) Fault trees cannot model a multi-state system, thus there is not this kind of gate in DFTs. The difference between multi-state interaction and the WSP gate is that, in multi-state interaction, we do not distinguish components by primary and standby. The purpose of multi-state interaction is not to increase the redundancy. Instead, it models the case in which the failure distribution of one component depends on the working state of other components. In this sense, it looks more like the PDEP gate. However, the PDEP gate is to model common failure factor, which means that the failure of one component will bring down one or more components. In multi-state interaction, the component doesn't necessarily fail; it primarily focuses on how the degradation of one component will affect the others. For example, consider a system that consists of two components A and B in series with exponential failure distribution. A has multi-state: good, moderate and failed. B is binary state: good and failed. The failure rate of B depends on the state of A. If A is

in state good, the failure rate for B is $\lambda_{B,g}$; if A is in state moderate, the failure rate for B change to $\lambda_{B,m}$. The failure rates for A are $\lambda_{A,g}$ and $\lambda_{A,m}$ when it is in state good and moderate respectively.

The CTBN representation for multi-state interaction is shown in Figure 3.6. The CIM for node B is listed in Table 3.16.

Table 3.16: CIM for node B of Multi-state interactions

A(good)	State	B(good)	B(failed)
	B(good)	$-\lambda_{B,g}$	$\lambda_{B,g}$
	B(failed)	0	0
A(moderate)	State	B(good)	B(failed)
	B(good)	$-\lambda_{B,m}$	$\lambda_{B,m}$
	B(failed)	0	0
A(failed)	State	B(good)	B(failed)
	B(good)	$-\lambda_{B,m}$	$\lambda_{B,m}$
	B(failed)	0	0

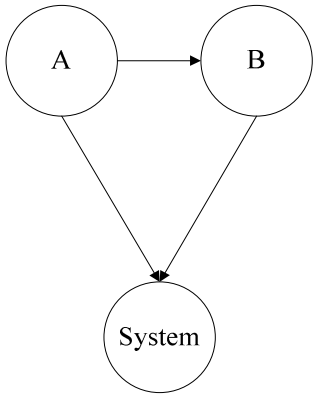


Figure 3.6: CTBN representation for multi-state interaction

3.4 CTBN Modeling of Repair Policies

In this subsection, we will present the modeling of repair policies within CTBN formalism through the CPU subsystem of a running example named the Cardiac Assist System (CAS) (Boudali, Crouzen et al. 2007; Portinale, Raiteri et al. 2010) described in Section 3.6.1. The CPU subsystem consists of two different CPUs: a primary CPU P with failure rate λ_P and a backup warm spare CPU B with failure rate λ_B and dormancy factor $\alpha_B = .5$. Both CPU are functionally dependent on cross switch CS (failure rate λ_{CS}) and system supervision SS (failure rate λ_{SS}). The failure of either CS or SS will force the failure of both CPUs and so the failure of the CPU subsystem. The dependencies among these components are modeled by dynamic gates and are shown as a part of DFT for the whole system in Figure 3.10. P and B are the input events of a WSP gate; this means that P is the principal component and in the case of failure, it will be substituted by B. The PDEP gate forces both P and B to fail if the event Trigger occurs (failure of either CS or SS). The corresponding CTBN representation of this subsystem is show in Figure 3.10.

3.4.1 CPU Subsystem with SGR Policy

In the case when SGR policy is employed, the corresponding CTBN representation is show in Figure 3.7. Compared with non-reparable case, there are extra green links (dash line) from CPU to reparable components (CS, SS, P and B), which means that the failure of the CPU subsystem will activate the repair process of each component. Node CPU has 3 states (0, 1 and 2). State 0 means it is working; state 1 means it is under repair and state 2 means the repair action is completed. The initial state of CPU is 0. When it fails, it enters state 1; the repair state and the node evolve following

the corresponding global repair rate μ_{SGR} (the expected transition time from state 1 to state 2 is $\frac{1}{\mu_{SGR}}$). When the repair is done, it will enter state 2, which is an instant state indicating that the global repair action is done, and all components under repair (nodes with green in-flow link) are immediately set to be functional again; and CPU will switch to state 0 instantly. Table 3.17 is the detailed CIM for node CPU. For node P, if CPU is in state 2, the repair process is completed. If it is in state 1, it will return back to state 0 immediately. Otherwise, it will stay in state 1. The detailed CIM is in Table 3.18. Other nodes (CS, SS and B) have the similar logic as node P.

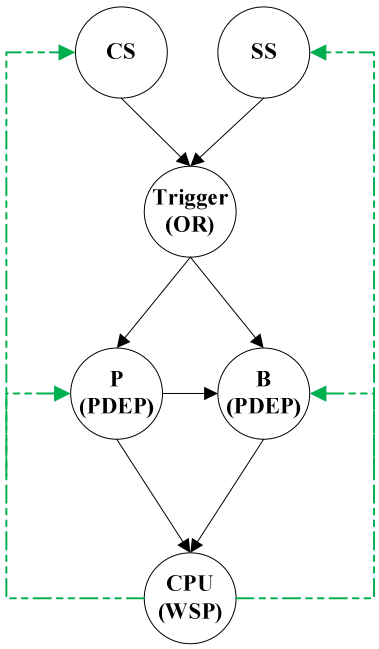


Figure 3.7: CTBN representation for SGR, SLR and SLR-min policy

Table 3.17: CIM for node CPU of CPU subsystem with SGR policy

{P(0), B(0)}, {P(0), B(1)}, {P(1), B(0)}	State	CPU(0)	CPU(1)	CPU(2)
	CPU(0)	0	0	0
	CPU(1)	∞	$-\infty$	0
	CPU(2)	∞	0	$-\infty$
{P(1), B(1)}	State	CPU(0)	CPU (1)	CPU (2)
	CPU (0)	$-\infty$	∞	0
	CPU (1)	0	$-\mu_{SGR}$	μ_{SGR}
	CPU (2)	∞	0	$-\infty$

Table 3.18: CIM for node P of CPU subsystem with SGR policy

{Trigger (0), CPU (0)}, {Trigger (0), CPU (1)}	State	P(0)	P(1)		{Trigger (1), CPU (0)}, {Trigger (1), CPU (1)}	State	P(0)	P(1)
	P(0)	$-\lambda_P$	λ_P			P(0)	$-\infty$	∞
	P(1)	0	0			P(1)	0	0
Trigger (0), CPU (2)	State	P(0)	P(1)		Trigger (1), CPU (2)	State	P(0)	P(1)
	P(0)	$-\lambda_P$	λ_P			P(0)	$-\infty$	∞
	P(1)	∞	$-\infty$			P(1)	∞	$-\infty$

3.4.2 CPU Subsystem with SLR Policy

In the case when SLR policy is employed, the corresponding CTBN representation is the same as that of SGR shown in Figure 3.7. However, node CPU only has two states (0 it is working and1 it fails), the CIM is listed in Table 3.19. Nodes CS, SS, P and B all have 3 states (0, 1 and 2). State 0 means that the component is working; state 1 means that it fails and state 2 means it is being repaired. For node P, its repair process is activated by the failure of CPU. If CPU is in state 1 and P is in state 1, P will transits to state 2 immediately and evolves with repair rate μ_P. The detailed CIM of P is listed in Table 3.20. Other nodes (CS, SS and B) have the similar logic as node P.

Table 3.19: CIM for CPU of CPU subsystem with SLR policy

{P(0), B(0)} {P(0), B(1)} {P(1), B(0)} {P(0), B(2)} {P(2), B(0)}	State	CPU (0)	CPU (1)
	CPU (0)	0	0
	CPU (1)	∞	$-\infty$
{P(1), B(1)} {P(1), B(2)} {P(2), B(1)} {P(2), B(2)}	State	CPU (0)	CPU (1)
	CPU (0)	$-\infty$	∞
	CPU (1)	0	0

Table 3.20: CIM for P of CPU system with SLR policy

Trigger (0), CPU (0)	State	P(0)	P(1)	P(2)		Trigger (1), CPU (0)	State	P(0)	P(1)	P(2)
	P(0)	$-\lambda_P$	λ_P	0			P(0)	$-\infty$	∞	0
	P(1)	0	0	0			P(1)	0	0	0
	P(2)	μ_P	0	$-\mu_P$			P(2)	μ_P	0	$-\mu_P$
Trigger (0), CPU (1)	State	P(0)	P(1)	P(2)		Trigger (1), CPU (1)	State	P(0)	P(1)	P(2)
	P(0)	$-\lambda_P$	λ_P	0			P(0)	$-\infty$	∞	0
	P(1)	0	$-\infty$	∞			P(1)	0	$-\infty$	∞
	P(2)	μ_P	0	$-\mu_P$			P(2)	μ_P	0	$-\mu_P$

3.4.3 CPU Subsystem with SLR-min Policy

In the case when SLR-min policy is employed, the corresponding CTBN representation is the same as that of SGR in Figure 3.7. The CIM of CPU is similar to that of SLR case, which is listed in Table 3.21. Node CS, SS, P and S all have 2 states (0 and 1). State 0 means they are working, and state 1 means they fails or are been repaired. For node P, when CPU is in state 0 (CPU returned back to working state), if P is in state 1, it will stay in state 1 (with repair rate 0); this models the situation when the subsystem is functional, the repair action will stop. Other nodes (CS, SS and B) have the similar logic as node A.

Table 3.21: CIM for CPU of CPU subsystem with SLR-min policy

{P(0), B(0)}, {P(1), B(0)}, {P(0), B(1)}	State	CPU (0)	CPU (1)
	CPU(0)	0	0
	CPU (1)	∞	$-\infty$
{P(1), B(1)}	State	CPU (0)	CPU (1)
	CPU (0)	$-\infty$	∞
	CPU (1)	0	0

Table 3.22: CIM for P of CPU subsystem with SLR-min policy

{Trigger (0), CPU (0)}	State	P(0)	P(1)	{Trigger (1), CPU (0)}	State	P(0)	P(1)
	P(0)	$-\lambda_P$	λ_P		P(0)	$-\infty$	∞
	P(1)	0	0		P(1)	0	0
{Trigger (0), CPU (1)}	State	P(0)	P(1)	{Trigger (1), CPU (1)}	State	P(0)	P(1)
	P(0)	$-\lambda_P$	λ_P		P(0)	$-\infty$	∞
	P(1)	μ_P	$-\mu_P$		P(1)	μ_P	$-\mu_P$

3.4.4 CPU Subsystem with CR Policy

When CR policy is applied to the CPU subsystem, there is no extra links and nodes needed. The corresponding CTBN representation is shown in Figure 3.8. The net representation is the same as the non-reparable case; however, CIM of each node is different from that of non-reparable case. Table 3.23 is the CIM for P with non-reparable case; Table 3.24 is the CIM for P with CR policy. Notice that the difference between them is when P is in state 1, it will stay there in non-reparable case, while it will evolve with repair rate μ_P in CR policy case. Other nodes (CS, SS and B) have the similar logic as node P. CIMs for node Trigger and node CPU are the same as that of non-reparable case.

Table 3.23: CIM for P of CPU subsystem with CR policy, non-reparable case

Trigger(0)	State	P(0)	P(1)
	P(0)	$-\lambda_P$	λ_P
	P(1)	0	0
Trigger(1)	State	P(0)	P(1)
	P(0)	$-\infty$	∞
	P(1)	0	0

Table 3.24: CIM for P of CPU subsystem with CR policy

Trigger(0)	State	P(0)	P(1)
	P(0)	$-\lambda_P$	λ_P
	P(1)	μ_P	$-\mu_P$
Trigger(1)	State	P(0)	P(1)
	P(0)	$-\infty$	∞
	P(1)	μ_P	$-\mu_P$

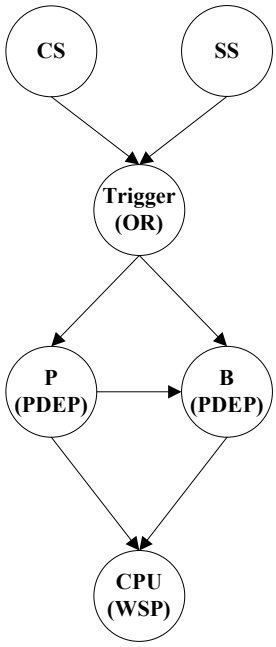


Figure 3.8: CTBN representation for CR policy

3.4.5 CPU Subsystem with CR-limit Policy

When CR-limit policy is applied, since the repair crews are limited, there should be a control center which can assign a repair crew to a down component, and reclaim the repair crew from a component when the repair process is completed. In the CPU

subsystem example, one more node M is added and plays this role. Node M has two way links connected with reparable components (CS, SS, P and B). If a component fails, and node M has available repair crew, it will assign this repair crew to this component; if there is not repair crew available, it will keep the component on hold. When a repair process is completed, the component will report to node M and node M will reclaim the repair crew from that component. The states of node M are dependent on the total number of repair crews available. Suppose there are n repair crews available, then node M has $n+1$ states $(0: n)$. $State = k$ $(k \leq n)$ means that there are k repair crews available. As an example, lets support $k = 2$, then node M has 3 states: 0, 1 and 2. And the initial state is 2, which means it has 2 repair crews available at the beginning. Nodes CS, SS, P and B all have 3 states: 0, 1 and 2. State 0 means that the component is working; state 1 means that the component fails and state 2 means that it is being repaired. The corresponding CTBN representation is shown in Figure 3.9. Table 3.25 is the CIM for node M with 2 repair crews. Notice that the initial state of node M is state 2. When there is not a component in repair state (state 2), node M will stay in state 2. If one component fails, node M will transits into state 1 immediately and stay there (of course, this component will enter repair state 2). If the component finishes its repair process (back to state 0), node M will transits back to state 2 instantly. However, before the 1st failed component finishes it repair, if one more component fails, node M will transits from state 1 to state 0. At this time, even more components fail, node M will still stick in state 0, because there is not an available repair crew for assigning, and the failed components stay in a failure state (state 1) and wait for the repair crew. Table 3.26 is the CIM for node P. We can see that, when node M is in state 0, if P is in state 1, it will stay in state 1. However, if node

M is in state 1 or state 2, P will enter state 2 instantly, which mean that it is assigned a repair crew and can enter repair process. Other nodes (CS, SS and B) have the similar logic as node A.

If we want to model this policy using Markov Chains, we can denote each state as $(CS, SS, P, B|R_1, R_2)$; R_1 and R_2 are the 1st and 2nd repair crew. The number of states we need for the case with 2 repair facilities is $2^4 \times C_4^2 = 2^4 \times \frac{4\times3}{2} = 96$. For the case with 3 or 4 repair facilities, the number of states need is $2^4 \times C_4^3 = 2^4 \times C_4^4 = 192$. Due to the natural limitation of DBNs, they cannot model this repair policy.

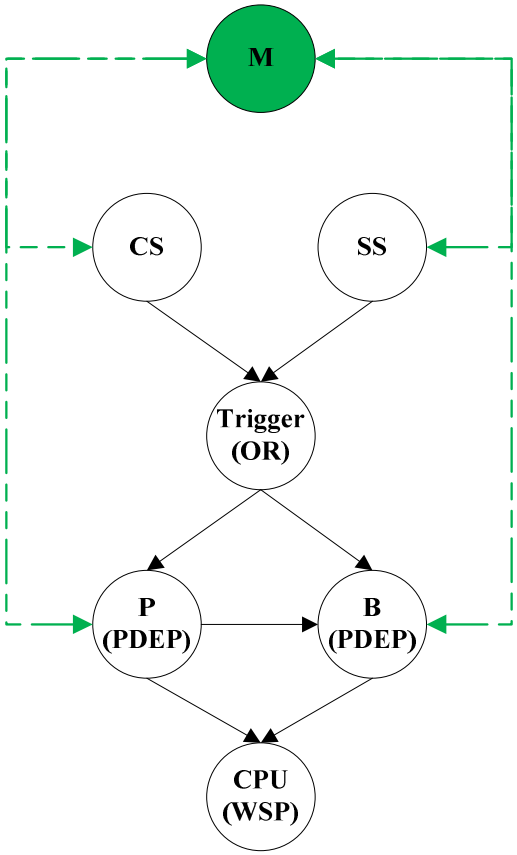


Figure 3.9: CTBN representation for CR-limit policy

Table 3.25: CIM for node M of CPU subsystem with 2 repair facilities

0 component (CS, SS, P, B) is in state 2	State	M(0)	M(1)	M(2)
	M(0)	$-\infty$	0	∞
	M(1)	0	$-\infty$	∞
	M(2)	0	0	0
1 component (CS, SS, P, B) is in state 2	State	M(0)	M(1)	M(2)
	M(0)	$-\infty$	∞	0
	M(1)	0	0	0
	M(2)		∞	$-\infty$
2 or more than 2 components (CS, SS, P, B) are in state 2	State	M(0)	M(1)	M(2)
	M(0)	0	0	0
	M(1)	∞	$-\infty$	0
	M(2)	∞	0	$-\infty$

Table 3.26: CIM for node P of CPU subsystem CR-limited policy

{Trigger (0), M (0)}	State	P(0)	P(1)	P(2)	{Trigger (1), M (0)}	State	P(0)	P(1)	P(2)
	P(0)	$-\lambda_P$	λ_P	0		P(0)	$-\infty$	∞	0
	P(1)	0	0	0		P(1)	0	0	0
	P(2)	μ_P	0	$-\mu_P$		P(2)	μ_P	0	$-\mu_P$
{Trigger (0), M (1)}, {Trigger (0), M (2)}	State	P(0)	P(1)	P(2)	{Trigger (1), M (1)}, {Trigger (1), M (2)}	State	P(0)	P(1)	P(2)
	P(0)	$-\lambda_P$	λ_P			P(0)	$-\infty$	∞	0
	P(1)	0	$-\infty$	∞		P(1)	0	$-\infty$	∞
	P(2)	μ_P	0	$-\mu_P$		P(2)	μ_P	0	$-\mu_P$

3.5 Case Studies:

In this subsection, the CTBN method is applied to three systems. The examples presented, are based on real-life systems. The Cardiac system is a typical example with dynamic behavior, which is broadly used in literatures (Boudali, Crouzen et al. 2007; Portinale, Raiteri et al. 2010) for evaluating the reliability modeling methods. The purpose of implementing the CTBN method to Cardiac system is two folders. First, we can show how to map the (Dynamic) Fault trees into CTBNs; second, we can evaluate the accuracy of CTBNs by comparing it with the results of other methods (Markov Chain,

Dynamic Bayesian Networks, Petri-net) taken from literatures (Montani, Portinale et al. 2005; Portinale, Raiteri et al. 2010). The second example is a ground vehicle system, which is used for a demonstration of how the CTBNs model multi-state interactions. The last example is based on the second example. It is a fleet of vehicles system. The purpose of this example is to demonstrate how the CTBNs model the repairable system with limited shared repair crews, which cannot be modeled by Dynamic Bayesian Networks.

3.5.1 Cardiac System

The Cardiac Assist System (CAS) model is taken from literatures (Boudali, Crouzen et al. 2007; Portinale, Raiteri et al. 2010) and is based on a real-world system. It consists of three separate subsystems: the CPU subsystem, the motor subsystem and the pump subsystem. The failure of either one of the above subsystem will cause the whole system to fail. As we introduce in Section 3.5, the CPU subsystem has two different CPUs: a primary CPU P and a backup warm spare CPU B (with dormancy failure rate $\alpha_B = 0.5$). Both CPUs are functionally dependent a cross switch CS and a system supervision SS. The failure of either CS or SS will force both CPUs to fail. The motor subsystem also has two motors: a primary MA and a cold spare MB. The switching component MS will activate the spare motor when the primary one fails on condition that MS is still working. This means that if MS fails before MA, MS cannot effetely turn on MB and then the whole subsystem fails. Finally, there are three pumps in the pump subsystem. Two primary pumps PA and PB running in parallel and a cold shared pump PS. The pump subsystem fails if all three pumps fail. The failure rate for each component is listed in Table 3.27 and the DFT for the whole CAS system is shown in Fig. 3.10. The CTBN representation corresponding to the DFT in Figure 3.10 is shown in Figure 3.11.

In this thesis, all numerical experiments with CTBNs are performed with CTBN-RLE (Shelton, Fan et al. 2010) developed by Shelton and his team. And for all ∞ in CIMs, we make them equal to a larger number; here we choose 10^{10}.

Table 3.27: Failure rates for the CAS system component

Component	Subsystem	Failure rate
Primary CPU-P	CPU	$\lambda_P = .5E-3$
Backup CPU-B	CPU	$\lambda_B = .5\mathrm{E}-3$
Cross switch-CS	CPU	$\lambda_{CS} = .2E-3$
System supervision-SS	CPU	$\lambda_{SS} = .2E-3$
Primary Motor-MA	MOTOR	$\lambda_{MA} = 1E-3$
Cold spare Motor-MB	MOTOR	$\lambda_{MB} = 1E-3$
Switching Component- MS	MOTOR	$\lambda_{MS} = .01E-3$
Primary pump-PA	PUMP	$\lambda_{PA} = 1E-3$
Primary pump-PB	PUMP	$\lambda_{PB} = 1E-3$
Shared pump-PS	PUMP	$\lambda_{PS} = 1E-3$

In order to validate the correctness of CTBNs modeling method in RAM analysis, first, we consider the system as non-reparable and estimate the unreliability of the system with the mission time from 100 to 1000 hours (with time step 100), and then compare them with the results from paper (Montani, Portinale et al. 2005), which are calculated by MC-based method Galileo. Table 3.28 shows the comparison between them. Since Galileo method is based on continuous time Markov chain analysis, we consider the results obtained from it as true value. From the Table we can see that the results estimated from CTBNs are almost identical to those of Galileo, which validates the correctness of CTBN modeling method. The average of Relative Deviation Percentage (RDP) is as small as 2.3 %, where RDP is calculated by $RDP = \frac{\sum_i^n \left| \frac{Value(CTBN_i) - Value(Galileo_i)}{Value(Galileo_i)} \right|}{n} \times 100$. The estimation of the reliability using

CTBNs is a filtering task which queries the node System (Top Event) with empty observation in in other nodes (Basic Event).

Table 3.28: The unreliability of CAS estimated by CTBNs

Hours	CTBN	Galileo	Deviation	RDP
100	0.041633	0.046034	-0.0044	9.56%
200	0.09786	0.103223	-0.00536	5.20%
300	0.163169	0.169335	-0.00617	3.64%
400	0.236178	0.24148	-0.0053	2.20%
500	0.317489	0.31671	0.000779	0.25%
600	0.391319	0.392059	-0.00074	0.19%
700	0.469853	0.465402	0.004451	0.96%
800	0.534843	0.534898	-5.5E-05	0.01%
900	0.600071	0.59931	0.000761	0.13%
1000	0.663378	0.657889	0.005489	0.83%
			Mean RDP	2.30%

In the reparable case, in order to compare results of CTBN with those of other methods taken from the paper (Portinale, Raiteri et al. 2010), we assume the same setting as that in Portinale's paper. Only the CPU subsystem is reparable: first only the two CPUs are applied with CR policy, and then extended to Trigger (CS and SS) with the same policy. Their repair rates are all equal to 0.1 ($\mu_{CS} = \mu_{SS} = \mu_P = \mu_B = .1$). The *DRPFTproc* (Bobbio and Raiteri 2004)is a Stochastic Petri Nets based reliability analysis tool and the *RADyBaN* (Portinale, Raiteri et al. 2010)is a DBN-based method. Table 3.29 shows the comparison of results obtained from these methods and CTBNs. From the Table we can see that the agreement between them.

Table 3.29: The unavailability (1- point availability) of CAS obtained by CTBNs with CR policy

Hours	CPU repair			CPU+Trigger repair		
	CTBN	**DRPFTproc** (Bobbio and Raiteri 2004)	**RADyBaN** (Portinale, Raiteri et al. 2010)	**CTBN**	**DRPFTproc** (Bobbio and Raiteri 2004)	**RADyBaN** (Portinale, Raiteri et al. 2010
100	0.04883	0.044330	0.044283	0.007782	0.011282	0.011243
200	0.094698	0.095198	0.096916	0.025652	0.027652	0.027566
300	0.160094	0.155094	0.156659	0.057963	0.054963	0.054837
400	0.221637	0.220137	0.221550	0.088617	0.092117	0.091957
500	0.291619	0.288119	0.2893821	0.136437	0.137437	0.137252
600	0.356405	0.356905	0.358023	0.184482	0.188982	0.188779
700	0.421624	0.424624	0.425606	0.248771	0.244771	0.244558
800	0.490268	0.489768	0.490624	0.306446	0.302946	0.302729
900	0.556211	0.551211	0.551952	0.360365	0.361865	0.36165
1000	0.607691	0.608191	0.608829	0.421148	0.420148	0.419939

Table 3.30: The unavailability (1- point availability) of CAS obtained by CTBNs with CR-limit policy

Hours	CR	CR-limit			
		4 Crews	**3 Crews**	**2 Crews**	**1 Crew**
100	0.04883	0.04935022	0.04987309	0.05495257	0.107054
200	0.094698	0.09523264	0.09671853	0.09862899	0.16194
300	0.160094	0.16017298	0.16141105	0.16729006	0.174128
400	0.221637	0.22171244	0.22239923	0.22279578	0.294837
500	0.291619	0.29245792	0.29389953	0.30237777	0.334138
600	0.356405	0.35675185	0.35805307	0.3583714	0.3813
700	0.421624	0.42220654	0.42272338	0.42998631	0.506544
800	0.490268	0.49037719	0.49089903	0.49673556	0.54792
900	0.556211	0.55698206	0.55789716	0.56496994	0.617032
1000	0.607691	0.6081461	0.60920054	0.61310904	0.652559

Table 3.30 lists the unavailability estimated by CTBNs when the number of repair crew varies. The 2nd column (CR) is the unlimited repair crew case; the 3rd to 6th columns are the cases when number of repair crew varies from 4 to 1. The instantaneous availability decreases as the number of repair facility decreases. In this example, CR-L4

should have the same result as that of CR because there are only 4 components in this subsystem. Having 4 repair crews available actually is the case of unlimited repair crews. Results (the 2nd and 3rd column) in the table above show the agreement between them.

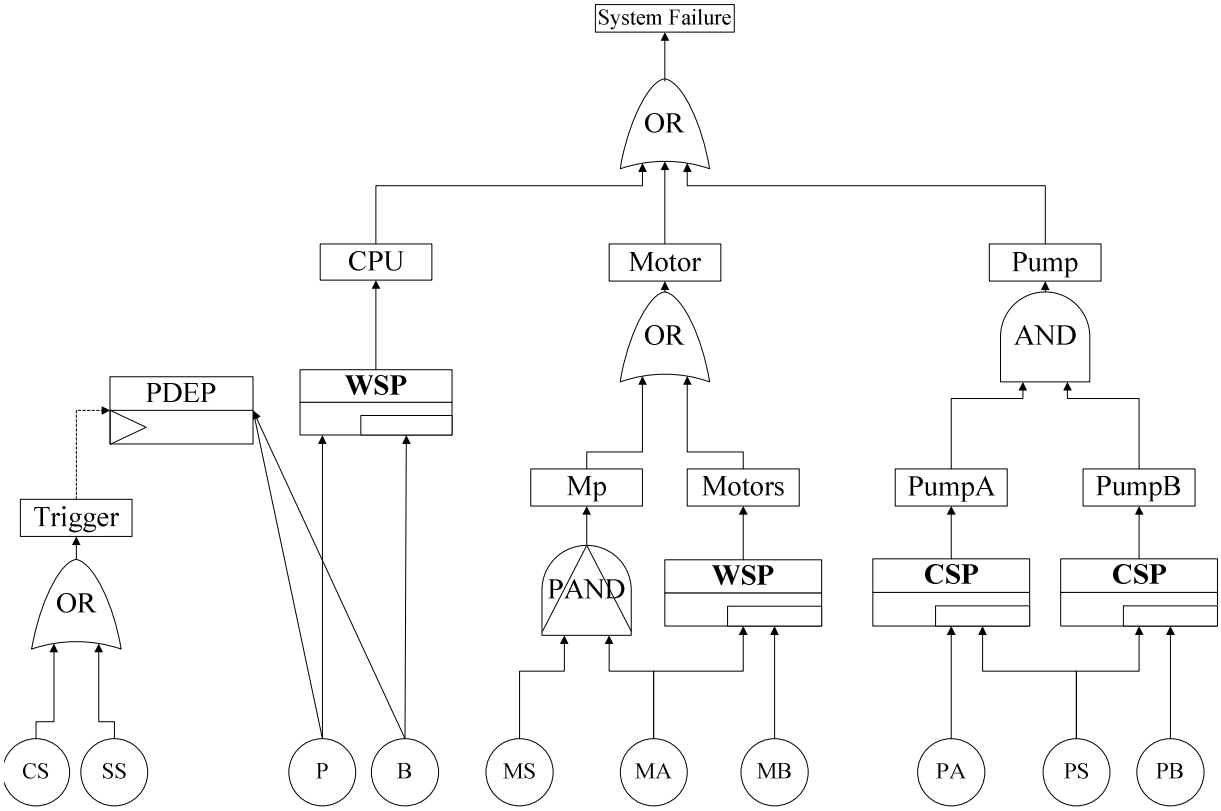


Figure 3.10: The DFT model of CAS

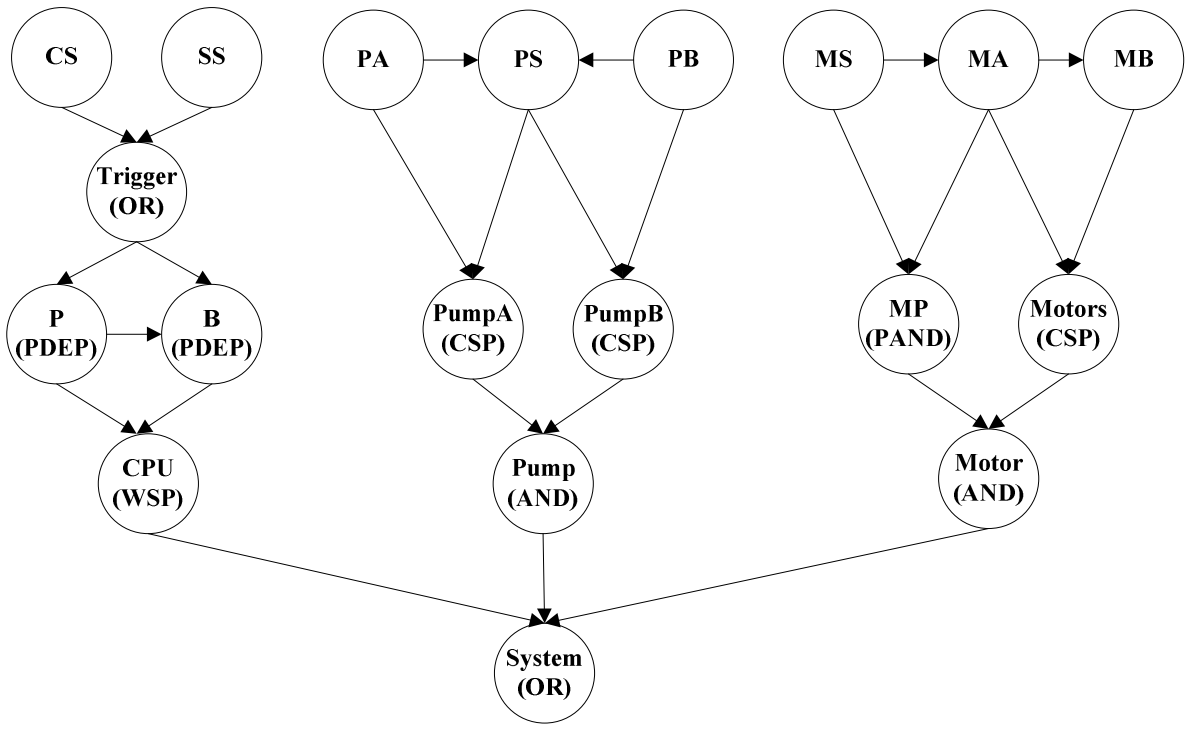


Figure 3.11: The CTBN representation corresponding to the DFT in Figure 3.10

3.5.2 Ground Vehicle System

Let's look at a ground vehicle. It consists of four main subsystems in series: the body subsystem, the chassis subsystem, the powertrain subsystem and the electrical subsystem. Let's focus more detail on the chassis subsystem and the powertrain subsystem. The chassis subsystem has four components in series: suspensions, the brakes, the wheels & tires, and the axles. The powertrain subsystem has three components in series: the engine, the transmission and the cooling subsystem. Although all components and subsystems are in series structure, there are some interactions among these components and subsystems. For example, the operating condition of suspension has a great impact on the wheels & tires, the body and the axles (Clifton 1990). With the suspension degradation, such as the shock absorber being stuck (less or no movement), the wheels & tires, the axles and the body would bear more energy from a shock and

would result in damages in these components, which will accelerate their failures. Another example of interaction is between the cooling subsystem and the engine (Walter 2001). Commonly the perfect operating temperature for an engine is around 180F. The degradation of cooling system will lead to engine overheating or being too cold. One common effect of overheating or cooling to an engine is that the abnormal temperature will cause the engine lubricating oil to thin or become too dense and result in poor lubrication, which will cause damage to the engine and accelerate its failure. Thus, we assume that both the suspension and cooling system have multiple working states: good, moderate and failed. All other subsystems have binary states: working and failed. We also assume that all failure time and repair time follow exponential distribution. The failure rates $\lambda_{.}$ and repair rates $\mu_{.}$ for each subsystem in different states are shown in Table 3.31. The original failure rate for each subsystem is failures per mile. In order to make the unit consistent with repair rate (per hours), we transfer failure rate from failures per mile into failures per hour by using average speed factor 50 miles per hour. It is not easy to define the failure of a body subsystem, and this failure is not common except in car accidents. Thus in this study we don't consider the failure of a body subsystem, and just put its failure rate and repair rate as 0. There are more than one suspension, wheel & tire in a vehicle, the failure rate and repair rate for these two subsystems are aggregated from independent multiple suspensions, wheels & tires.

Table 3.31: Failure and repair rate for each subsystem of ground vehicle example

Subsystem	Component	State	Failure Rate or Repair Rate	
Chassis (CH)	Brakes(BR)		SU(good)	SU(moderate)
		Good	$\lambda_{br,g} = .5e-3$	$\lambda_{br,m} = .8e-3$
		Failed	μ_{br} =.5	
	Wheels & Tires(WT)		SU(good)	SU(moderate)
		Good	$\lambda_{wt,g} = .16e-3$	$\lambda_{wt,m} = .2e-3$
		Failed	$\mu_{.wt} = 1.2$	
	Suspension(SU)	Good	$\lambda_{s,g} = .5e-4$	
		Moderate	$\lambda_{s,m} = .1e-3$	
		Failed	μ_s=3	
	Axles(AX)		SU(good)	SU(moderate)
		Good	$\lambda_{ax,g} = .5e-4$	$\lambda_{ax,m} = .7e-4$
		Failed	μ_{ax} =8	
Powertrain (PT)	Engine(EG)		CO(good)	CO(moderate)
		Good	$\lambda_{en,g} = .23e-4$	$\lambda_{en,m} = .35e-4$
		Failed	$\mu_{en} = 10$	
	Transmission(TR)	Good	$\lambda_{tr} = .83e-4$	
		Failed	μ_{tr} =4	
	Cooling(CO)	Good	$\lambda_{c,g}$ =.75e-4	
		Moderate	$\lambda_{c,m}$ =.15e-3	
		Failed	μ_c =3.5	
Electrical subsystem(EL)	Electrical subsystem(EL)	Good	λ_{el} =.12e-3	
		Failed	μ_{el} =.8	
Body (BO)	Body (BO)	Good	λ_{bo} =0	
		Failed	μ_{bo} =0	

The (Dynamic) Fault trees are only limited to modeling binary state systems. The CTBNs can easily handle a multi-state component. The CTBN representation for the ground vehicle system is shown in Figure 3.12.

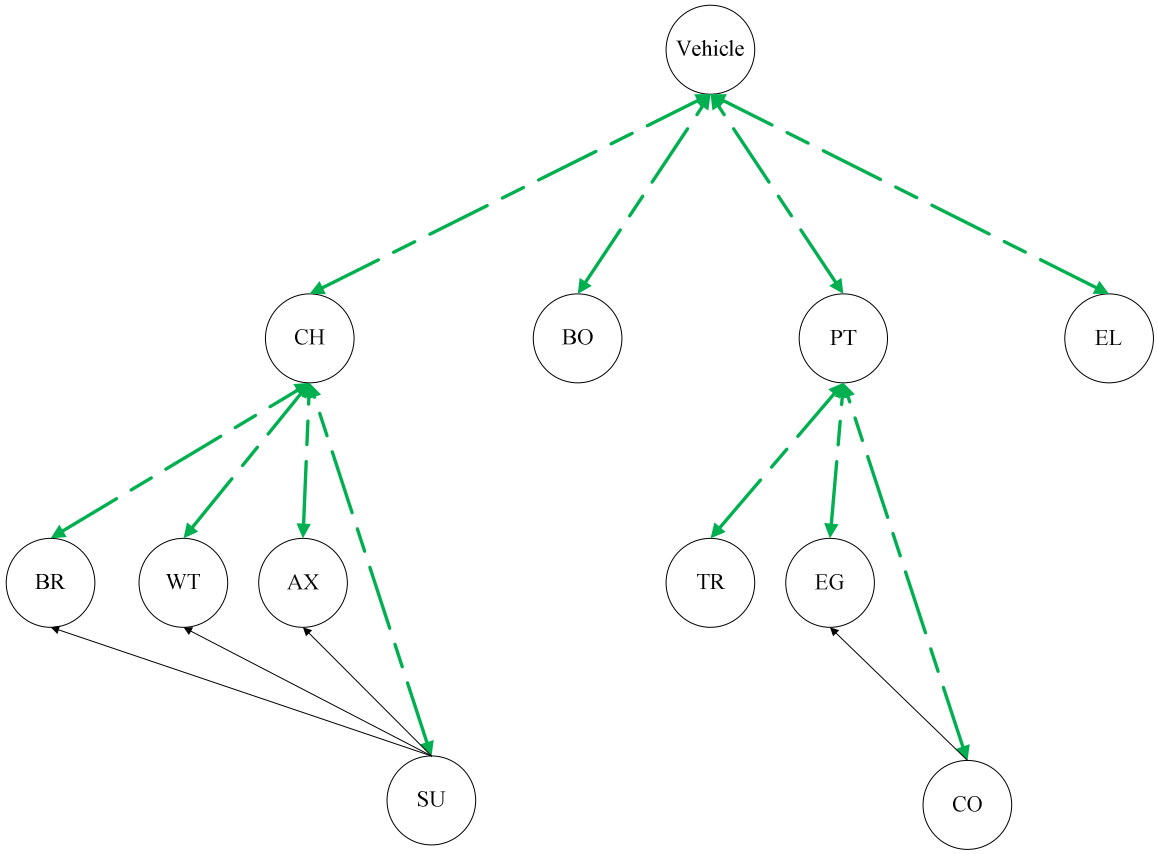


Figure 3.12: The CTBN representation of ground vehicle system

There is no redundancy in this system; all subsystems and components are in series structures. As discussed in the previous subsection, the subsystem repair policies are identical to the component repair policies in this system. Furthermore, without redundancy, if one component fails, the whole system fails. Then other components stop operating and cannot fail, but they are still in good state (case 2 of PDEP gate, Section 3.2.4). Thus only one component would fail each time and one repair crew would be enough. Based on the discussion above, in this example we just need to consider the CR policy. The CIMs for some nodes are shown for Table 3.32 to Table 3.34. Nodes CO, TR, EL have similar CIMs with node SU; Nodes WT, BO, AX and EG have similar CIMs with node BR; Node PT and Vehicle have similar CIMs with node CH. The double arrow dash green links in this example work in this way: when a component fails, it will bring

down the subsystem and finally bring down the whole vehicle. On the other hand, when the subsystem/system fails, if the component is in failure state, it will start the repair procedure automatically; if it is in other states, it will stay in those states. These model the situation when a subsystem is brought down by one component. Only the failed component needs repair, other components would stay in their current state due to the facts that when the subsystem fails, other components stop operating. All nodes are in good state at the beginning.

Table 3.32: CIM for node SU for ground vehicle example

CH(good)	State	SU(good)	SU(moderate)	SU(failed)
	SU(good)	$-\lambda_{su,g}$	$\lambda_{su,g}$	0
	SU(moderate)	0	$-\lambda_{su,m}$	$\lambda_{su,m}$
	SU(failed)	μ_{su}	0	$-\mu_{su}$
CH(failed)	State	SU(good)	SU(moderate)	SU(failed)
	SU(good)	0	0	0
	SU(moderate)	0	0	0
	SU(failed)	μ_{su}	0	$-\mu_{su}$
CH(standby)	State	SU(good)	SU(moderate)	SU(failed)
	SU(good)	0	0	0
	SU(moderate)	0	0	0
	SU(failed)	μ_{su}	0	$-\mu_{su}$

Table 3.33: CIM for node BR of ground vehicle example

CH(good)	SU(good)	State	BR(good)	BR(failed)
		BR(good)	$-\lambda_{br,g}$	$\lambda_{br,g}$
		BR(failed)	μ_{br}	$-\mu_{br}$
	SU(moderate)	State	B(good)	B(failed)
		BR(good)	$-\lambda_{br,m}$	$\lambda_{br,m}$
		BR(failed)	μ_{br}	$-\mu_{br}$
	SU(failed)	State	B(good)	B(failed)
		BR(good)	$-\lambda_{br,m}$	$\lambda_{br,m}$
		BR(failed)	μ_{br}	$-\mu_{br}$
CH(failed)	SU(good)	State	BR(good)	BR(failed)
		BR(good)	0	0
		BR(failed)	μ_{br}	$-\mu_{br}$
	SU(moderate)	State	B(good)	B(failed)
		BR(good)	0	0
		BR(failed)	μ_{br}	$-\mu_{br}$
	SU(failed)	State	B(good)	B(failed)
		BR(good)	0	0
		BR(failed)	μ_{br}	$-\mu_{br}$
CH(standby)	SU(good)	State	BR(good)	BR(failed)
		BR(good)	0	0
		BR(failed)	μ_{br}	$-\mu_{br}$
	SU(moderate)	State	B(good)	B(failed)
		BR(good)	0	0
		BR(failed)	μ_{br}	$-\mu_{br}$
	SU(failed)	State	B(good)	B(failed)
		BR(good)	0	0
		BR(failed)	μ_{br}	$-\mu_{br}$

Table 3.34: CIM for node CH for ground vehicle example

Vehicle(good)	BR, WT, AX and SU are all in good state	State	CH(good)	CH(failed)	CH(repair)
		CH(good)	0	0	0
		CH(failed)	∞	$-\infty$	0
		CH(repair)	∞	0	$-\infty$
	One of BR, WT, AX or SU fails	State	CH(good)	CH(failed)	CH(repair)
		CH(good)	$-\infty$	∞	0
		CH(failed)	0	0	0
		CH(repair)	0	0	0
Vehicle(failed)	BR, WT, AX and SU are all in Good state	State	CH(good)	CH(failed)	CH(repair)
		CH(good)	0	0	0
		CH(failed)	∞	$-\infty$	0
		CH(repair)	0	0	0
	One of BR, WT, AX or SU fails	State	CH(good)	CH(failed)	CH(repair)
		CH(good)	0	0	0
		CH(failed)	0	$-\infty$	∞
		CH(repair)	0	0	0

First, we perform the standard reliability analysis (there is no repair action, put all repair rates as 0) using Dynamic Bayesian Networks and Continuous Time Bayesian Networks. The reliability of the ground vehicle at 10000 miles (200 hours) is .8296 from DBNs and .8311 from CTBNs respectably. Next, we will take the repair actions into consideration and perform the availability analysis. As we discussed in the previous section, in this ground vehicle example, the mean availability is a more practical metric than point availability because it indicates how much proportion of time the vehicle is operational in a mission period. By using CTBNs, the expected up time is 199.74 hours in the first 200 hours. The mean availability for this 200 hours mission period is $\bar{A} = \frac{199.74}{200} = .9987$. For DBNs, there is not a direct inference algorithm for calculating the

mean availability. However, we can estimate the point availability for each hour of these 200 hours and get the mean from them. The mean availability from DBNs is .9952 for the first 200 hours. We can see that these two results are close to each other. When the mission miles increase to 20000 miles (400 hours), the reliability and mean availability given by CTBNs are $R = .6872$ and $\bar{A} = \frac{399.46}{400} = .9986$; and the results given by DBNs are $R = .6869$ and $\bar{A} = .9947$ respectely. From the results above, we can see that when the mission time increase from 200 hours to 400 hours, the reliability decrease from .82 to .68, while the mean availability is almost stable, both are around .99.

3.5.3 A Fleet of Vehicles

This is a practical example taken from our industrial partner. Consider a ground combat team consisting of a fleet of ground combat vehicles with different functionalities. For example, the Stryker CV (Command Vehicle) (GDLS 2011), the Stryker MGS (Mobile Gun System), the Stryker ICV (Infantry Carrier Vehicle), and the Stryker MEV (Medical Evacuation Vehicle) et al. Each vehicle is a system and the ground combat team has limit number of repair crews shared by these vehicles. For example in Figure 3.13, the vehicles ICV, MGS, MEV and CV form a ground combat team. And assume that there are two repair crews in this team. The problem is to estimate the mean availability per vehicle in this ground combat team. It is calculated in this way: given a mission time T, we estimate the up time for the four vehicles in the team and then get the average up time per vehicle. Finally it is divided by mission time so we can get the mean availability per vehicle for this combat team. For security purpose and the proprietary nature of the

data, in this study, we use the ground vehicle system in Section 3.5.2 to denote the combat vehicles in this ground combat team example and do the demonstration analysis.

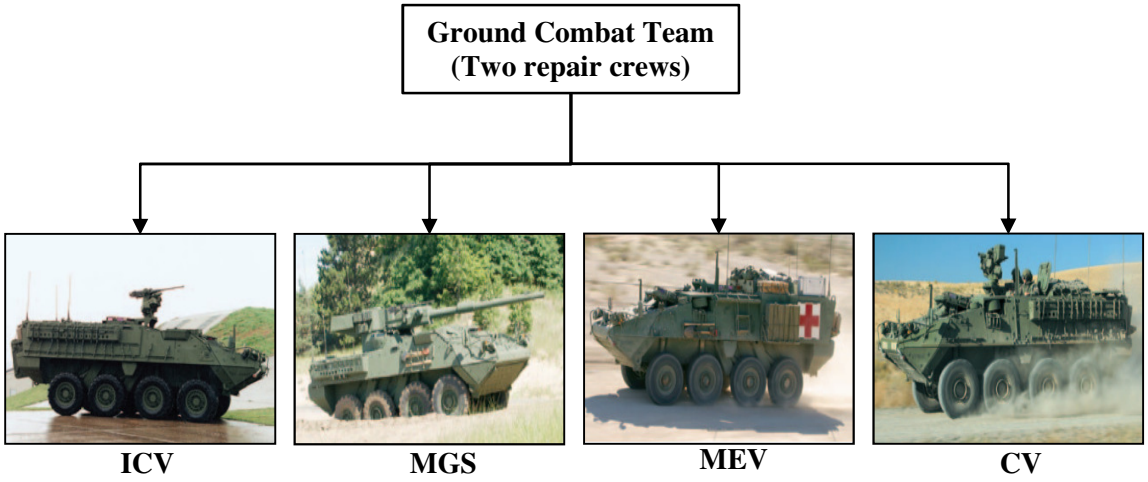


Figure 3.13: The ground combat team, consists of a fleet of vehicles

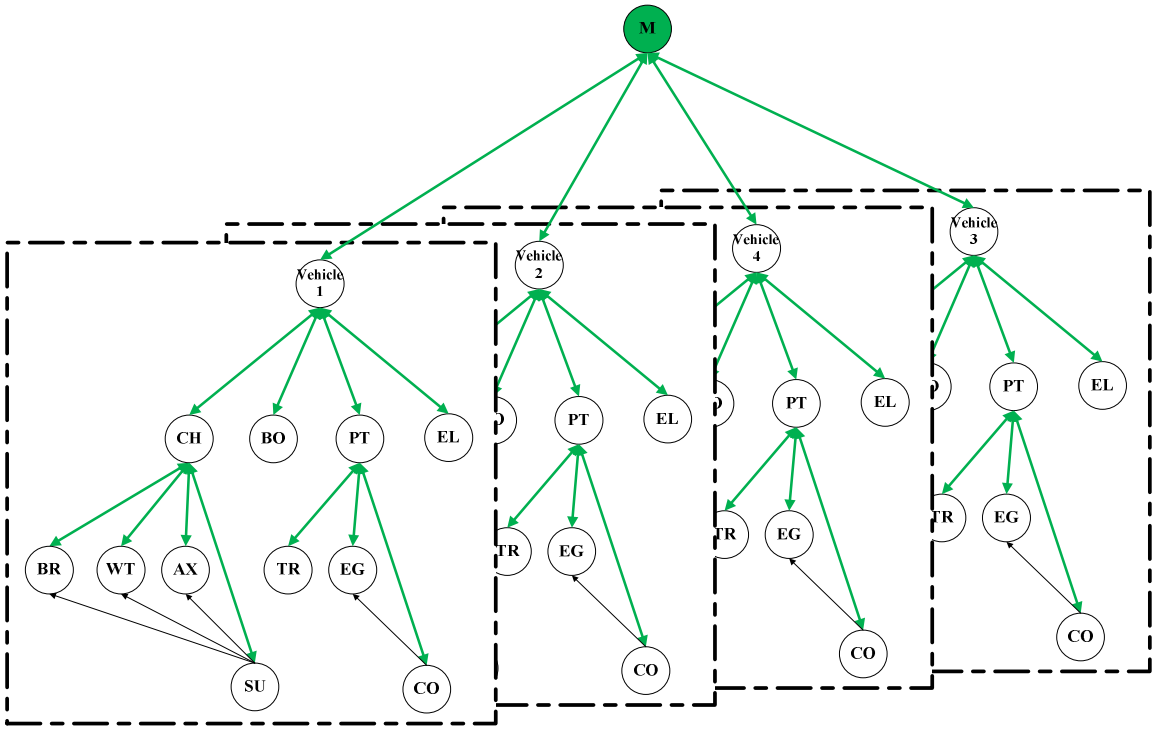


Figure 3.14: The CTBN representation for the ground combat team

The CTBN modeling of this team is shown in Figure 3.14. From the figure, we can see that, for each vehicle, its structure is identical with that of Figure 3.12 in the Ground Vehicle Example. We cannot apply the CR repair policy in this example. When a component fails, it needs to check whether there are repair crews available or not. The CR-limit policy is employed here. There are only two repair crews in this team. If there is not a repair crew available, the failed component has to wait; if there are repair crews available, the failed component can be repaired immediately. The green M node is the maintenance node, which assigns and reclaims repair crews to and from vehicles. Besides the functionalities in ground vehicle example, the double arrow dash green links here also do the following works: when a component fails, a repair crew would be assigned from M node to vehicle node if there is one available; and the vehicle node will assign the repair crew to subsystem and the subsystem will pass on it to the component. After the component is repaired and back to working state, it returns the repair crew back to the subsystem; and then the subsystem passes on it to the vehicle node and finally to the M node. In this study, we simply put the subsystems of the four vehicle having the same failure rates and repair rates as those in the ground vehicle example.

By using CTBNs, the mean availability per vehicle for the mission time of 400 hours is .9397. It is less than the result of .9986 from ground vehicle example (Section 3.5.2) because there are only 2 repair crews available in this example; while there is not limitation for repair crews in ground vehicle example. The mean availability per vehicle changes as the number of repair crew changes. When the number of repair crews decreases to 1, the mean availability decreases to .9016; when the number of repair crews increase to 3, the mean availability is .9713; when the number of repair crews is 4 or

more, the mean availability is .9983, which is almost identical to the result from ground vehicle system example. This can be explained by the fact that, when there are 4, or more than 4 repair crews available, it is identical to that case with unlimited repair crews because there are only 4 vehicles in this fleet.

Table 3.35: CIM for node SU in the fleet example

CH(good)	State	SU(good)	SU(moderate)	SU(failed)	SU(repair)
	SU(good)	$-\lambda_{su,g}$	$\lambda_{su,g}$	0	0
	SU(moderate)	0	$-\lambda_{su,m}$	$\lambda_{su,m}$	0
	SU(failed)	0	0	0	0
	SU(repair)	0	0	∞	$-\infty$
CH(failed) or CH(standby)	State	SU(good)	SU(moderate)	SU(failed)	SU(repair)
	SU(good)	0	0	0	0
	SU(moderate)	0	0	0	0
	SU(failed)	0	0	0	0
	SU(repair)	0	0	0	0
CH(repair)	State	SU(good)	SU(moderate)	SU(failed)	SU(repair)
	SU(good)	0	0	0	0
	SU(moderate)	0	0	0	0
	SU(failed)	0	0	$-\infty$	$-\infty$
	SU(repair)	μ_{su}	0	0	$-\mu_{su}$

Table 3.36: CIM for node BR in the fleet example

CH(good)	SU(good)	State	BR(good)	BR(failed)	BR(repair)
		BR(good)	$-\lambda_{br,g}$	$\lambda_{br,g}$	0
		BR(failed)	0	0	0
		BR(repair)	0	∞	$-\infty$
	SU(moderate)	State	B(good)	B(failed)	BR(repair)
		BR(good)	$-\lambda_{br,m}$	$\lambda_{br,m}$	0
		BR(failed)	0	0	0
		BR(repair)	0	∞	$-\infty$
	SU(failed)	State	B(good)	B(failed)	BR(repair)
		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	0	∞	$-\infty$
	SU(repair)	State	B(good)	B(failed)	BR(repair)

		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	∞	0	$-\infty$
CH(failed)	SU(.)	State	BR(good)	BR(failed)	BR(repair)
		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	∞	0	$-\infty$
CH(standby)	SU(.)	State	BR(good)	BR(failed)	BR(repair)
		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	μ_{br}	0	$-\mu_{br}$
CH(repair)	SU(good)	State	BR(good)	BR(failed)	BR(repair)
		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	μ_{br}	0	$-\mu_{br}$
	SU(moderate)	State	BR(good)	BR(failed)	BR(repair)
		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	μ_{br}	0	$-\mu_{br}$
	SU(failed)	State	BR(good)	BR(failed)	BR(repair)
		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	μ_{br}	0	$-\mu_{br}$
	SU(repair)	State	BR(good)	BR(failed)	BR(repair)
		BR(good)	0	0	0
		BR(failed)	0	0	0
		BR(repair)	μ_{br}	0	$-\mu_{br}$

Table 3.36 Continues

Table 3.37: CIM for node CH in the fleet example

Vehicle(good)	BR, WT, AX and SU are all in good state	State	CH(good)	CH(failed)	CH(standby)	CH(repair)
		CH(good)	0	0	0	0
		CH(failed)	∞	$-\infty$	0	0
		CH(standby)	∞	0	$-\infty$	0
		CH(repair)	∞	0	0	$-\infty$
	Otherwise	State	CH(good)	CH(failed)	CH(standby)	CH(repair)
		CH(good)	$-\infty$	∞	0	0
		CH(failed)	0	0	0	0
		CH(standby)	0	∞	$-\infty$	0
		CH(repair)	0	0	0	0
Vehicle(failed)	BR, WT, AX and SU are all in good state	State	CH(good)	CH(failed)	CH(standby)	CH(repair)
		CH(good)	0	0	0	0
		CH(failed)	∞	$-\infty$	0	0
		CH(standby)	∞	0	$-\infty$	0
		CH(repair)	∞	0	0	$-\infty$
	Otherwise	State	CH(good)	CH(failed)	CH(standby)	CH(repair)
		CH(good)	$-\infty$	∞	0	0
		CH(failed)	0	0	0	0
		CH(standby)	0	∞	$-\infty$	0
		CH(repair)	0	0	0	0
Vehicle(repair)	BR, WT, AX and SU are all in Good state	State	CH(good)	CH(failed)	CH(standby)	CH(repair)
		CH(good)	0	0	0	0
		CH(failed)	∞	$-\infty$	0	0
		CH(standby)	∞	0	$-\infty$	0
		CH(repair)	∞	0	0	$-\infty$
	Otherwise	State	CH(good)	CH(failed)	CH(standby)	CH(repair)
		CH(good)	$-\infty$	∞	0	0
		CH(failed)	0	$-\infty$	0	∞
		CH(standby)	0	∞	$-\infty$	0
		CH(repair)	0	0	0	0

Table 3.38: CIM for node Vehicle in the fleet example

M(2), M(1)	CH, BO, PT and EL are all in good state	State	Vehicle (good)	Vehicle (failed)	Vehicle (repair)
		Vehicle (good)	0	0	0
		Vehicle (failed)	∞	$-\infty$	0
		Vehicle (repair)	∞	0	$-\infty$
	Otherwise	State	Vehicle (good)	Vehicle (failed)	Vehicle (repair)
		Vehicle (good)	$-\infty$	∞	0
		Vehicle (failed)	0	$-\infty$	∞
		Vehicle (repair)	0	0	0
M(0)	CH, BO, PT and EL are all in Good state	State	Vehicle (good)	Vehicle (failed)	Vehicle (repair)
		Vehicle (good)	0	0	0
		Vehicle (failed)	∞	$-\infty$	0
		Vehicle (repair)	∞	0	$-\infty$
	Otherwise	State	Vehicle (good)	Vehicle (failed)	Vehicle (repair)
		Vehicle (good)	$-\infty$	∞	0
		Vehicle (failed)	0	0	0
		Vehicle (repair)	0	0	0

Table 3.39: CIM for node M in the fleet example

# of vehicle in state Repair =0	State	M(2)	M(1)	M(0)
	M(2)	0	0	0
	M(1)	∞	$-\infty$	0
	M(0)	∞	0	$-\infty$
# of vehicle in state Repair =1	State	M(2)	M(1)	M(0)
	M(2)	$-\infty$	∞	0
	M(1)	0	0	0
	M(0)	0	∞	$-\infty$
# of vehicle in state Repair >=2	State	M(2)	M(1)	M(0)
	M(2)	$-\infty$	0	∞
	M(1)	0	$-\infty$	∞
	M(0)	0	0	0

3.6 Conclusion

In this chapter we propose CTBN formalism for RAM modeling of dynamic repairable systems. We also show how to translate special purpose FT gates, called dynamic gates, into the framework of CTBNs. We applied our proposed method to three

case examples derived from practical application. We evaluated the performance of our method and compared to other methods in the case examples. The numerical results shows that the proposed method is as accurate as the traditional methods, which indicates that it is a good alternative for existing RAM modeling methods.

In the next chapter, in the system design optimization framework, we will employ CTBNs as a reliability (availability) estimation method to calculate the system reliability (availability) and do the system design optimization analysis.

Chapter 4 System Design Optimization Using NSGA-II and CTBNs

In this chapter, we propose a system design framework for systems with dynamic failure behaviors or various repair policies. The CTBNs are used to estimate the reliability (availability) and the multi-objective GA algorithm NSGA-II is implemented to solve for the Pareto solutions. We first provide the background about multi-objective optimization and a brief introduction of NSGA-II, and then present the NSGA-II+CTBNs system design optimization framework. Finally, to show how this framework works, a case example is demonstrated.

4.1 Multi-objective Optimization Problem

In general, for a problem with n objective functions, the multi-objective formulation can be formulated as follows:

$$\min f_i(x) \ \ for\ i = 1,2,\dots,n$$

Subject to

$$g_j(x) \le 0, \qquad j = 1,2,\dots,J,$$

$$h_k(x) = 0, \qquad k = 1,2,\dots,K.$$

There are n objective functions and p variables so $f(x)$ is an n dimensional vector, and x is a p dimensional vector corresponding to p decision variables. Solutions to a multi-objective optimization problem are often mathematically expressed in terms of non-dominated or superior points. Non-dominance can be defined as: in a minimization problem, a solution x_1 dominated a solution x_2 (a), if and only if x_1 is no worse than x_2

in all objectives, i.e.$f_i(x_1) \leq f_i(x_2)\ \forall\ i, j \in \{1,2,\ldots,n\}$; and (b), $x_1$is strictly better than x_2 in at least one objective, i.e. $f_i(x_1) < f_i(x_2)$ for at least one i. Thus, instead of a unique solution to the problem, the solution to a multi-objective optimization problem is a set of Pareto-optimal points (Zeleny 1982).

Generally there are two common ways to solve multi-objective problems: 1) combine them into a single objective function and obtain a single solution such as in the cases of weighted sum method or utility function, or 2) obtain a set of non-dominated Pareto-optimal solutions.

For multi-objective problems, it can be problematic to combine the objectives into a single objective (e.g. weighted sum method, utility functions) to obtain a single solution. A slight perturbation in the parameters used to combine the objectives could result in very different optimal solutions. This can be a problem because the exact objective function weights or utility functions are often not that clear. The Pareto set includes all rational choices, among which we have to select the final solution by trading the objectives again each other. The search is then not for one optimal solution but for a set of solutions that are optimal in a broader sense, i.e. they are Pareto-optimal.

4.1.1 Multi-objective GAs

In this study, we use CTBNs to estimate the system reliability (availability). Thus at least one objective function is non-close-form; it is black-box type. For black-box type optimization problem, meta-heuristic based methods like GAs are the most efficiency approaches to solve them. Genetic Algorithm (GA) was proposed by Holland (Holland 1975). GAs are a particular class of evolutionary algorithms that uses techniques inspired

by some mechanisms of natural selection. They are essentially search techniques used to find approximate solutions to difficult combinatorial optimization problems.

The GA starts with a population of random individuals (chromosomes) that are revised over successive generations. The crossover and mutation operators are used to introduce new prospective design solutions each generation. During each successive generation, each individual is evaluated and a value of fitness is returned by a fitness function. Individuals with high-fitness values rank at the top while individuals with low-fitness values are likely to be abandoned from the population. The algorithm continues for a pre-determined maximum number of generations or until no additional improvement is observed.

Several versions of multi-objective GAs, most often referred as multi-objective evolutionary algorithms (MOEAs), have been developed, such as:

- Vector evaluated genetic algorithm (VEGA) (Schaffer 1985);
- Multi-objective genetic algorithm (MOGA) (Fonseca and Fleming 1993);
- Niched-Pareto genetic algorithm (NPGA) (Horn, Nafpliotis et al. 1994);
- Non-dominated sorting genetic algorithm (NSGA) (Srinivas and Deb 1994);
- Strength Pareto evolutionary algorithm (SPEA) (Zitzler and Thiele 1999);
- Non-dominated sorting genetic algorithm-II (NSGA-II) (Deb, Pratap et al. 2002).

As a well-know MOEA, the NSGA-II is the most widely used and has been proven to perform well on various real-world application problems (Coello Coello 2006). The pseudo-code of NSGA-II is presented in Algorithm 4.1.

We will employ NSGA-II in our system design optimization framework, since there have been many studies ensuring that NSGA-II can often converge to Pareto set and the obtained solution can often spread well over the Pareto set. NSGA-II takes the fast-non-dominated-sort mechanism to ensure the well convergence which is shown in Algorithm 4.2. Moreover, it adopts the Density Estimation and Crowding Comparison Operator (Deb, Pratap et al. 2002) to cut the solutions which have bad distributions so as to obtain a good spread of solutions. The above merits of NSGA-II make it a promising choice of solving the black-box type problem in our study. For more details of NSGA-II, one can refer to the paper (Deb, Pratap et al. 2002).

Algorithm 4.1: The Pseudo-Code of NSGA-II

1: Set the parent vector $P = \emptyset$, the offspring vector $Q = \emptyset$, the collect vector $R = \emptyset$ and the generation number $t = 0$.
2: Initialize the parent vectorP_0.
3: **While** $t <$ the terminate generation number **do**
4: (1) Combine the parent and offspring population via $R_t = P_t \cup Q_t$
5: (2) Sort all solutions of R_t to get all non-dominated fronts F =fast-non-dominated-sort(R_t) where $F = (F_1, F_2, \dots)$.
6: (3) Set $P_{t+1} = \emptyset$ and $i = 1$
7: (4)
8: **While** the parent population size $|P_{t+1}| + |F_i| < N$ **do**
9: (a) Calculate crowding-distance of F_i.
10: (b) Add the ith non-dominated front $F_i$to the parent pop P_{t+1}.
11: (c) $i = i + 1$.
12: **End while**
13: (5) Sort the F_i according to the crowding distance.
14: (6) Fill the parent pop P_{t+1} with the first $N - |P_{t+1}|$ elements of F_i.
15: (7) Generate the offspring population to Q_{t+1}.
16: (8) Set $t = t + 1$
17: **End while**
18: the population in vector P are the non-dominated solutions.

Algorithm 4.2: The Pseudo-Code for the function: fast-non-dominated-sort(P)

1: For each population p in the P, we get the solutions which p dominates and save these solutions into S_p. We also need to calculate the n_p which is the number of solutions which dominates p.
2: Find the solutions whose $n_p = 0$ and add them to the first front F_1.
3: Initialize the front counter $i = 1$.
4: **While** F_i is not empty **do**
5: Set the temp vector $Q = \emptyset$
6: **For** each $p \in F_i$ **do**
7: **For** each $q \in S_p$ **do**
8: $n_q = n_q - 1$.
9: if $n_q = 0$ then add q to the Q.
10: **End for**
11: **End for**
12: i= $i + 1$ and the solutions in Q compose the F_i
13: **End while**

4.2 CTBNs and NSGA-II System Design Optimization Framework

In this study, for system design optimization problem, we only consider three system metrics: the system reliability (availability), cost and weight. However, in practical application, it is not limited to these three metrics. Consider a system with N components and for component i, it has M_i option for chose, where for option j of component i, the failure rate is λ_{ij}, the repair rate is μ_{ij} , the cost is c_{ij} and the weight is w_{ij}. If j option is chose for component i, then $x_{ij} = 1$, otherwise, $x_{ij} = 0$.

The system performance metrics are shown as follows:

The reliability R (availability A) of the system can be estimated by CTBNs. For a specified system, a CTBN is constructed.

$$R/A = CTBNs\left(\left\{\sum_{j=1}^{M_1} x_{1j}\lambda_{1j}, \sum_{j=1}^{M_1} x_{1j}\mu_{1j}\right\}, \dots, \left\{\sum_{j=1}^{M_N} x_{Nj}\lambda_{Nj}, \sum_{j=1}^{M_N} x_{Nj}\mu_{Nj}\right\}\right)$$

where j is the selected option for components.

We assume that the system cost and weight are simply the summation of each component:

$$C = \sum_{i=1}^{N}\sum_{j=1}^{M_i} x_{ij}c_{ij}$$

$$W = \sum_{i=1}^{N}\sum_{j=1}^{M_i} x_{ij}w_{ij}$$

And the system design optimization problem can be formula as:

$$\left\{\begin{array}{c} \max CTBNs\left(\left\{\sum_{j=1}^{M_1} x_{1j}\lambda_{1j}, \sum_{j=1}^{M_1} x_{1j}\mu_{1j}\right\}, \dots, \left\{\sum_{j=1}^{M_N} x_{Nj}\lambda_{Nj}, \sum_{j=1}^{M_N} x_{Nj}\mu_{Nj}\right\}\right), \\ \min \sum_{i=1}^{N}\sum_{j=1}^{M_i} x_{ij}c_{ij}, \min \sum_{i=1}^{N}\sum_{j=1}^{M_i} x_{ij}w_{ij} \end{array}\right\}$$

Subject to

$$\sum_{j=1}^{M_i} x_{ij} = 1 \text{ for } i = 1,2, \dots, N$$

4.2.1 Chromosomal Representation

In this study, binary coding scheme is employed. The length of chromosome is $\sum_{i=1}^{N} M_i$.

The first M_1bits are for the M_1 options of component 1. Each bit is corresponding to one option, if the option is chose, it is 1 otherwise it is 0. From $M_1 + 1$ bit to $M_1 + M_2$ bits are for options of component 2. The arrangements of remaining components are similar to component 2. For example, consider Figure 4.1, the chromosome contains twelve bits for a configuration which consists of three components, with options of four for each component. For component1, the 1st option is chose, for component 2, the 2nd option is chose and the last option is chose for component 3.

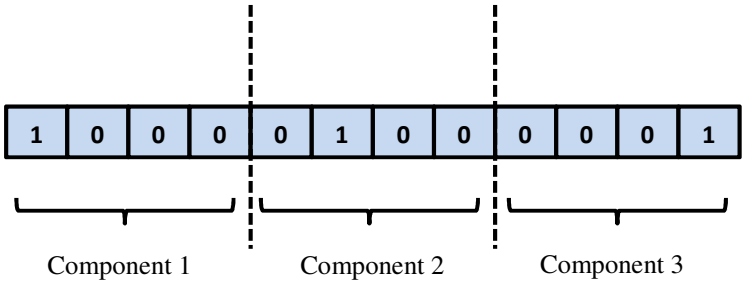


Figure 4.1: Encoding of the solutions

4.2.2 Genetic Operator

On the basis of coding scheme, we adopt the single-point crossover and bitwise mutation for NSGA-II. The detailed implementations of them are presented in Figure 4.2.

In order to guide the search within the feasible region, we utilize the constraint handling approach based on the concept of constrained-dominate proposed in the paper (Deb, Pratap et al. 2002). Concretely, a solution i constrained-dominates j must satisfy one of the following three conditions: 1) Solution i is feasible but solution j is not; 2) Solution i and j are both feasible, and i dominates j; 3) Solution i and j are both infeasible, but i violates less constraints than j.

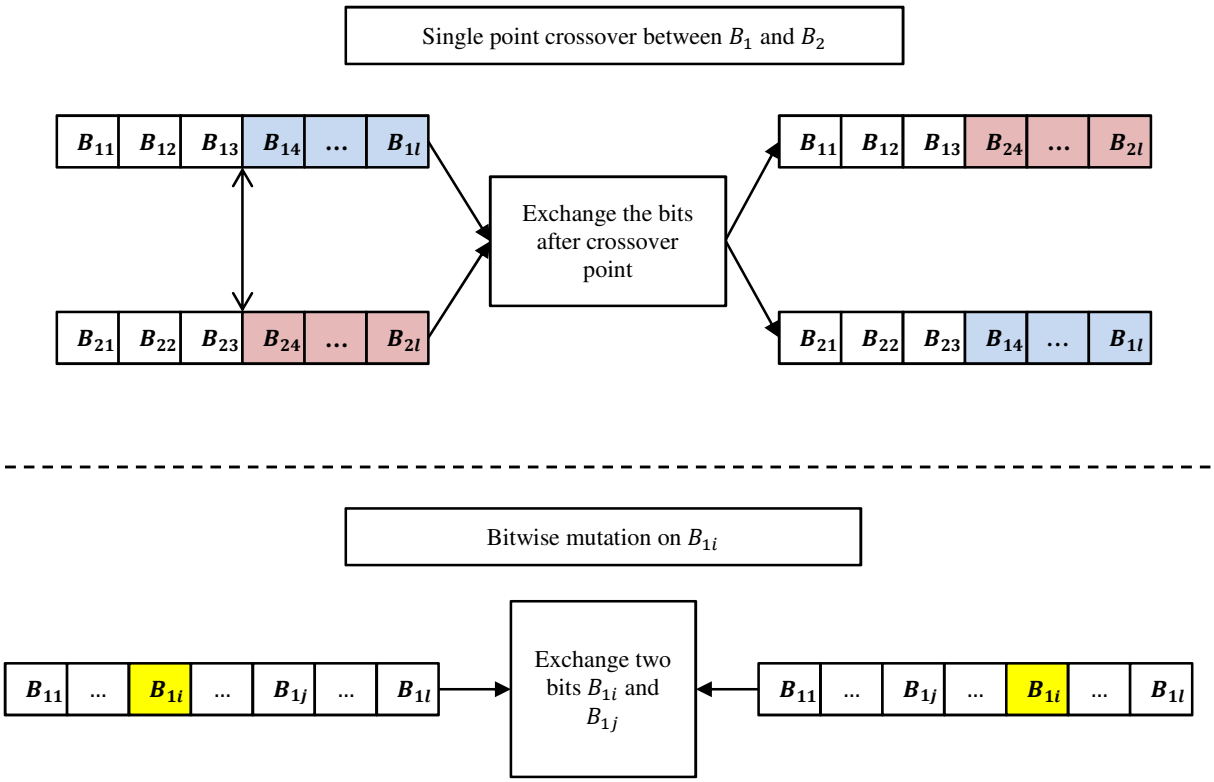


Figure 4.2: Crossover and Mutation

4.3 Case Example: Ground Vehicle System Design

To demonstrate the proposed system design optimization framework, the ground vehicle system example (Section 3.5.2) in the previous chapter is considered. In this example, except body subsystem and electrical subsystem, other subsystems all have four levels of options for chose. Each level has different failure rates, cost and weight as shown in Table 4.1.

The system performance metrics are system reliability, cost and weight. The parameter settings of NSGA-II are as follows: the crossover probability is set to be .9 and the mutation rate is .001. The terminate generation is set to 50 and the population size is 500.

Figure 4.3 shows the 104 solutions found in the Pareto-front. To better visualize the solutions obtained, figure 4.4 and figure 4.5 show the two dimensional representation of the same solutions.

Table 4.1: Choices for each subsystem of Series-Parallel system

Subsystems	Failure Measure				Cost				Weight			
	1	2	3	4	1	2	3	4	1	2	3	4
BR	.5e-3	.3e-3	.62e-3	.43e-3	9	12	9	8	21	26	19	34
WT	.16e-3	.21e-3	.23e-3	.28e-3	5	4	3	6	35	45	43	45
AX	.5e-4	.3e-4	.7e-4	.6e-3	4	9	7	6	65	47	38	42
SU	.5e-4	.7e-4	.6e-4	.9e-4	7	7	9	6	43	17	34	26
TR	.83e-4	.67e-4	.91e-4	.68e-4	12	14	13	17	23	32	42	45
EG	.23e-4	.33e-4	.27e-4	.35e-4	6	6	5	8	98	79	86	89
CO	.33e-4	.28e-4	.25e-4	.36e-4	8	9	8	4	12	19	27	34

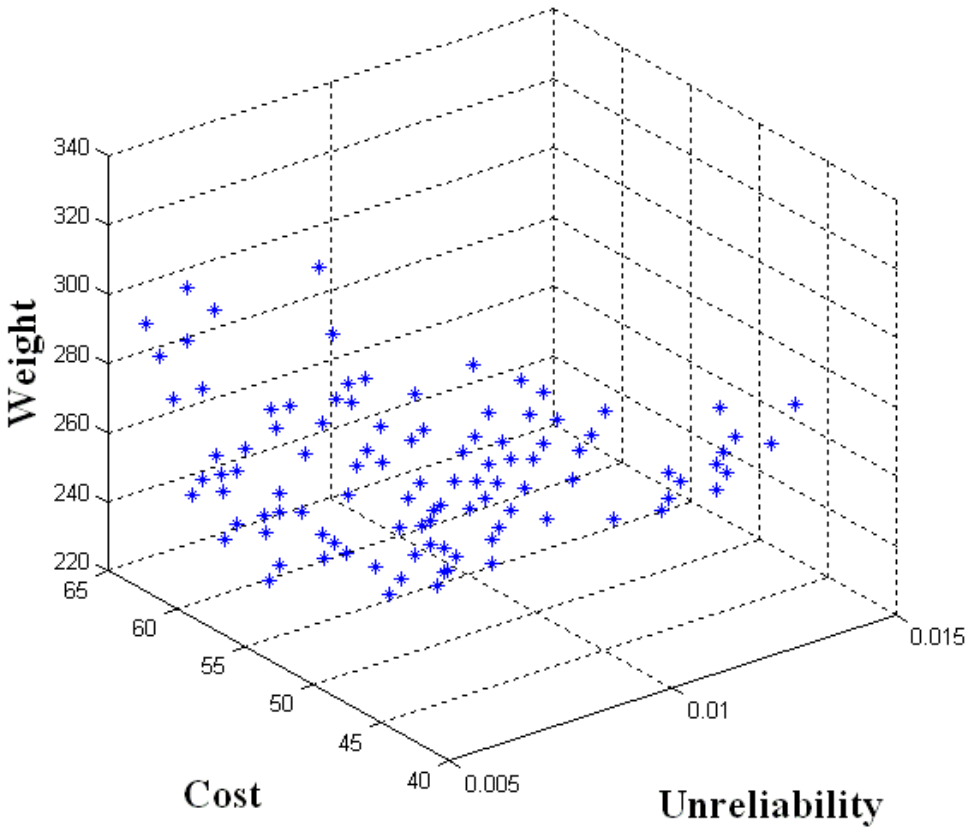


Figure 4.3: Unreliability vs Cost vs Weight

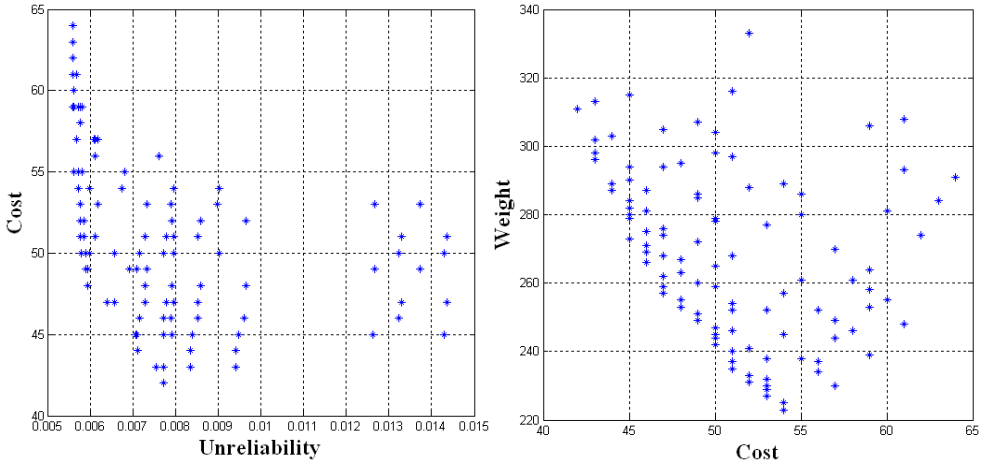


Figure 4.4: Unreliability vs Cost (left); Unreliability vs Weight (right)

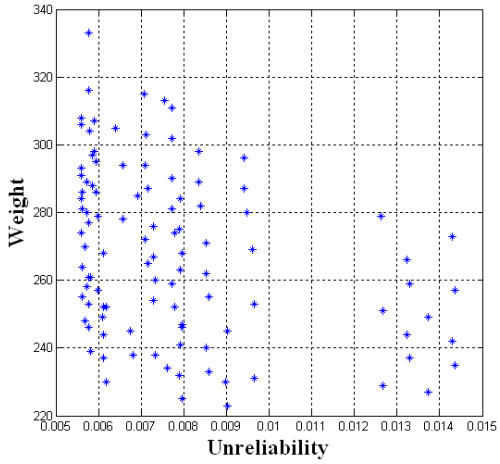


Figure 4.5: Cost vs Weight

As we discussed before, there are limitations in meta-heuristic based-methods like NSGA-II. There is no guarantee that the solutions are globally optimal. However, for black-box type optimization problems, NSGA-II is the most efficient way to solve it currently. In the next chapter, for systems with simple structures (series/parallel) and close-form objective functions, we propose a modified adaptive ϵ-constraint method to identify all Pareto-optimal solutions.

4.4 Conclusion

We have presented a system design optimization framework for systems with dynamic behaviors or various repair policies. We employ the CTBNs to estimate the system reliability (availability) and put it as a multi-objective optimization problem and then use NSGA-II to solve it. Finally, the proposed framework is applied to an example of a ground vehicle system to illustrate its performance.

Chapter 5 System Design Optimization using Modified Adaptive ϵ-Constraint Method

In the previous section, for systems with dynamic behavior or various repair policy, we present a CTBNs and NSGA-II based system design optimization framework. However, as we mentioned before, NSGA-II cannot guarantee globally optimal. In this chapter, for systems with simple structures (series/parallel) and close-form objective functions, we proposed a modified adaptive ϵ-constraint method to identify all the Pareto-optimal solutions for the system design optimization analysis. A brief introduction of traditional ϵ-constraint method and adaptive ϵ-constraint method are given. Then we present the modified adaptive ϵ-constraint method in detail. Finally the proposed method is implemented in two case examples to evaluate its performance with NSGA-II and other ϵ-constraint methods. The first case example is the well-known Redundancy Allocation Problem (RAP) in Series-Parallel systems. The other case example is a practical configuration selection problem which is taken from our industrial partner.

5.1 The Traditional ϵ-constraint Methods

The traditional ϵ-constraint method is a multi-objective optimization technique proposed by Chankong and Haimes (Chankong and Haimes 1983) for generating Pareto-optimal solutions. It transforms the multi-objective problem into a series of several single-objective problems with updated constraints, using the following procedure:

$$\begin{gathered} min\ f_i(x) \\ \text{s.t.} \\ f_j(x) \le \varepsilon_j \quad for\ all\ j = 1,2,\dots,m,\ \ j \ne i \end{gathered}$$

where

$i \in \{1,2,3,\dots,m\}$
$\epsilon = (\epsilon_1, \epsilon_2, \dots, \epsilon_m)$ are the upper bounds of each objective function

In order to identify all non-dominated solutions, the vector of upper bounds must be varied (iteratively increase or decrease by a pre-defined constant Δ) along the Pareto front for each objective and perform a new optimization process for each new upper bound vector. The generation of different non-dominated points using different upper bound values is illustrated in bi-objective case in Figure 5.1.

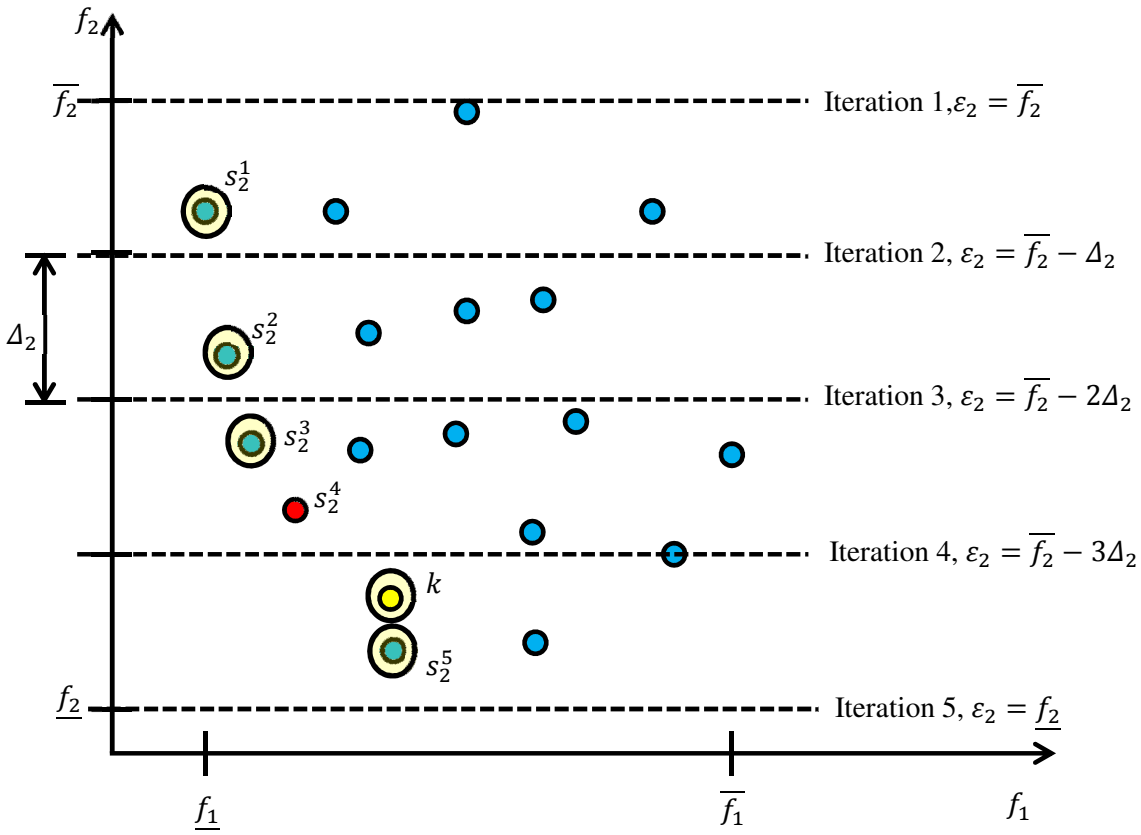


Figure 5.1: An illustrative example of Generating different solutions with the traditional ϵ-constraint method generating different solutions sequentially under two objective functions that need to be minimized

There are two limitations to the traditional ϵ-constraint method. Firstly, it is the necessity to choose a pre-defined constant Δ. Since only one solution can be found in each interval, the discretization has to be fine enough not to "miss" any Pareto-optimal

solution. As shown in Figure 5.1, there are two Pareto-optimal points in iteration 3. However, solution s_2^4 (highlighted in red) is missed due to the larger Δ. Second, it will identify non-Pareto-optimal solution. The main reason is that it is just takes one objective function into consideration. Let's look back to Figure 5.1 again. In iteration 4, the two solutions have the same fitness value of f_1, so both of them are identified as solution. However, solution k is not a Pareto-optimal solution and is dominated by solution s_2^5. The traditional ϵ-constraint method can be summarized as follows.

Algorithm 5.1: The traditional ϵ-constraint method

Input

Objective bounds $\underline{f_i}$, $\overline{f_i}$ for each $i \in \{2, \dots, m\}$

Increments Δ_i for each $i \in \{2, \dots, m\}$

Output

Set of solution contain Pareto-optimal solution set S

$S = \emptyset$
For $\varepsilon_2 := \underline{f_2}$ to $\overline{f_2}$ step Δ_2 do
    For $\varepsilon_3 := \underline{f_3}$ to $\overline{f_3}$ step Δ_3 do
    ⋮
        For $\varepsilon_m := \underline{f_m}$ to $\overline{f_m}$ step Δ_m do
            Solve (1) for x, $S := S \cup x$
        End for
    ⋮
    End for
End for
Return S

To cope with the drawback of the traditional ϵ-constraint method, Ozlen (Ozlen and Azizoglu 2009) presented an adaptive ϵ-constraint method for the multi-objective integer programming (MOIP) problem. Unlike the traditional ϵ-constraint method which determines ϵ by decreasing a fixed Δ in each iteration, the adaptive ϵ-constraint method uses an adaptive ϵ value. It determines the ϵ based on the solutions of previous iteration.

This increases the efficiency of the algorithm dramatically and will not miss a single Pareto-optimal solution. In order to avoid identifying non-Pareto-optimal solutions, instead of using a single objective function by implementing a proper weight for each objective function, the adaptive ϵ-constraint method constructs a new single objective function (which is a weighted sum of all the original objective functions) and solves this new objective function. This way, it takes all objective functions into consideration during the search.

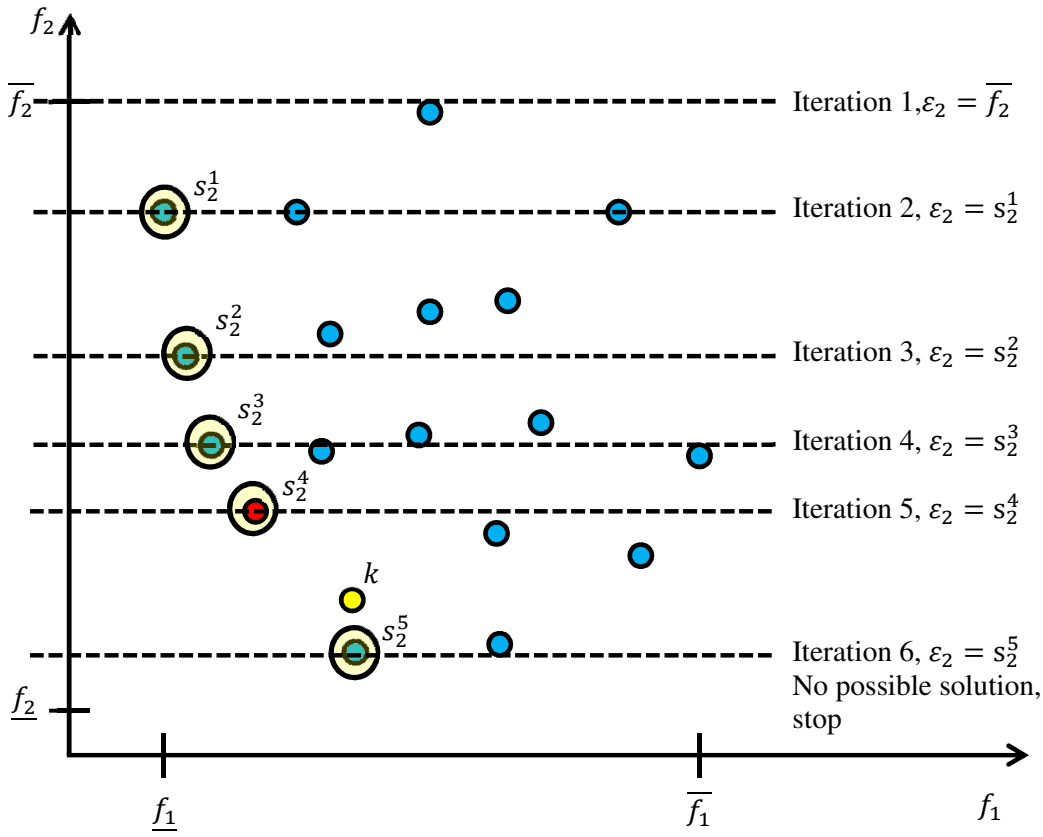


Figure 5.2: An illustrative example of the adaptive ϵ-constraint method generating different solutions sequentially under two objective functions that need to be minimized

Let's reconsider a tri-objective problem:

$$\mathbf{P1}: min\{ f_1(x), f_2(x), f_3(x)\}$$

The specific weighted sum single objective function problem solved by the ϵ-adaptive constraint method is:

$$\mathbf{P2}: min\ f_1(x) + w_2 f_2(x) + w_3 f_3(x)\}$$

$$w_2 = \frac{1}{f_2^{GUB} - f_2^{GLB} + 1},\ w_3 = \frac{W_2}{f_3^{GUB} - f_3^{GLB} + 1}$$

$$s.t.$$

$$f_2 < \epsilon_2$$

$$f_3 < \epsilon_3$$

Here $f_2^{GUB}, f_2^{GLB}, f_3^{GUB}$ and f_3^{GLB} are the upper and lower bounds on $f_2(x)$ and $f_3(x)$ values of any feasible solution respectively.

The weight for f_1 is 1, and since all objective functions are linear integer functions, the minimal increment of $f_1$is 1, which is always greater than the maximal increment of f_2. By the same logic, the maximal increment of f_3 is always less than the minimal increment of f_2. These weights make sure that f_1 has the high priority, then f_2 and f_3. Let's look back to the bi-objective problem in Figure 5.2 again. In iteration 4, under the constraint determined by solution in iteration 3, we cannot miss solution s_2^4 (red point). While in iteration 5, the yellow point k and s_2^5 have the same f_1 value. However, since we use the weight sum objective function, we also take f_2 into consideration. With f_2 included, obviously yellow point k is a dominated point.

The proposed approach of assigning proper weights to the objectives thus allows one to solve the weighted sum objective function and still maintain the hierarchy of the multiple objectives. The adaptive ϵ-constraint method to solve P1 can be summarized as follows.

Algorithm 5.2: The adaptive ϵ-constraint method

Input

Objective bounds f_i^{GLB}, f_i^{GUB} for each $i \in \{2, \dots, m\}$,

$$w_i = \frac{w_{i-1}}{f_i^{GUB}, -f_i^{GLB}, +1}, \mathrm{w}_1 = 1$$

$$f = f_1 + w_2 f_2 + \cdots + w_m f_m$$

$$\epsilon_i = f_i^{GUB}, i \in \{2, \dots, m\}, flag = 1$$

Output

Set of Pareto-optimal solution set S

1: $S = \emptyset$
2: **While** $flag = 1$ **do**
3: $\epsilon_{m-1} \coloneqq f_{m-1}^{GUP}$
4: **While** $flag = 1$ **do**
5: $\epsilon_{m-2} \coloneqq f_{m-2}^{GUP}$
6: ⋮
7: $\epsilon_2 \coloneqq f_2^{GUP}$
8: **While** $flag = 1$ **do**
9: Solve (1) for x,
10: **If** the solution is impossible
11: $flag = 0$
12: **Else**
13: $S_2 \coloneqq S_2 \cup x$, $\varepsilon_2 \coloneqq f_2(x) - 1$
14: **End if**
15: **End while**
16: $\{\epsilon_3 \coloneqq \max(f_3(x)) - 1 : x \in S_2\}$, $S_3 \coloneqq S_3 \cup S_2$
17: ⋮
18: **End while**
19: $\{\epsilon_m \coloneqq \max(f_m(x)) - 1 : x \in S_{m-1},\}$ $S_m \coloneqq S_m \cup S_{m-1}$
20: **End while**
21: $S \coloneqq S_m$
22: Return S

The adaptive ϵ-constraint method improves the efficiency over the traditional method significantly. However, there are still drawbacks in this method. It has the potential to identify a lot of identical solutions. In order to illustrate this inefficiency and introduce our method, let us consider a numerical example taken from Ozlen's paper (Ozlen and Azizoglu 2009).

Numerical Example

This is a 5×5 Tri-objective Assignment Problem (TAP). Table 1 has the three objective coefficients for assigning each row to each column. Each solution is represented by a sequence of column index values assigned to row 1 through 5. Accordingly, in sequence 5-4-3-2-1, row 1 is assigned to column 5 and row 2 is assigned to column 5.

Using the single-objective assignment solutions one can identify general upper and lower bounds on the individual objectives as

$$\underline{f_1} = 86, 2-1-4-3-5$$

$$\underline{f_2} = 128, 1-5-4-3-2$$

$$\underline{f_3} = 129, 3-2-1-5-4$$

$$\overline{f_1} = 358, 4-2-3-5-1$$

$$\overline{f_2} = 411, 4-2-1-5-3$$

$$\overline{f_3} = 451, 4-5-3-1-2$$

Table 5.1: Three objective coefficients for the numerical example problem

c^1	1	2	3	4	5		c^2	1	2	3	4	5		c^2	1	2	3	4	5
1	99	19	74	55	41		1	28	39	19	42	7		1	29	67	2	90	7
2	23	81	93	39	49		2	66	98	49	83	42		2	84	37	64	64	87
3	66	21	63	24	66		3	73	26	42	13	54		3	54	11	100	83	61
4	65	41	7	39	66		4	46	42	28	27	99		4	75	63	69	96	3
5	93	30	5	4	13		5	80	17	99	59	68		5	66	99	34	33	21

The iteration details of algorithm 5.2 are presented in Table 5.2. We report the number of IPs solved and the ϵ_2 and ϵ_3 bound values. The objective function values of

the bi-objective solutions are stated in group each representing a single execution the "while loop" from step 8 to step 15 in algorithm 5.2.

Table 5.2: The iteration details

G1	$\epsilon_3 \leq 451$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
1	86	214	324	411
2	96	186	204	213
3	125	131	342	185
4	209	128	367	130
5	Infeasible			127
	Max(f_3)		367	

G2	$\epsilon_3 \leq 366$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
6	86	214	324	411
7	96	186	204	213
8	125	131	342	185
9	Infeasible			130
	Max(f_3)		342	

G3	$\epsilon_3 \leq 341$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
10	86	214	324	411
11	96	186	204	213
12	180	183	229	185
13	253	132	328	182
14	Infeasible			
	Max(f_3)		328	

G4	$\epsilon_3 \leq 327$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
15	86	214	324	411
16	96	186	204	213
17	180	183	229	185
18	269	173	320	182
19	Infeasible			
	Max(f_3)		324	

G5	$\epsilon_3 \leq 323$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
20	91	246	314	411
21	96	186	204	245
22	180	183	229	185
23	269	173	320	182
24	Infeasible			172
	Max(f_3)		320	

G6	$\epsilon_3 \leq 319$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
25	91	246	314	411
26	96	186	204	245
27	180	183	229	185
28	Infeasible			182
	Max(f_3)		314	

G7	$\epsilon_3 \leq 313$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
29	96	186	204	411
30	180	183	229	185
31	Infeasible			182
	Max(f_3)		229	

G8	$\epsilon_3 \leq 228$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
32	96	186	204	411
33	Infeasible			185
	Max(f_3)		204	

G9	$\epsilon_3 \leq 203$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
34	171	261	191	411
35	179	233	194	260
36	224	187	190	232
37				186
	Max(f_3)		194	

G 10	$\epsilon_3 \leq 193$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
38	171	261	191	411
39	212	242	173	260
40	224	187	190	241
41	Infeasible			186
	Max(f_3)		191	

G 11	$\epsilon_3 \leq 190$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
42	188	269	133	411
43	212	242	173	268
44	224	187	190	241
45	Infeasible			186
	Max(f_3)		190	

G 12	$\epsilon_3 \leq 189$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
46	188	269	133	411
47	212	242	173	268
48	Infeasible			241
	Max(f_3)		173	

G 13	$\epsilon_3 \leq 172$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
49	188	269	133	411
50	283	261	140	268
51	Infeasible			260
	Max(f_3)		140	

G 14	$\epsilon_3 \leq 139$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
52	188	269	133	411
53	Infeasible			268
	Max(f_3)		133	

G 15	$\epsilon_3 \leq 132$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
54	291	348	129	411
55	Infeasible			347
	Max(f_3)		129	

G 16	$\epsilon_3 \leq 128$			$\epsilon_2 \leq$
#	f_1	f_2	f_3	
56	Infeasible			411

From Table 5.2, we can see that a total of 56 Integer Programming (IP) are solved to identify 15 unique tri-objective non-dominated solutions. The 15 non-dominated solutions are listed in Table 5.3. 56 IP are solved to get only 15 unique solutions. From Table 5.2, we also can see that there are so many identical solutions between groups. If we can skip solving the duplicated solutions, we can speed up the search and increase the algorithm efficiency significantly. This is the motivation for proposing the modified adaptive ϵ-constraint method.

Table 5.3: The Pareto-optimal solutions for the numerical example problem

Solutions	f_1	f_2	f_3
1	86	214	324
2	91	246	314
3	96	186	204
4	125	131	342
5	171	261	191
6	179	233	194
7	180	183	229
8	188	269	133
9	209	128	367
10	212	242	173
11	224	187	190
12	253	132	328
13	269	173	320
14	283	261	140
15	291	348	129

5.2 The Modified Adaptive ϵ-Constraint Method

In order to introduce the modified adaptive ϵ-constraint method, let us reexamine the results from group 1 (G1, iterations 1-5) and group 2 (G2, iterations 6-9) of Table 5.2. With constraints $\epsilon_2 \leq 411$ and $\epsilon_3 \leq 451$, we get the 1st Pareto-optimal solution to

be $f_1 = 86$, $f_2 = 214$, and $f_3 = 314$. The constraints of the 1st iteration in group 2 (iteration 6) are $\epsilon_2 \leq 411$ and $\epsilon_3 \leq 366$.

The only difference between these two iterations is that the constraint $\epsilon_3 < 451$ is changed to $\epsilon_3 \leq 366$. Notice that, with $\epsilon_3 < 451$, the obtained objective value $f_3 = 314$ also meets the constraint $\epsilon_3 \leq 366$. Thus, with the constraints $\epsilon_2 \leq 411$ and $\epsilon_3 \leq 366$, the solution should be identical to that with constraints $\epsilon_2 \leq 411$ and $\epsilon_3 \leq 451$, which is $f_1 = 86$, $f_2 = 214$, and $f_3 = 314$. Within each group #, the next iteration depends on the previous iteration (ϵ_2 is the previous $f_2 - 1$). For each group, if the 1st iteration has the identical solution as that of the previous group, the solutions of the following iterations in this group will be identical to that of the previous group until it hits the upper bound of constraint ϵ_3. For example, in group 1 and group 2, since the solution of the 1st iterations are the same for these two group, the iteration 7-8 have the same solution as iteration 2-3 in group 1, until it violates the constraint $\epsilon_3 \leq 366$. The solution of iteration 9 cannot be identical to that of iteration 4 because $f_3 = 367$ in iteration 4 violates the constraint $\epsilon_3 \leq 366$ in iteration 9. From the analysis above, we can see that if we skip solving for the same solution, we can reduce the number of Integer Programming (IP) formulations solved so that the whole search procedure can be speeded up. In order to avoiding solving for the repeated solutions, we save the previous group temporarily. .

In particular, we add two refinement strategies to the adaptive ϵ-constraint algorithm. The first strategy involves checking solutions from the previous group on the current constraint set, in the very first iteration, before attempting to solve the new problem. We find the solution s_{1stV} that first violates the current constraint set. If s_{1stV} is

not the 1st solution in the previous group, let us say it is the k^{th} solution in the previous group, then we just copy the first $k-1$ solutions in the previous group as the first $k-1$ solutions in the current group and continuous the outer "while loop" of the Adaptive ϵ-Constraint Algorithm. If the very first solution of the previous group violates the current constraint set, we do not achieve any efficiency; we apply the algorithm as normal. However, in the latter case, a second refinement strategy might come in handy. For example, let's look back to group 1 and group 2 in Table 5.2 again. Before processing iteration 6 (1st iteration of group 2), we save all solutions in the group 1 and check these solutions on the current constraint set ($\epsilon_2 \leq 411$ and $\epsilon_3 \leq 366$). We can see that the $4^{th}(k=4)$ solution violates the constraints. Thus the first 3 solutions from group 1 are copied directly as the first three solutions for group 2 and the outer "while loop" of the algorithm continuous in iteration 9, which does not have a possible solution and group 2 is done.

As discussed above, within each group, current solution exploits solutions from the previous groups. In the second refinement strategy, we check the current solution against the solutions in the previous group. If there is an identical solution $s_{identical}$ in the previous group, then it is possible that the current iteration will have the same next solution as that of $s_{identical}$. What we need to do is check the next solution of $s_{identical}$ against the current constraint set. If it doesn't violate current constraints, then we just copy this one as our next solution and skip the IP solving. Let us reexamine the numerical example from Table 5.2. In group 4 and group 5, the solutions of the first iteration of both groups are not identical. However, when we get the solution of iteration 21, we find that it is identical to the solution of iteration 16. Then we can check the solution of iteration

17 against the current constraint set. It does not violate the constraints in this example, so the iteration 17 and 22 should have the same solution and we can skip the IP solving of iteration 22. Following the same logic, we can avoid solving the IP and get the solution for iteration 23.

Thus, the modified adaptive ϵ-constraint method is obtained by adding the first refinement strategy before step 8 and the second refinement strategy before step 9 of the basic adaptive ϵ-constraint method (Algorithm 5.2). The pseudo code for the two refinement strategies is provided below.

Search Refinement Strategy #1

1: Save solutions of previous group S_{i-1}
2: Check 1st solution in S_{i-1} on current constraint
3: **If** it doesn't violate constraints
4: Copy solution 1 by 1 in S_{i-1} to current group
5: As solution until it violate the current constraints
6: **Else**
7: Process step 8 to 15 in Procedure 1 as normal
8**: End if**

Search Refinement Strategy #2

1: Save solutions of previous group S_{i-1}
2: Check current solution with solution set of previous group
3: **If** there is an identical solution $s_{identical}$ in previous group
4: Check the solution next to $s_{identical}$ again current constraints
5: **If** it doesn't violate current constraints
6: Copy solution as the next solution of current iteration
7: **Else**
8: Process step 8 to 15 in Procedure 1 as normal
9: **End if**
10: **Else**
11: Process step 8 to 15 in Procedure 1 as normal
12: **End if**

By applying the modified adaptive ϵ-constraint method to the numerical example mentioned above (i.e., employ the two search refinement strategies), only 35 IPs are needed to be solved to obtain all the 15 unique Pareto-optimal solutions. Compared with the 56 IPs needed for the original adaptive ϵ-constraint method, there is $(56-35)/56 = 37.5\%$ improvement. To evaluate the proposed method more comprehensively, we randomly generated an additional 1000 TAP problem sets (randomly generate 1000 sets of matrix $0< c^i \leq 100, i = 1,2,3$) and solved them by the two methods. The average improvement observed from the 1000 problems is 39%. However, the savings do vary from problem to problem. This will be further confirmed by implementing the proposed method on the Redundancy Allocation Problem (RAP) for a Series-Parallel system in the later sections.

5.3 Case Example: RAP of Series Parallel System

The redundancy allocation problem for Series-Parallel systems has received much attention in the literature (Oiddir, Rahli et al. 2004; Levitin and Lisninaski 2001; Lyu, Rangarajan et al. 2002). A series-parallel system has a total of s independent subsystems arranged in series; and for the i-th subsystem, it can have up to $n_{max,i}$ functionally equivalent components arranged in parallel. Each component potentially varies in reliability, cost, weight and other characteristics. A subsystem can work properly if at least one of its components is operational. The n_i components are selected from m_i available component types where multiple copies of each type can be selected. A typical structure of Series-Parallel system is illustrated in Figure 5.3. Increasing the number of redundant components will increase the system reliability, but also increases cost and

weight. The goal is to optimally allocate the redundant components while balancing multiple competing objectives.

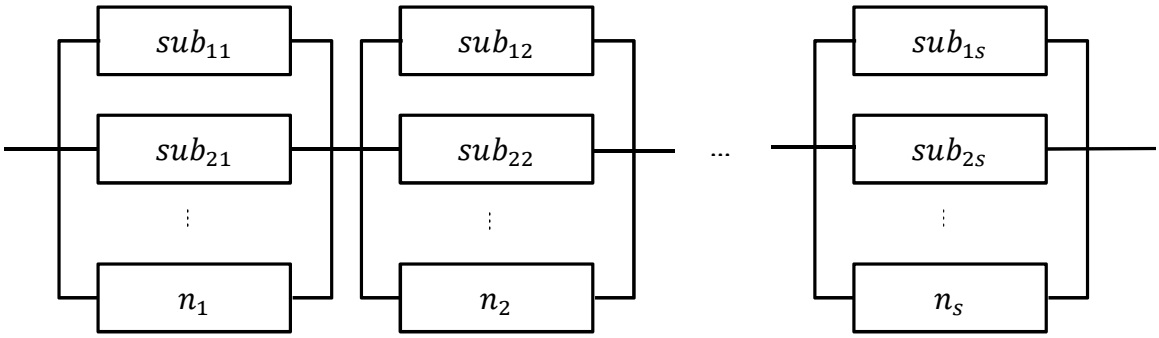


Figure 5.3: General Series-Parallel redundancy system

The formulation of RAP in a multi-objective setting with reliability, cost and weight considerations can be presented as:

$$\max\left[\prod_{i=1}^{S}\left(1-\prod_{j=1}^{m_i}\left(1-r_{ij}\right)^{x_{ij}}\right)\right],\quad \min\left[\sum_{i=1}^{S}\sum_{j}^{m_i} c_{ji}x_{ij}\right],\quad min\left[\sum_{i=1}^{s}\sum_{j}^{m_i} w_{ij}x_{ij}\right]$$

s.t.

$$1 \le \sum_{j=1}^{m_i} x_{ij} \le n_{max,i} \quad for\ \forall i = 1,2,\dots,s$$

$$x_{ij} \in \{0,1,2,\dots,\}$$

where:
s: the number of subsystems
x_{ij}: decision variable, the number of the jth type component used in subsystem i
m_i: the total number of available components for subsystem i
$n_{max,i}$: the maximum number of components in parallel used in subsystem i
r_{ij}, c_{ij}, w_{ij}: the reliability, cost and weight of the jth available component for subsystem i

For the multi-objective RAP, the objectives are to determine the optimal design configuration that maximize system reliability, minimizes the total cost and minimizes the system weight for a Series-Parallel system.

5.3.1 Decomposition

In solving multi-objective RAP problems, decomposing the original problem into sub-problems and combining the solutions intelligently can greatly aid the process of constructing the Pareto-optimal solution set. This is illustrated below.

We first decompose the original RAP into several sub-problems and solve each sub-problem using the modified adaptive ϵ-constraint method so as to identify all the non-dominated solutions for each sub-problem. Then, we sequentially filter each pair of non-dominated solution sets and pool the resulting solutions together to obtain the non-dominated solution set for the original RAP.

Decomposition is generally good for efficiency because of the reduced complexity of sub-problems. As a result, non-optimal solutions are filtered out early in the process when sub-problems are small and easy to solve. The details of the decomposition procedure are illustrated below.

Let us consider the original RAP P3:

$$\mathbf{P3}: max\left[\prod_{i=1}^{S}\left(1-\prod_{j=1}^{m_i}\left(1-r_{ij}\right)^{x_{ij}}\right)\right],\quad min\left[\sum_{i=1}^{S}\sum_{j}^{m_i}c_{ji}x_{ij}\right],\quad min\left[\sum_{i=1}^{s}\sum_{j}^{m_i}w_{ij}x_{ij}\right]$$

By changing the maximization of reliability to an equivalent minimization formulation, we can get P4 as:

$$\mathbf{P4}: min\left[\prod_{i=1}^{s}\left(\prod_{j=1}^{m_i}\left(1-r_{ij}\right)^{x_{ij}}\right)\right],\quad min\left[\sum_{i=1}^{s}\sum_{j}^{m_i}c_{ji}x_{ij}\right],\quad min\left[\sum_{i=1}^{s}\sum_{j}^{m_i}w_{ij}x_{ij}\right]$$

Further, by using log transformation, we can change the product terms of reliability into summation terms of $log(reliability)$ and have P5 as:

$$\mathbf{P5}: min\left[\sum_{i=1}^{s} log\left(\prod_{j=1}^{m_i}(1-r_{ij})^{x_{ij}}\right)\right],\ \ min\left[\sum_{i=1}^{s}\sum_{j}^{m_i} c_{ji}x_{ij}\right],\ \ min\left[\sum_{i=1}^{s}\sum_{j}^{m_i} w_{ij}x_{ij}\right]$$

Finally, P5 can be presented as:

$$\mathbf{P6}: min\left\{\sum_{i=1}^{s}\left\{min\left[log\left(\prod_{j=1}^{m_i}(1-r_{ij})^{x_{ij}}\right)\right],\ \ min\left[\sum_{j}^{m_i} c_{ji}x_{ij}\right],\ \ min\left[\sum_{j}^{m_i} w_{ij}x_{ij}\right]\right\}\right\}$$

Thus, the Pareto-optimal solution set of P3 is identical to that of the solution sets from P4, P5 and P6.

P6 can be decomposed into several subsystems. The Pareto-optimal solution set of P6 can be obtained by combining the Pareto-optimal solutions sets of each subsystem. The procedure is illustrated in Figure 5.4.

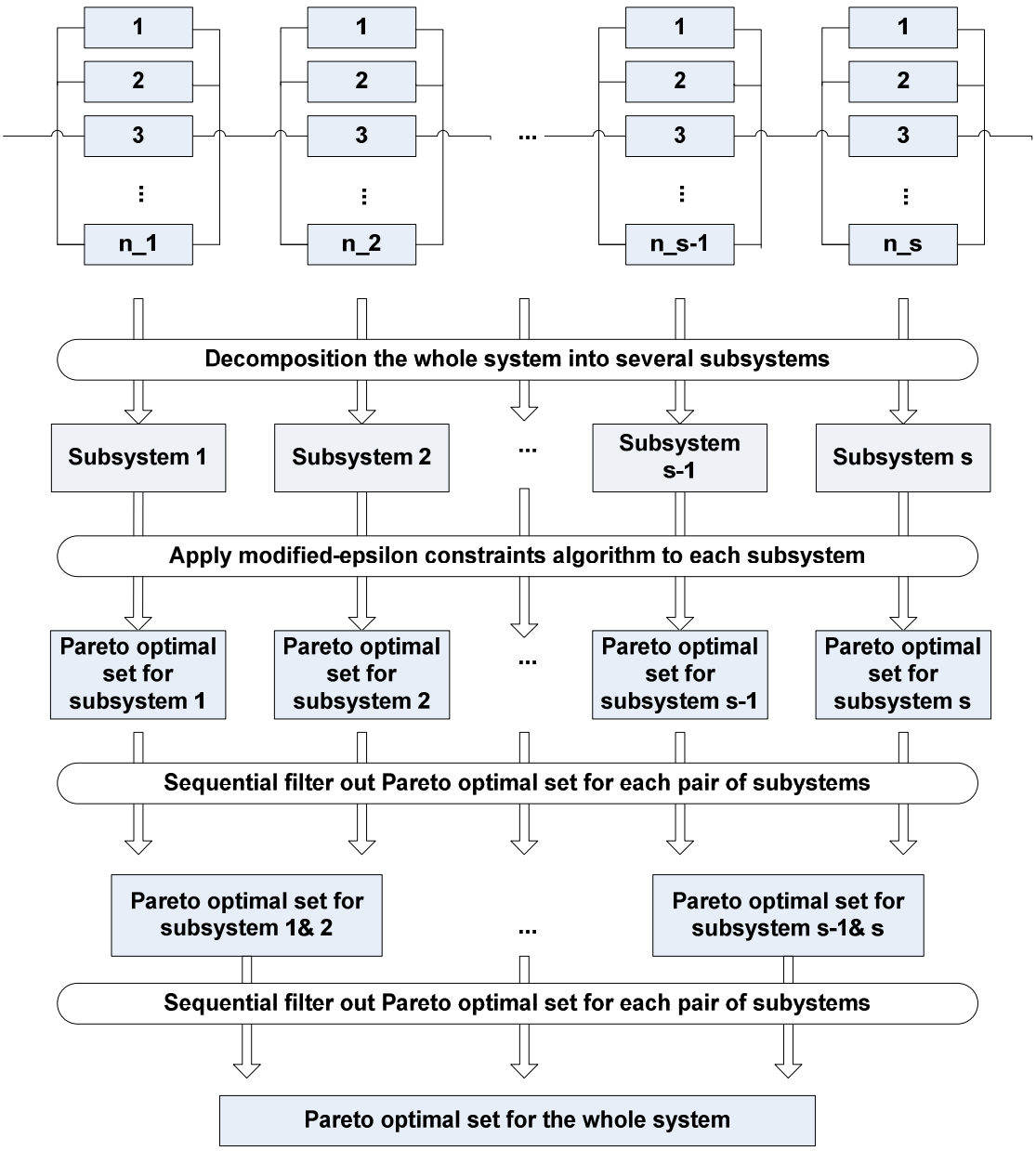


Figure 5.4: Framework of decomposition based modified adaptive ϵ-constraint method for RAP problems

For subsystem i, we solve the following sub-problem:

$$\mathbf{P7}: \left\{ \begin{array}{c} min\, f_1 = \left[\sum_{j}^{m_i} c_{ji} x_{ij}\right],\ min\, f_2 = \left[\sum_{j}^{m_i} w_{ij} x_{ij}\right], \\ \min f_3 = \left[log\left(\prod_{j=1}^{m_i} \left(1 - r_{ij}\right)^{x_{ij}}\right)\right] \end{array} \right\}$$

In the proposed method, we assume that all objective functions are linear integer functions. However, in the RAP problem, the reliability objective (or the unreliability objective in the transformed formulation) is not an integer. In order to implement the proposed method, we need to make some additional modifications. First, the non-integer objective function (unreliability) is always set to be the last objective function (f_3) as shown in P7. In the adaptive ϵ-constraint method, the weights for each objective function are selected so as to guarantee that the maximal increment of any current objective function is always less than the minimal increment of previous objective function. In the current example, the weight w_2 in P2 guarantees that the maximal increment of f_2 is always less than the minimal increment of f_1 (the minimal increment of f_1 is 1 by default due to the fact that f_1 is an integer function); similarly, the weight w_3 in P2 guarantees that the maximal increment of f_3 is always less than the minimal increment of f_2. Since f_3 is the last objective function and deals with reliability, even though it is a non-integer, given that there is no additional objective function beyond this objective, we can apply the proposed method (ignoring the fact the reliability objective is not integral). The other modification necessary to apply the proposed decomposition-based adaptive ϵ-constraint method to RAP problems is that, in step 16 of Algorithm 5.2, $\varepsilon_3 \coloneqq \max(f_3(x)) - 1$ is replaced by $\varepsilon_3 \coloneqq \max(f_3(x)) - \Delta$, here Δ is a very small number.

Numerical Example

In this section, we experimentally compare the proposed method with meta-heuristic based approach NSGA-II on a RAP for a Series-Parallel system example taken from literature (Taboada and Coit 2006). This Series-Parallel system consists of three subsystems ($s = 3$), with an option of five, four and five type of components in each

subsystem ($m_1 = 5, m_2 = 4, m_3 = 5$) respectively. The maximum number of components is seven (($n_{max,1} = n_{max,2} = n_{max,3} = 7$) in each subsystem. Table 5.4 lists the component parameters for each subsystem.

Table 5.4: Component parameters for each subsystem

Component Type *j*	**Subsystem *i***								
	1			**2**			**3**		
	Rel.	***Cost***	***Weight***	***Rel.***	***Cost***	***Weight***	***Rel.***	***Cost***	***Weight***
1	.94	9	9	.97	12	5	.96	10	6
2	.91	6	6	.86	3	7	.89	6	8
3	.89	6	4	.70	2	3	.72	4	2
4	.75	3	7	.66	2	4	.71	3	4
5	.72	2	8				.67	2	4

The experiments of proposed method and NSGA-II were run on a HP desktop, with an AMD Quad-Core CPU operating at 2.3 GHz and 8 GB of RAM. The proposed method is coded in MATLAB®R2008b and NSGA-II was coded in C which is taken from the website of Deb's lab (Deb 2005). For NSGA-II, we vary its population size from 100 up to 5000, with *generation*=100, *crossover probability*= .8 and *mutation probability* = .008. Results from the proposed method and NSGA-II are shown in Table 5.5.

Table 5.5: Results from the proposed method and NSGA-II

Our method							
# of Pareto-optimal Points	**6,112**						
CPU Time (MATLAB R2008b)	1,728 seconds						
NSGA-II							
Population Size	100	200	500	1,000	2,000	4,000	5,000
# of Pareto Points	85	141	289	589	1,109	2,109	2,324
# of Pareto-optimal Points	15	27	66	214	558	1,247	**1,263**
CPU Time (C)	12s	28s	69s	177s	442s	1,231s	1,699s

From Table 5.5, we can see that, the proposed method identifies all 6,112 non-dominated points in 1,728 seconds. For NSGA-II, the number of Pareto points it identified increases as the population size increases. When the population size is 5,000, it identified 2,324 Pareto points. However, beside these Pareto points, only parts of them are Pareto-optimal points (1,263 out of 2,324). The disadvantages of NSGA-II are that, it cannot generate all Pareto-optimal points; and more important, it cannot guarantee all points it identified are Pareto-optimal. In other words, NSGA-II gives out a set of Pareto points, but it doesn't tell you which one is Pareto-optimal and which one is not. Figure 5.5 shows the 6,112 solutions identified by the proposed method in blue start and 1,263 solutions found by NSGA-II with population size 5,000 in red triangle. Figures 5.6 to figure 5.7 show the same results under two dimensional representations.

For this problem, the number of IP solved using adaptive ϵ-constraint method is 5773, while only 1680 IPs need to be solved using our proposed method to identify all the Pareto-optimal solutions, translating to an improvement of $\frac{5773-1680}{5773} = 70.9\%$. We can see that the proposed method outperforms the adaptive ϵ-constraint method significantly.

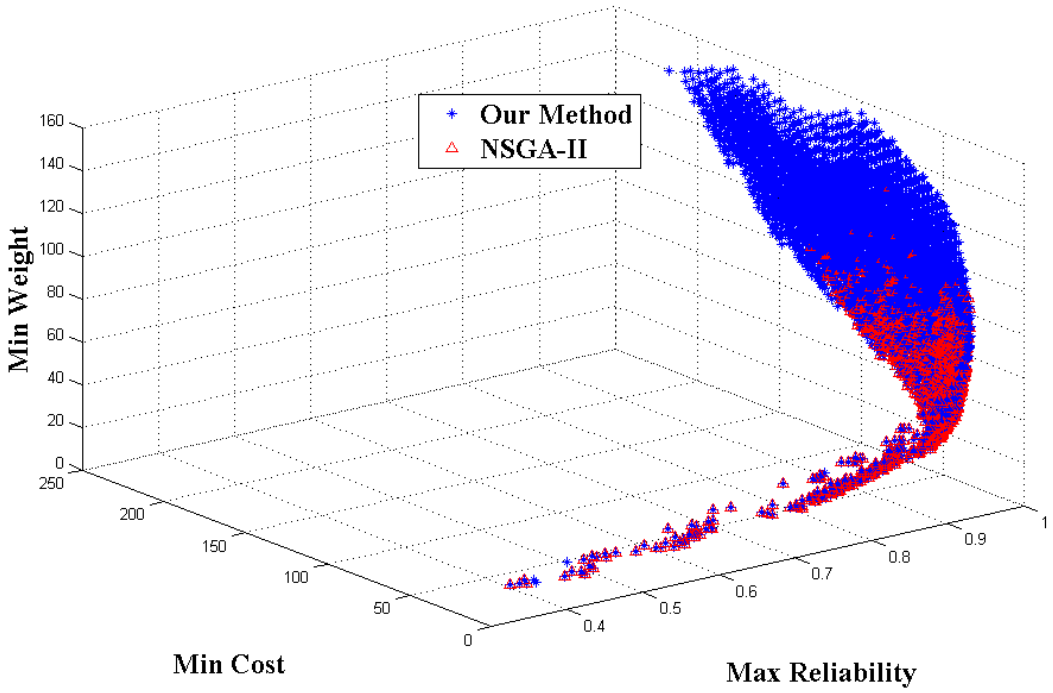


Figure 5.5: Pareto-optimal solution obtained by the proposed method and NSGA-II for the RAP problem

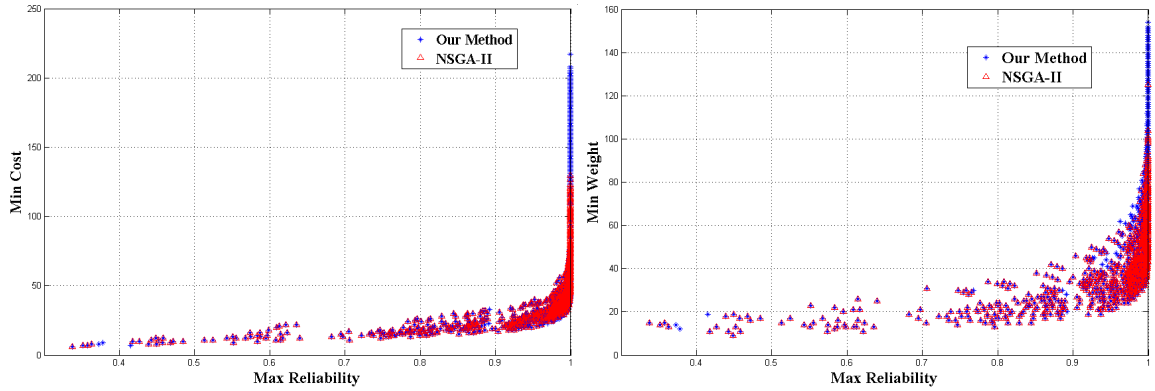


Figure 5.6: Pareto-optimal solutions plotted in the space of Reliability vs. Cost (left); Reliability vs. Weight (right)

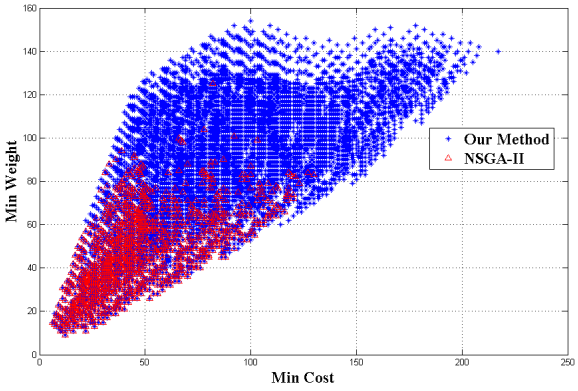


Figure 5.7: Pareto-optimal solutions plotted in the space of Cost vs. Weight

5.4 Case Example: Configuration Selection Problem

The configuration selection analysis evaluates configuration alternatives based on a large set of competing criteria, such as cost, weight and power et al. In this study, we implement the proposed modified adaptive ϵ-constraint method on the configuration selection problem and compare its results with that of adaptive ϵ-constraint method one so as to show its superiority.

The configuration selection problem considered in this study has three linear integer objective functions. They are minimizing the cost, weight and maximizing the power (combat power). The system consists of eleven subsystems. There are four options for choosing in subsystem 3 and subsystem 10, three options for choosing in subsystem 6. All other subsystems have five options and total there are 51 options. The cost, weight and power for each option are listed in Table 5.6. The interaction constraints include five in-compatible (IC) constraints, eight pre-requisite (PR) constraints and six co-requisite (CR) constraints. The in-compatible constraint(A, B) means if option A is chose, then option B cannot be chose or vice versa. The pre-requisite constraint (A, B) means if option B is chose, option A also must be chose while it doesn't happen in vice versa case.

The co-requisite constraint (A, B) means option A and option B have to be chose together. The five in-compatible constraints are: {(Opt9_1, Opt2_4), (Opt5_5, Opt3_1), (Opt9_5, Opt4_1), (Opt2_3, Opt7_3), (Opt8_3, Opt10_4)}; the eight pre-requisite are: {(Opt4_3, Opt2_3), (Opt1_2, Opt3_2), (Opt5_5, Opt4_4), (Opt11_4, Opt5_5), (Opt9_5, Opt7_4), (Opt5_4, Opt8_4), (Opt7_4, Opt8_4), (Opt10_3, Opt9_5)}; the six co-requisite constraints are: {(Opt7_1, Opt4_3), (Opt10_4, Opt5_4), (Opt8_4, Opt6_2), (Opt11_1, Opt7_4), (Opt7_5, Opt9_4), (Opt3_2, Opt10_2)}. And the problem can be formulated as:

$$\left\{\min C = \sum_{i=1}^{11}\sum_{j=1}^{M_j} x_{ij}c_{ij}\,, \min W = \sum_{i=1}^{11}\sum_{j=1}^{M_j} x_{ij}w_{ij}\,, \max P = \sum_{i=1}^{11}\sum_{j=1}^{M_j} x_{ij}p_{ij}\right\}$$

Subject to:

$$\begin{cases} \sum_{j=1}^{M_i} x_{ij} = 1, \qquad for\ i = 1,2,\dots,11 \\ x_{ij} - x_{lk} \le 0, \qquad ij, lk \in PR \\ x_{ij} + x_{lk} \le 1, \qquad ij, lk \in IC \\ x_{ij} - x_{lk} = 0, \qquad ij, lk \in CR \\ x_{ij} \in \{0,1\} \end{cases}$$

where M_i is the number of option for subsystem i.

Finally, 159 Pareto-optimal solutions are identified by both methods which are shown in Figure 5.8. Figures 5.9 and Figure 5.10 show also the two dimensional representation of the solutions.

Table 5.6: Component parameters for each subsystem

Subsystem	Options	Cost	Weight	Power	Subsystem	Options	Cost	Weight	Power
Subsys1	Opt1_1	42	36	97	Subsys7	Opt7_1	98	38	7
	Opt1_2	35	60	24		Opt7_2	78	80	48
	Opt1_3	50	99	49		Opt7_3	54	27	79
	Opt1_4	49	62	67		Opt7_4	1	10	47
	Opt1_5	29	1	33		Opt7_5	84	38	47
Subsys2	Opt2_1	51	50	33	Subsys8	Opt8_1	28	25	8
	Opt2_2	54	93	21		Opt8_2	66	96	10
	Opt2_3	60	51	92		Opt8_3	1	32	26
	Opt2_4	92	57	66		Opt8_4	51	69	99
	Opt2_5	81	19	64		Opt8_5	15	95	52
Subsys3	Opt3_1	12	32	69	Subsys9	Opt9_1	95	19	93
	Opt3_2	70	28	71		Opt9_2	95	58	60
	Opt3_3	39	48	77		Opt9_3	9	18	96
	Opt3_4	48	21	55		Opt9_4	23	39	96
Subsys4	Opt4_1	66	86	53		Opt9_5	36	5	56
	Opt4_2	45	11	18	Subsys10	Opt10_1	54	64	43
	Opt4_3	7	29	84		Opt10_2	59	40	63
	Opt4_4	81	95	86		Opt10_3	61	5	11
	Opt4_5	100	32	99		Opt10_4	96	6	29
Subsys5	Opt5_1	82	66	94	Subsys11	Opt11_1	97	26	64
	Opt5_2	19	26	50		Opt11_2	60	6	87
	Opt5_3	72	21	34		Opt11_3	61	46	7
	Opt5_4	19	17	46		Opt11_4	87	84	26
	Opt5_5	58	62	95		Opt11_5	62	24	4
Subsys6	Opt6_1	56	35	36					
	Opt6_2	61	32	80					
	Opt6_3	5	72	13					

For the adaptive ϵ-constraint method, it needs to solve 1702 IP, while the proposed modified adaptive ϵ-constraint method only solves 454 IP. It improves the algorithm efficiency by $\frac{1702-454}{1792} = 73.3\%$, which is a significant improvement.

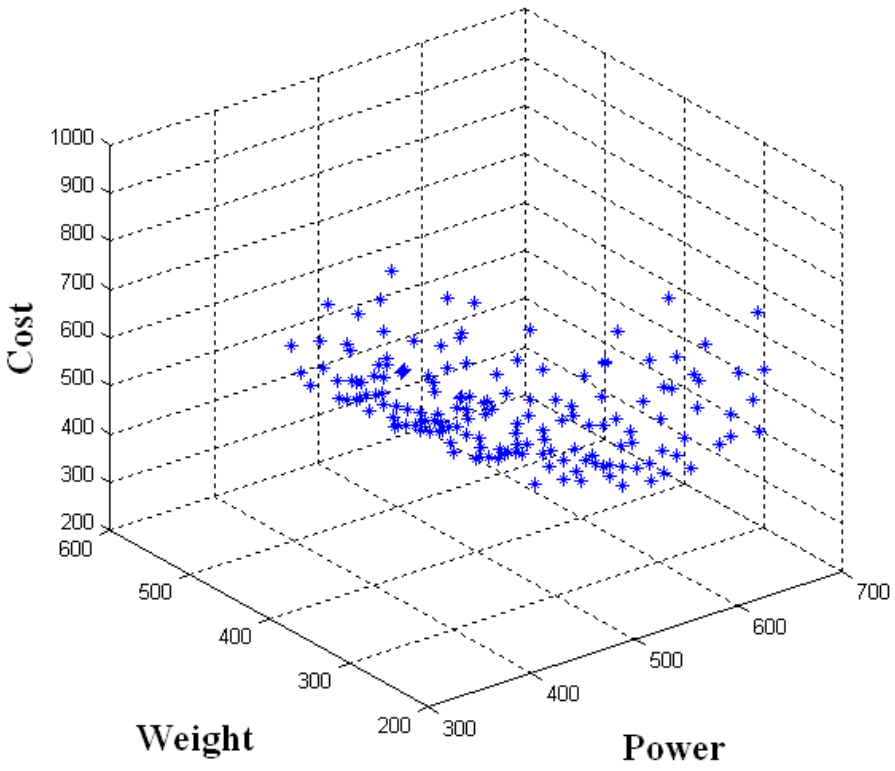


Figure 5.8: Cost vs Weight vs Power

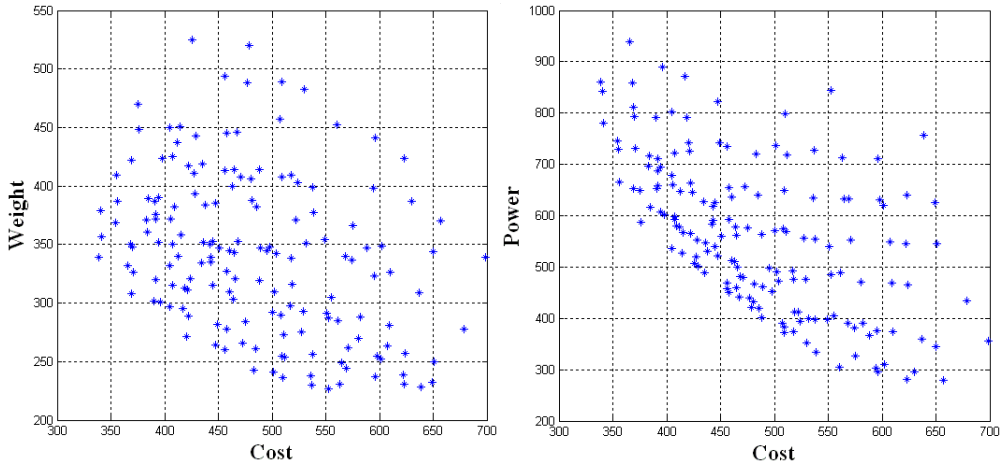


Figure 5.9: Cost vs Weight (left); Cost vs Power (right)

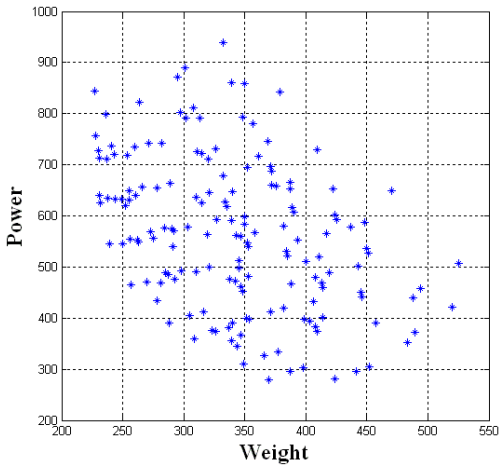


Figure 5.10: Weight vs Power

5.5 Conclusion

In this chapter, we propose a modified adaptive ϵ-constraint method to identify all Pareto-optimal solutions for linear multi-objective optimization problem. Compared with existing methods, our method not only improves the algorithm efficiency significantly but also is able to cope with at most one non-integer linear objective function. Based on the proposed optimization method, we present a system design optimization framework for simple structure systems (series/parallel) which have close-form reliability (availability) formulas. The presented framework is applied to a special case of system design optimization problem, the RAP of Series-Parallel system to evaluate its performance. Furthermore, the proposed optimization method is evaluated on a configuration selection problem taken from our industrial partner. Both numerical results show that the proposed method outperforms the existing methods, but the improvement is problem dependable.

Chapter 6 Conclusion

6.1 Contributions

The contributions made in this dissertation are as follows:

1. We have proposed a new CTBN formalism for RAM modeling of dynamic repairable systems. The previous work considered two main approaches: Dynamic Bayesian networks and Markov chain based models. Compared with DBNs, the CTBN framework belongs to the class of event-based BN formalisms which is essentially used for modeling reversible processes. Thus the advantage of CTBNs, over DBNs is their ability to handle various repair polices. Compared with Markov chain models, where each system state explicitly describes the state of all the system variables, CTBNs do not suffer from the state space explosion problem of Markov chain models. In particular, the CTBN is a local-state model, where the state of the current node is only dependent on its parent node. In short, CTBNs are more efficient and tractable than Markov chain models and also more suitable to model dynamic behavior among components than are DBNs.
2. Based on the CTBN framework, we have proposed CTBN constructs for the static (Fault tree) and dynamic (Dynamic Fault tree) gates, typically found in reliability tools. Furthermore, we have shown how to model different repair policies using CTBNs. The CTBN RAM modeling framework is applied to model three case examples to estimate the system reliability and availability: the Cardiac system, the ground vehicle system and the fleet of vehicles system.

3. Based on the proposed CTBN RAM modeling formalism and NSGA-II, we present a system design optimization framework for dynamic repairable systems. The CTBNs are employed to estimate system reliability (availability) while the meta-heuristic optimization method NSGA-II is used to solve the black-box multi-objective optimization problem. The CTBNs and NSGA-II based system design framework is applied to a ground vehicle system to identify the Pareto solution set.
4. We propose a modified adaptive ϵ-constraint method which is able to identify all Pareto-optimal solutions for integer linear multi-objective optimization problem. Compared with the existing ϵ-constraint method, the proposed one improves the algorithm efficiency significantly by avoiding solving for the duplicate solutions. Furthermore, in the application to the RAP, the proposed method is adjusted to cope with linear multi-objective optimization problem with one non-integer objective function.
5. We regard the typical system design problem, the RAP of Series-Parallel systems, as a multi-objective optimization problem. Consequently, we utilize the proposed modified adaptive ϵ-constraint method and the decomposition scheme to cope with this Multi-Objective Redundancy Allocation Problem (MORAP). Compared with existing MORAP methods, the main advantage of the proposed method is that it is able to identify all Pareto-optimal solutions. The modified adaptive ϵ-constraint method is evaluated on the RAP of Series-Parallel systems and the configuration selection problem.

6.2 Future Research

The following is a list of avenues for future research:

1. The CTBN RAM modeling framework, as defined in this work, is primarily geared towards the modeling of systems whose component failure time and repair time follows exponential distribution. The Markovian property assumption restricts the expressive power of CTBNs to model non-exponential distribution over time. There are two possible ways to cope with non-exponential distribution processes. The first one is to use the Phase-type distribution (Nodelman 2007) to map the non-exponential process into several exponential processes; the second one is to add hidden variables (Nodelman 2007) as parent nodes to the non-exponential nodes to control their evolution. However, both of these approaches will add complexity and computation burden to the model. Thus, finding a new and efficient way for CTBNs to cope with non-exponential distribution processes would allow a broader use for CTBNs in the RAM modeling applications.
2. In the CTBNs and NSGA-II based multi-objective system design optimization framework, the meta-heuristic based NSGA-II has limitations of not being able to identify all Pareto-optimal solution and to guarantee the solutions are Pareto-optimal. It would be very intriguing to find an optimization method which can break these limitations for the black-box multi-objective system design problem.
3. Compared with the existing ϵ-constraint method, the modified adaptive ϵ-constraint method, as proposed in this work, is able to improve the algorithm efficiency by avoiding solving the majority of duplicate solutions. However, the proposed method, while reducing the identification of duplicate solutions

significantly, cannot avoid all duplicated solutions. The investigation of incorporating a more sophisticated checking mechanism into the proposed method so that it can avoid all the duplicated solutions is definitely a worthwhile task to undertake in the future.

REFERENCES

Bellman, R. and S. Dreyfus (1958). "Dynamic Programming and the Reliability of Multicomponent Devices." Operations Research **6**(2): 200-206.

Billionnet, A. (2008). "Redundancy allocation for series-parallel systems using integer linear programming." Ieee Transactions on Reliability **57**(3): 507-516.

Bobbio, A., L. Portinale, et al. (2001). "Improving the analysis of dependable systems by mapping fault trees into Bayesian networks." Reliability Engineering & System Safety **71**(3): 249-260.

Bobbio, A. and D. C. Raiteri (2004). Parametric fault trees with dynamic gates and repair boxes. Reliability and Maintainability, 2004 Annual Symposium (RAMS 2004), Los Angeles.

Boudali, H., P. Crouzen, et al. (2007). Dynamic fault tree analysis using input/output interactive markov chains. the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. UK.

Boudali, H. and J. B. Dugan (2005). "A discrete-time Bayesian network reliability modeling and analysis framework." Reliability Engineering & System Safety **87**(3): 337-349.

Boudali, H. and J. B. Dugan (2005). "A new Bayesian network approach to solve dynamic fault trees." Annual Reliability and Maintainability Symposium, 2005 Proceedings: 451-456.

Boudali, H. and J. B. Dugan (2006). "A continuous-time Bayesian network reliability modeling, and analysis framework." Ieee Transactions on Reliability **55**(1): 86-97.

Bulfin, R. L. and C. Y. Liu (1985). "Optimal Allocation of Redundant Components for Large Systems." Ieee Transactions on Reliability **34**(3): 241-247.

Busacca, P. G., M. Marseguerra, et al. (2001). "Multiobjective optimization by genetic algorithms: application to safety systems." Reliability Engineering & System Safety **72**(1): 59-74.

Chankong, V. and Y. Haimes (1983). Multiobjective Decision Marking Theory and Methodology. New York., Elsevier Science.

Clifton, R. E. J. (1990). Apparatus for preventing truck roll over in the event of failure of its suspension system. U. patent. USA.

Coello Coello, C. A. (2006). "Evolutionary multi-objective optimization: an historical view of the field." Computational Intelligence Magazine **1**: 28-36.

Cohn, I., T. El-Hay, et al. (2009). Mean field variational approximation for continuous-time Bayesian networks. The Twenty-Fifth Conference Conference on Uncertainty in Artificial Intelligence Montreal, QC, Canada 91-100.

Coit, D. W. (2001). "Cold-standby redundancy optimization for nonrepairable systems." Iie Transactions **33**(6): 471-478.

Coit, D. W. and A. E. Smith (1996). "Reliability optimization of series-parallel systems using a genetic algorithm." Ieee Transactions on Reliability **45**(2): 254-&.

Deb, K. (2005). "Multi-objective NSGA-II code in C." from http://www.iitk.ac.in/kangal/codes.shtml. .

Deb, K., A. Pratap, et al. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II." Ieee Transactions on Evolutionary Computation **6**(2): 182-197.

Dhingra, A. K. (1992). "Optimal Apportionment of Reliability and Redundancy in Series Systems under Multiple Objectives." Ieee Transactions on Reliability **41**(4): 576-582.

Dugan, J. B. (2000). "Galileo: A tool for dynamic fault tree analysis." Computer Performance Evaluation, Proceedings **1786**: 328-331.

Dugan, J. B., S. J. Bavuso, et al. (1992). "Dynamic Fault-Tree Models for Fault-Tolerant Computer-Systems." Ieee Transactions on Reliability **41**(3): 363-377.

Dugan, J. B., S. J. Bavuso, et al. (1993). "Fault-Trees and Markov-Models for Reliability-Analysis of Fault-Tolerant Digital-Systems." Reliability Engineering & System Safety **39**(3): 291-307.

Dugan, J. B., K. J. Sullivan, et al. (2000). "Developing a low-cost high-quality software tool for dynamic fault-tree analysis." Ieee Transactions on Reliability **49**(1): 49-59.

Fan, Y. and C. R. Shelton (2008). Sampling for approximate inference in continous time Bayesian networks. The Tenth International Symposium on Artificial Intelligence and Mathmatics. Fort Lauderdale, FL, USA.

Fonseca, C. M. and P. J. Fleming (1993). Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. The fifth international conference on genetic algorithms. San Mateo California**:** 416-423.

Fyffe, D. E., W. W. Hines, et al. (1968). "System reliability allocation and a computational algorithm." IEEE Transaction on Reliability **17**: 64-69.

GDLS. (2011). "Stryker ground combat vehicle family." Retrieved April 19, 2011, from http://www.gdls.com/programs?option=com_gdlsmain&id=32&Itemid=37.

Holland, J. (1975). Adaption in natural and artificial systems, U. Michigan Press.

Horn, J., N. Nafpliotis, et al. (1994). A niched pareto genetic algorithm for multiobjective optimization. the first IEEE conference on evolutionary computation. Piscataway, NJ**:** 82-87.

John, C., H. Loy, et al. (2003). Through life support for building services system. Worldwide CIBSE/ASHERAE, Gathering of the Building Services Industry. Florence**:** 24-26.

Kulturel-Konak, S., D. W. Coit, et al. (2008). "Pruned Pareto-optimal sets for the system redundancy allocation problem based on multiple prioritized objectives." Journal of Heuristics **14**(4): 335-357.

Kulturel-Konak, S., A. E. Smith, et al. (2003). "Efficiently solving the redundancy allocation problem using tabu search." Iie Transactions **35**(6): 515-526.

Levitin, G. and A. Lisninaski (2001). "A New Approach to Solving Problem of Multistate System Reliability Optimization." Quality and Reliability Engineering International **17**: 93-94.

Liang, Y. C. and A. E. Smith (2004). "An ant colony optimization algorithm for the redundancy allocation problem (RAP)." Ieee Transactions on Reliability **53**(3): 417-423.

Lyu, M. R., S. Rangarajan, et al. (2002). "Optimal allocation of test resources for software reliability growth modeling in software development." Ieee Transactions on Reliability **51**(2): 183-192.

Marseguerra, M., E. Zio, et al. (2004). "A multiobjective genetic algorithm approach to the optimization of the technical specifications of a nuclear safety system." Reliability Engineering & System Safety **84**(1): 87-99.

Mettas, A. and W. Zhao (2004). Modeling and Analysis of Complex Repairable Systems, ReliaSoft Corporation.

Misra, K. B. (1971). "Dynamic Programming Formulation of the Redundancy Allocation Problem." International Journal of Mathematical Education in Science and Technology **2**(3): 207 - 215.

Misra, K. B. (1972). "Reliability optimization of a series-parallel system." IEEE Trans. Reliability **R-21**(4): 230-238.

Misra, K. B. and U. Sharma (1991). "An Efficient Algorithm to Solve Integer-Programming Problems Arising in System-Reliability Design." Ieee Transactions on Reliability **40**(1): 81-91.

Montani, S., L. Portinale, et al. (2005). Dynamic Bayesian Networks for Modeling Advanced Fault Tree Features in Dependability Analysis. European Safety and Reliability Conference (ESREL 2005), Tri City, Poland.

Montgomery, A. D. (1996). Logistics, an integral part of cost efficient space operations. Florida, 32899, National Aeronautics and Space Administration (NASA), Kennedy Space Center.

Nelson, J. R. (1977). Life Cycle Analysis of Aircraft Turbine Engines. Santa Monica, CA, The Rand Corporation.

Nodelman, U. (2007). Continuous Time Bayesian Networks. PhD Dissertation, STANFORD UNIVERSITY.

Nodelman, U., D. Koller, et al. (2005). Expectation propagation for continous time Bayesian networks. The Twenty-First International Conference on Uncertainty in Artificial Intelligence. Edinburgh, Scotland, UK**:** 431-440.

Oiddir, R., M. Rahli, et al. (2004). "Ant colony optimization for new redesign problem of multi-state electrical power systems." Journal of Electrical Engineering **55** (3-4): 57-63.

Ozlen, M. and M. Azizoglu (2009). "Multi-objective integer programming: A general approach for generating all non-dominated solutions." European Journal of Operational Research **199**(1): 25-35.

Painton, L. and J. Campbell (1995). "Genetic Algorithms in Optimization of System Reliability." Ieee Transactions on Reliability **44**(2): 172-178.

Portinale, L. and R. D. C. (2009). A GSPN Semantics for Continuous Time Bayesian Networks with Immediate Nodes, Computer Science Department, UPO,.

Portinale, L., D. C. Raiteri, et al. (2010). "Supporting reliability engineers in exploiting the power of Dynamic Bayesian Networks." International Journal of Approximate Reasoning **51**(2): 179-195.

Raiteri, D. C., G. Fraceschinis, et al. (2004). Repariable Fault Tree for the automatic evaluation of repair policies. International Conference on Dependable Systems and Networks, Florence, Italy.

Ramirez-Marquez, J. E. and D. W. Coit (2004). "A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems." Reliability Engineering & System Safety **83**(3): 341-349.

Rashika, G. and A. Manju (2006). "Penalty guided genetic search for redundancy optimization in multi-state series-parallel power system." Journal of combinatorial optimization **12**(3): 257-277.

Ravi, V., B. S. N. Murty, et al. (1997). "Nonequilibrium simulated annealing-algorithm applied to reliability optimization of complex systems." Ieee Transactions on Reliability **46**(2): 233-239.

Reliasoft. (2011). "Availability." Retrieved April 19, 2011, from http://www.weibull.com/SystemRelWeb/availability.htms.

Ren, Y. S. and J. B. Dugan (1998). "Design of reliable systems using static & dynamic fault trees." Ieee Transactions on Reliability **47**(3): 234-244.

Salazar, D., C. M. Rocco, et al. (2006). "Optimization of constrained multiple-objective reliability problems using evolutionary algorithms." Reliability Engineering & System Safety **91**(9): 1057-1070.

Saria, S., U. Nodelman, et al. (2007). Reasoning at the right time granularity. The Twenty-third Conference on Uncertainty in Artificial Intelligence. Univ. of BC, Vancouver, BC, Canada**:** 421-430.

Schaffer, J. (1985). Multiple objective optimization with vector evaluated genetic algorithms. Genetic algorithms and their applications: the first international conference on genetic algorithms. Hillsdale, NJ**:** 93-100.

Sharma, U. and K. B. Misra (1990). "An efficient algorithm to solve integer programming problem in reliability optimization." International Journal of Quality & Reliability Management **7**(5): 44-46.

Shelton, C. R., Y. Fan, et al. (2010). "Continuous Time Bayesian Network Reasoning and Learning Engine." Journal of Machine Learning Research **11**: 1137-1140.

Srinivas, N. K. and A. Deb (1994). "Multiobjective optimization using non-dominated sorting in genetic algorithms." J Evol Comput **2**(3): 221-248.

Taboada, H. A., F. Baheranwala, et al. (2007). "Practical solutions for multi-objective optimization: An application to system reliability design problems." Reliability Engineering & System Safety **92**(3): 314-322.

Taboada, H. A. and D. W. Coit (2006). MOEA-DAP: A New Multiple Objective Evolutionary Algorithm for Solving Design Allocation Problems, Rutgers University IE Working Paper

Taboada, H. A., J. F. Espiritu, et al. (2008). "MOMS-GA: A multi-objective multi-state genetic algorithm for system reliability optimization design problems." Ieee Transactions on Reliability **57**(1): 182-191.

Tavakkoi-Moghaddam, R., J. Safari, et al. (2008). "Reliability optimization of series-parallel systems with a choice of redundancy strategies using a genetic algorithm." Reliability Engineering & System Safety **93**(4): 550-556.

Tian, Z. G. and M. J. Zuo (2006). "Redundancy allocation for multi-state systems using physical programming and genetic algorithms." Reliability Engineering & System Safety **91**(9): 1049-1056.

Tillman, F. A., C. L. Hwang, et al. (1977). "Determining Component Reliability and Redundancy for Optimum System Reliability." Ieee Transactions on Reliability **26**(3): 162-165.

Walter, J. (2001). John Walter (2001), Automotive cooling system component interactions, Dissertation, Texas Tech University, Texas Tech University.

Weber, P. and L. Jouffe (2003). Reliability modeling with Dynamic Bayesian Networks. 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SAFEPROCESS'03), Washington, D.C., USA.

Zafiropoulos, E. P. and E. N. Dialynas (2004). "Reliability and cost optimization of electronic devices considering the component failure rate uncertainty." Reliability Engineering & System Safety **84**(3): 271-284.

Zeleny, M. (1982). Multiple criteria decision making, McGraw-Hill.

Zia, L. and D. W. Coit (2010). "Redundancy Allocation for Series-Parallel Systems Using a Column Generation Approach." Ieee Transactions on Reliability **59**(4): 706-717.

Zitzler, E. and L. Thiele (1999). "Multiobjective evolutionary algorithms: A comparative case study and the Strength Pareto approach." Ieee Transactions on Evolutionary Computation **3**(4): 257-271.

ABSTRACT

NOVEL MODELS AND ALGORITHMS FOR SYSTEMS RELIABILITY MODELING AND OPTIMIZATION

by

DINGZHOU CAO

August 2011

Advisors: Dr. Ratna Babu Chinnam and Dr. Alper Murat

Major: Industrial Engineering

Degree: Doctor of Philosophy

Recent growth in the scale and complexity of products and technologies in such the defense and other industries as the defense is has become a challenge for attaining cost efficiency in challenging product development, realization, and sustainment costs. Uncontrolled costs and routine budget overruns are forcing companies causing all parties involved to seek become leaner in their product development processes and treatment to treat products' of reliability, availability, and maintainability of the system as a true "design parameter". To this effect, accurate estimation and management of the system reliability of a design during the "earliest stages" of new product development is critical not only critical for managing product development and manufacturing costs but also to for controlling life cycle costs (LCC). In this regard, the overall objective of this research study is to develop an integrated framework for "design for reliability" (DFR) upfront during the upfront product development by treating reliability as a design parameter. The

aim here is to develop the theory, methods, and tools necessary for: 1) accurate assessment of system reliability and availability and 2) optimization of the design to meet system reliability targets. In modeling the system reliability and availability, we aim to address the limitations of existing methods, in particular the Markov chains method and the Dynamic Bayesian Network approach, by incorporating a Continuous Time Bayesian Network framework for more effective modeling of sub-system/component interactions, dependencies, and various repair policies. We also propose a multi-object optimization scheme to aid the designer in obtaining identifying the optimal design(s) with respect to system reliability/availability targets and other system design requirements. In particular, the optimization scheme would entail optimal selection of sub-system and component alternatives. The theory, methods, and tools to be developed will be are extensively tested and validated using simulation test-bed data and actual case studies from our industry applications.

AUTOBIOGRAPHICAL STATEMENT

DINGZHOU CAO

Dingzhou Cao did his undergraduate studies at Jinan University, Guangzhou, P. R. China, where he received his Bachelor of Science degree in Mathematics and Applied Mathematics in 2004. He received his Master of Science degree in Applied Mathematics in 2006 from the same school. Since 2007, Dingzhou has been studying at Wayne State University (WSU) for a Ph.D. in the Department of Industrial and System Engineering (ISE). During his Ph.D. program, he has worked as graduate research assistant in the ISE department at WSU.