# DESIGN SPACE EXPLORATION TECHNIQUES FOR RELIABLE SYSTEM SYNTHESIS

Thesis submitted to
Indian Institute of Technology Kharagpur
for the partial fulfillment of the requirements
for the award of the degree of

Master of Technology
In
Computer Science & Engineering
by

Vishnuvardhan P
13CS60D03
M.Tech CSE

Under the guidance of
Prof Soumyajit Dey


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

# DESIGN SPACE EXPLORATION TECHNIQUES FOR RELIABLE SYSTEM SYNTHESIS

Thesis submitted by

Vishnuvardhan P
13CS60D03
M.Tech CSE

Approved by

Prof Soumyajit Dey
(Supervisor)

Prof Rajib Mall
(Head of the Department)


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

# Declaration

I, **VishnuVardhan P (13CS60D03)** hereby declare that the thesis on **"Design Space Exploration Techniques for Reliable System Synthesis"** is a presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with due reference to the literature, and acknowledgement of collaborative research and discussions. The work was done under the guidance of **Dr. Soumyajit Dey**, at Indian institute of Technology, Kharagpur.

_____

VishnuVardhan P


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

# Certificate

This is to certify that the project entitled **"Design Space Exploration Techniques for Reliable System Synthesis"** is a bonafide record of the work carried out by Mr. VishnuVardhan P (Roll No.13CS60D03) under my supervision and guidance for the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science & Engineering during the academic session 2013-2015 in the Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur.

The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree.

---

Prof. Soumyajit Dey

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Kharagpur, India 721302

# Acknowledgment

I express my sincere gratitude to **Dr. Soumyajit Dey**, Assistant Professor, Computer Science & Engineering Department, Indian Institute of Technology, Kharagpur for his supervision and guidance during the project. His valuable advices, encouragement, suggestions and friendly attitude towards me, during this work helped a long way in bringing this project report to this stage. I am also thankful to him for extending all kind of help and for providing the necessary facilities required during this work.

I am very much thankful to **Prof. Rajib Mall**, Head, Department of Computer Science & Engineering Department, IIT Kharagpur for providing necessary facilities during the research work.

I am thankful to Faculty Advisor **Prof. Sudeshna Sarkar** and all the faculty members of the department for their valuable suggestions, which helped me improve on this research work.

I express my sincere gratitude to **Mr. Saurav Kumar Ghosh**, for his valuable guidance, continuous support and encouragement, throughout the course of this research work.

Vishnuvardhan P

# Abstract

Design space exploration (DSE) refers to the systematic activity of exploring elements of design space. The design space is defined as the set of all possible design alternatives for the system. Here we deal with the exploration of design alternatives for reliable systems. A system is generally designed as a set of interconnected subsystems, each with its own reliability and associated cost attributes. The overall system reliability is a function of the subsystem reliability metrics. The cost of the system is the sum of the costs for all the subsystems.

The objective here is to obtain design alternative that achieves either the target reliability while minimizing the total cost, or maximize the reliability while using only the specified cost. The challenge here is the sheer size of the design space. An exhaustive search of all design space alternatives (reliability options for the subsystems) to optimize the system reliability with minimum cost may not be feasible at all the times because of performance reasons. This thesis aims at different approaches to get near optimal solutions for the optimization of system reliability and presents details of different approaches to allocate reliability values to the subsystems and its implementation as well as results. The final section of the thesis describles the reliability optimization problem using a degradation model with replenishment of components.

**Keywords: —** reliability, cost, optimization, DTMC, degradation, weibull, replenishment

# Contents


**1 Introduction** **1**
- 1.1 Motivation . . . 1
- 1.2 Objective . . . 2
- 1.3 DTMC Representation Of System Specification . . . 2
- 1.4 Organization of the thesis . . . 4
- 1.5 Software and Hardware Platform . . . 4

**2 Literature Survey** **5**
- 2.1 Introduction . . . 5
- 2.2 MINLP Problem . . . 6
- 2.3 Reliability Allocation Techniques . . . 6
  - 2.3.1 Quantifying The Cost Function . . . 6
- 2.4 Summary . . . 7

**3 Optimization of Software Reliability Using ILP Solvers** **8**
- 3.1 MINLP Problem . . . 9
- 3.2 System Specification to DTMC Representation . . . 9
  - 3.2.1 Modelling statements to DTMC . . . 9
  - 3.2.2 Modelling System Specification to DTMC . . . 11
- 3.3 System Reliability Calculation Using DTMC Representation . . . 12
- 3.4 Optimization Of System Reliability As MINLP . . . 12
- 3.5 Problem Formulation . . . 13
- 3.6 Optimization Sequence . . . 14
- 3.7 Various Approaches . . . 15
  - 3.7.1 BONMIN (Basic Open-source Nonlinear Mixed INteger programming) . . . 16
  - 3.7.2 SCIP (Solving Constraint Integer Programs) . . . 16
- 3.8 Implementation . . . 17

3.8.1 Example Problem . . . 17
3.9 Optimization Using BONMIN Algorithm . . . 20
3.10 Optimization Using SCIP Algorithm . . . 20
3.11 Optimization Using GA Algorithm . . . 21
3.12 Comparison Of Results . . . 23
3.13 Conclusion . . . 25

**4 Optimization of Software Reliability Using Component Ranking & Cost Rate Function Approach 26**
4.1 Component Ranking Approach . . . 27
4.1.1 Example Problem . . . 27
4.1.2 Ranking of Components . . . 28
4.1.3 Optimization Algorithm Using Component Ranking . . . 28
4.2 Cost Rate Function Approach . . . 30
4.2.1 Quantifying the Cost Rate Function . . . 31
4.2.2 The Feasibility Value, $f$ . . . 31
4.2.3 Relaxation of the Feasibility . . . 32
4.2.4 Optimization Algorithm Using Cost Function Approach . . . 32
4.3 Results & Discussion . . . 34
4.4 Conclusion . . . 36

**5 Optimization Based On Reliability Degradation Over Time 37**
5.1 Weibull Distribution . . . 38
5.1.1 Weibull Reliability . . . 39
5.1.2 Characteristics of the Weibull Distribution . . . 40
5.1.3 Problem Definition and Formulation using Weibull Distribution . . . 42
5.1.4 Random Allocation of Weibull Parameters . . . 43
5.1.5 Optimization Procedure . . . 45
5.1.6 Results . . . 45
5.1.7 Reliability Optimization for Set Of Target Reliabilities . . . 45
5.2 Reliability Optimization With Replenishment Of Components . . . 46
5.2.1 Problem Description . . . 46
5.2.2 Solution Methodology . . . 48
5.2.3 Replenishment Algorithm . . . 50
5.2.4 Results and Conclusion . . . 51

# List of Figures


1.1 DTMC Model . . . 3

3.1 DTMC representation of the statements . . . 10
3.2 Graph representation of software model . . . 11
3.3 Optimization Methodology . . . 15
3.4 Optimization Approaches . . . 16
3.5 MATLAB program for BONMIN Solver . . . 19
3.6 Calling the Solver in MATLAB . . . 19
3.7 Output of BONMIN Solver . . . 20
3.8 Output of SCIP Solver . . . 21
3.9 Matlab GA Solver Window . . . 22
3.10 Output of Matlab GA Solver . . . 23

4.1 comparision of Component Ranking and Branch&Bound . . . 35
4.2 comparision of CostRate Function and Branch&Bound . . . 35
4.3 comparision of Three Algorithms . . . 36

5.1 Reliability Curve . . . 37
5.2 Slope parameter characteristics . . . 40
5.3 Scale parameter characteristics [12] . . . 41
5.4 Location parameter characteristics [12] . . . 42
5.5 system reliability at different points . . . 47

# List of Tables


3.1 Adjacency Matrix . . . . . 17
3.2 Component Information . . . . . 18
3.3 Result of SCIP and BONMIN . . . . . 23
3.4 Result of GA . . . . . 24

4.1 Cost Values . . . . . 34
4.2 Reliability Values . . . . . 34

5.1 Cost Values . . . . . 43
5.2 $\beta$ Values . . . . . 44
5.3 $\gamma$ Values . . . . . 44
5.4 Result . . . . . 45
5.5 Result with replenishment - 8 components . . . . . 52
5.6 Result with replenishment - 8 components . . . . . 52
5.7 Result with replenishment - 10 components . . . . . 53

# Chapter 1

# Introduction

Design Space Exploration (DSE) is a most important factor in embedded system design. It is described as the exploration of all the design alternatives before the actual system implementation. The DSE is useful for many engineering tasks like rapid prototyping, optimization, and system integration. The challenge here is that generally a large system have a large design space (set of design alternatives), so exploring every design alternative proves to be costly. Here we consider reliability of the system as one of the design criteria and parameter for the exploration. A system is generally designed as a set of interconnected subsystems. Each subsystem has its own reliability and associated cost attributes. Each subsystem can have different alternatives i.e. reliability and cost attributes as they can be made by different technologies, different vendors etc. The overall system reliability is a function of the subsystem reliability metrics. The cost of the system is the sum of the costs for all the subsystems. The objective here is to obtain design alternative that achieves either the target reliability while minimizing the total cost, or maximize the reliability while using only the specified cost.

## 1.1 Motivation

The reliability of embedded systems (from small consumer applications to mission critical systems) is an important design criteria as failure of these systems can lead to disaster. As the system reliability is a function of the subsystem reliability metrics, there may be many different design alternatives that implement a given system specification with different reliability metrics for each subsystem. The designer can make a decision which system alternative to implement by exploring different implementations and judge for their quality. Therefore, during the exploration phase, many design alternatives have to be evaluated. Design alternatives may consist of different hardware component allocations for each subsystem, different scheduling policies implemented on shared resources etc.

But, a large system can have thousands of alternatives. So, exploring all the options may not be cost

effective solution. So we need to search for a solution which is cost effective in terms of exploration based on some inherent properties of the system. For example, if we consider a software specification, a function can be called multiple times at different places in the system. If each different function has associated reliability metrics, the function call that has been called most number of times may contribute more to the overall system reliability. So, we try to explore the design alternative based on some properties of the system in a cost effective manner. Here, we express a system as a software specification with reliability and cost metrics which in turn is modelled into Discrete Time Markov Chain (DTMC).

## 1.2 Objective

Considering all of the facts discussed above and pointing out the necessity of developing the technique, the following objectives have been entitled to the present study.

1. Model the given specification into a Mixed Integer Nonlinear Programming(MINLP) problem and analyze the performance for given specification using some standard optimization solvers that provide global solution as well as near global solution
2. Develop a search technique to find out the near optimal solution in cost effective manner for a reliable system synthesis. In detail, given a system that is expressed as specification with explicit reliability constructs, I need to find out the search technique for a specific reliability configuration of the system that maximizes the overall reliability within the specified cost. This requires allocation of the minimum required reliability for each component of a system in order to achieve a system reliability goal with minimum cost
3. The reliability of a system decreases over a period of time. Considering this fact, develop a methodology to find out the solution in cost effective manner for a system that must satisfy the given reliability over a period of time

## 1.3 DTMC Representation Of System Specification

A system specification models the system level behavior. The present reliability-aware system level specification frameworks lacks the expressiveness with respect to system-level behavior. RELSPEC [1], a reliability specification framework using a standard imperative programming language extended with probabilistic semantics and programming constructs is used as the specification language. The specification is modelled into Discrete Time Markov Chain (DTMC) and used as reliability model.

Markov Model is used to represent the architecture of the software & provides a mean for analyzing the reliability of software. Markov chain is simplistic markov model. It models the state of a system with a random variable based on the markov property: “The state of the system at time t+1 depends only on the state of the system at time t”. A formal definition and details are provided in [1].

The following example shows the sample example and associated DTMC model [1]. The individual expression statements define the components of a system and the while construct represent the loop in a system. The reliability values specified are generated from the range analysis of the variable 'x'. The explicit reliability and cost domains are provided for each of the component in the specification. The proposed algorithms uses this setup for finding the system reliability. The details are explained in next chapter.

```
void Example()
{
        int i, x, alt;
        x = 7 <0.99>;
        i = 1 <0.86>;
        while (i < 10) <0.97>
        {
                if (x > 3) <0.988>
                alt = 1 <0.91>;
                else
                alt = -1 <0.96>;
                if (i > 5) <0.97>
                i = i + 2 <0.987>;
                else
                i = i + 1 <0.9>;
        }
}
```

The DTMC model for the above example is:

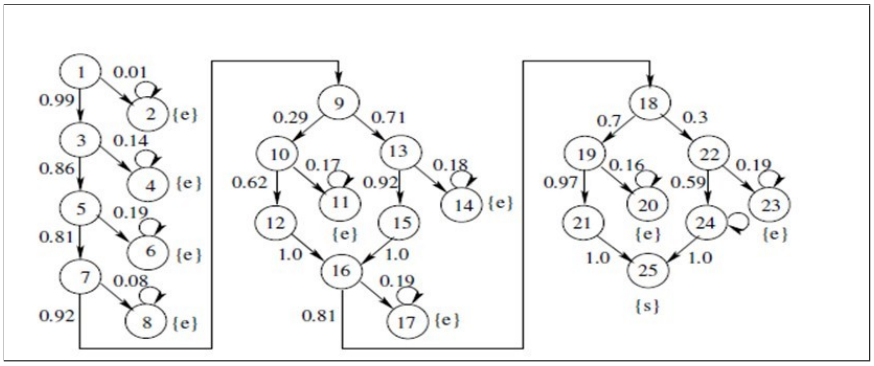


Figure 1.1: DTMC Model

## 1.4 Organization of the thesis

The thesis is broken into several chapters. The second chapter of the document details the review of literature that has been studied for this project. The third chapter provides the mapping of the given problem in to mixed integer non-linear problem (MINLP), its implementation details and provides analysis of different solvers performance. The fouth chapter describes two search techniques, their implementation and provides analysis of these two techniques along with comparison with golden solution.

Finally the fifth chapter describels weibull probability model and optimization of reliability of the system over time. The last chapter provides references.

## 1.5 Software and Hardware Platform

The proposed algorithms in this document are tested under following platform.

**Hardware:** Processor - intel core i5, Quadcore, 3.1GHz.
**Software:** C++, Matlab, Python, Ubuntu (kernel num. 3.5.0-23), OPTI Toolbox on windows platform

# Chapter 2

# Literature Survey

## 2.1 Introduction

This chapter deals with the background and review of literary information required to identify the information gap, to make the way out to the work and to discuss the findings. The first section of this chapter briefs on the Mixed Integer Non-linear Problem (MINLP) and using available open source MINLP solvers for the implementation. The second section of this chapter focuses on the use of cost function approach to improve the system reliability by allocation of appropriate reliability values to various subsystems.

A behavioral specification framework is provided in [1] that incorporates the aspect of reliability as an explicit design criteria which may aid in the design of safety critical embedded systems. It allows the designer to create reliability specification using a standard imperative programming language. This specification allows to provide reliability aspects explicitly for the programming constructs. It also provides least cost solution method using branch and bound algorithm which is used as golden solution for the proposed algorithms by the author.

Chao-Jung Hsu et al. [2] proposes a modular graph of the software system and incorporates path testing in to reliability estimation. Identifies all paths in the modular graph. Once all the paths are found, it calculate the path reliability from node n1 to n2 and estimate the software reliability from the path reliability. Sequence, branch and loop structure is well modelled.

## 2.2 MINLP Problem

Optimization problems generally can be modelled to Integer Linear Program (ILP) problems. As the inherent property of the problem under consideration is nonlinear, these problem can be modelled to MINLP form. Coremen [5] describes the ILP problem formulation and [6] describes the fundamentals of the MINLP problems. OPTI Toolbox [8] (OPTimization Interface Toolbox) is a free MATLAB toolbox for constructing Construct optimization problems in MATLAB and solve them using a range of supplied solvers. A range of open source and academic solvers are supplied for the Windows users. It provides the basic understanding of MINLP problem and examples solutions to solve the MINLP problems. It provides MINLP solvers like SCIP and BONMIN. It explains the method to represent the MINLP problems in Matlab specific files.

The techniques SCIP and BONMIN are also briefly explained in [9] and [10]. The algorithms they use are briefed and suggested where to use these techniques etc.

MATLAB uses genetic algorithm [7] for solving the MINLP problems and it provides GUI (Graphical User Interface) to select the algorithm and provide specific inputs to that algorithm. It provides the result in the result window with feasible and non-feasible solutions.

The COIN-OR [6] briefs about different open source and commercial MINLP as well as other type's problem solvers. It also provides libraries to be used and integrated with Matlab or any such tool.

## 2.3 Reliability Allocation Techniques

Mettas [3] provides the cost function formulation to be used in the nonlinear programming algorithm. It proposed general behavior of cost as the function of reliability. Mathematical formulation of cost function depend on certain parameters that must be supplied by the engineers. But, quantifying these parameters has not been an easy task. These problems are resolved in this paper by providing the general cost function whose parameters are easily quantified. A mechanism to increase the system reliability by fault tolerance is provided.

Mettas [mettas10] proposed a cost function approach. The objective of this function is to model an overall cost behavior for all types of components.

### 2.3.1 Quantifying The Cost Function

The cost function for each component $C_i$, in terms of the reliability, $R_i$ of each component is specified as:

$$C_i = f(R_i)$$

This function should:

- Look at the current reliability of the component,
- Look at the maximum possible reliability of the component, .
- Allow for different levels of difficulty (or cost) in increasing the reliability of each component.

The cost function $f_i$ would satisfy these three conditions:

1. $f_i$ is a positive definite function
2. $f_i$ is non-decreasing
3. $f_i$ increases at a higher rate for higher values of $R_i$.

The following default cost function is given by Metta[3].

$$C_i(R_i) = \exp^{(1-f)\frac{R_i - R_{min,i}}{R_{max,i} - R_i}}$$

where:

- $C_i(R_i)$ is the cost function as a function of component reliability.
- $R_{min,i}$ and $R_{max,i}$ are the minimum and maximum values of $R_i$ and $f_i$ is parameter ranging between 0 and 1 that represents the relative difficulty of increasing a component's reliability.

Cristiana Bolchini et al. [4] proposes a design methodology for embedded systems to introduce reliability-awareness. The approach extends the classical system-level design flow with reliability awareness and allows to specify reliability related requirements in the specification.

## 2.4 Summary

The author has studied the available scanty literature on the mentioned sections above and to the best of ability referred journals while presenting the contribution of various researchers. This study helped the author to investigate the problem and proposing the solutions for the given problem and presented in the subsequent chapters of the thesis.

# Chapter 3

# Optimization of Software Reliability Using ILP Solvers

A system is a collection of components. Optimization of system reliability given the behavioral model of the system can be specified as a Integer Non-Linear programming problem. The behavioral model of the system is represented as a software program that is a collection of statements. The individual components of a system represent a single statement or a compound statement. This chapter describes how the optimization of software reliability can be modeled in to MINLP(Mixed Integer Non-Linear programming) problem and solved using some of the available ILP solvers.

Problem Statement: "Assume we have a DTMC (Discrete Time Markov Chain) model of a software specification or model having c language like syntax. This model is represented by adjacency matrix and software components. The reliability and associated costs of each component and desired reliability of the system is given. The goal is to optimize the cost for the desired reliability of the software specification".

The proposed system shall take the DTMC model as the input and associated costs and reliabilities of each of the component in the system. Then it has to optimize the cost to get desired reliability of the system. The objectives of the software are:

- Minimize the cost of the system
- Maximize the reliability of the system

This chapter explains how to model the system specification in to a DTMC model, represent the model into an MINLP optimization problem and the methods that optimize the given specification using Integer Non-Linear Programming (ILP) problem. There are some available Integer Non Linear programming

solvers which take problem specification as the input and provide results of optimization. Three different types of ILP solvers are explained in this chapter along with an example.

## 3.1 MINLP Problem

A MINLP has the following form:

$$\begin{gathered} \text{minimize } f(x) \\ \text{such that} \\ A(x) \leqslant b \\ l \leqslant x \leqslant u \\ x \in R^n \\ x_i \in Z, j \in N \end{gathered}$$

where $f$ is a nonlinear objective function, $A(x) \leqslant b$ are the linear or non constraints, $l$ and $u$ are simple lower and upper bounds on the problem variables $x$, and $Z$ is the subset of indices denoting the variables required to be integer. A is a $m$ x $n$ sparse matrix, b is a $m$ x 1 vector. The goal is to minimize the objective function by selecting a value of $x$ that also satisfies all constraints.

## 3.2 System Specification to DTMC Representation

This section briefly describes how a system specification is modelled into a DTMC.

**Definition 1.** ***Discrete Time Markov Chain(DTMC):*** *A DTMC is discrete time probabilistic state space model that satisfies the following:*

- The state space $S$ is countable
- For all states i,j there is a given probability $p_{ij}$ such that

  $$Pr[X_{n+1} = j | X_n = s_n, X_{n-1} = s_{n-1}, ..., X_0 = s_0] = p_{ij}$$

  for all $s_0, s_1, ..., s_n \in S$ and $n \geqslant 0$
- $p_{ij} >= 0$, for all $i, j \in S$ and
  $\sum_{j \in S} p_{ij} = 1, \forall i \geqslant 0$
- $s_0$ = initial state, and $s_n$ = final state

### 3.2.1 Modelling statements to DTMC

The system specification is a set of statements that looks like a high level programming language[1]. The DTMC model representation of different statements given in [1] and breifly desribed here.

**Expression Statement**

A basic statment of the form "$x = E < r >$" models the execution of an individual component $E$ in a system with reliability $r$ and the result being stored in the variable $x$. The DTMC representation of a basic statement x = 0.9 is provided in Fig 3.1(a)

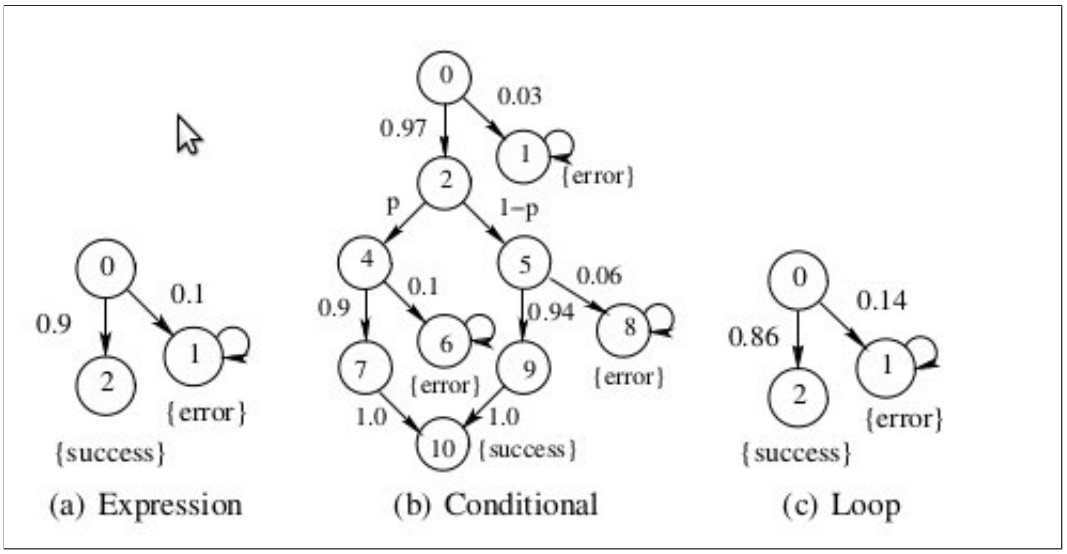


Figure 3.1: DTMC representation of the statements

**Conditional Statement**

A conditional statment of the form *"if E<r> then ST1 else ST2"* models the execution of an conditional execution individual components in a system based on the truth value of the expression $E < r >$. The DTMC of a conditional statement with r = 0.97 and reliability values of individual statements ST1 and ST2 being 0.9 and 0.94 respectively is provided in Fig 3.1(b).

**Loop Statement**

A loop statment of the form *"while* $E < r_1 >$ *do ST1"* models the execution of an conditional execution individual components in a system based on the truth value of the expression $E < r_1 >$. The DTMC of a loop statement with r1 = 0.97 and reliability values of individual statement ST1 being 0.98 is provided in Fig 3.1(c).

The paper [1] describes the DTMC representation of other statements as well.

### 3.2.2 Modelling System Specification to DTMC

Now, we can model the whole system specification in to equivalent DTMC by using the above constructs. The DTMC generated always have one unique start start 0 and one unique final state $f$. The figure 3.2 represents a DTMC of the below given example program:

```
void Example()
{
    int i, x, alt;
    [1] i = 7 <0.90>;
    [2] x = 1 <0.91>;
    [3] alt = 10 <0.96>;
    [4, 0.56] if (i < 10) <0.99> {
        [5] alt = 1 <0.93>;
    }
    else{
        [6] x = 1 <0.97>;
    }
}
```

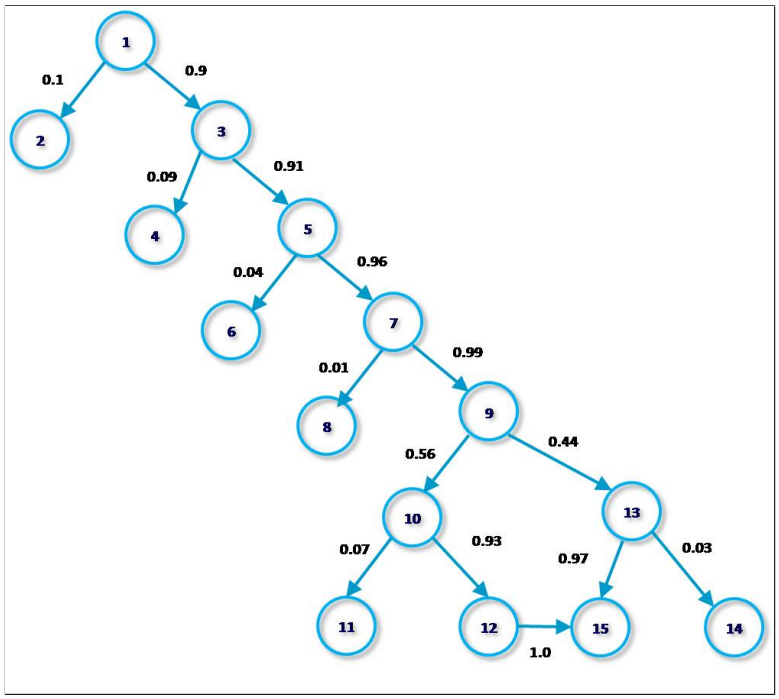


Figure 3.2: Graph representation of software model

Each statement in the software specification represents a component and reliability of the component is assoicated number specified in the chevron symbols < and >. Each statement (function call) is a component (subsystem) of the system. The values between the '[' and ']' specifies the component number for simple statements. For branching statements, the other number specifies the reliability of the branching statement. The software specification given is parsed and a DTMC representation of the specification is generated. It basically represents an adjacency matrix in which every edge is associated with one value representing the probability of success and failure.

## 3.3 System Reliability Calculation Using DTMC Representation

The execution of the system represents the execuation of all the paths of DTMC that reach to the final state of the DTMC from the initial state. The path of the DTMC that starts with the initial state and ends with the final state is called as success path. Each success path of DTMC can be viewed as series connection of components along that path. The DTMC path is assumed as success path of DTMC in this document. The following definitions are used for calculation of the system reliability.

**Definition 2.** ***DTMC Path:*** *A DTMC path* $\pi$ *is a sequenece of states* $\pi = s_0....s_n$*, where* $s_i \in S$ *such that* $Pr(s_i, s_{i+1}) > 0$*,* $\forall i \geqslant 0$*.* $s_0$ *is the initial state and* $s_n$ *is the final state..*

**Definition 3.** ***Path Index:*** *A path index* $PI(\pi)$ *of a DTMC path* $\pi$ *defined as* $PI(\pi) = \prod Pr(s_i, s_{i+1}) > 0$*, where transition from* $s_i$ *to* $s_{i+1}$ *represents a conditional true/false measure,* $\forall i \geqslant 0$*. If transition from* $s_i$ *to* $s_{i+1}$ *does not represent a conditional true/false measure, then the tranisition probability* $Pr(s_i, s_{i+1})$ *taken as 1.*

**Definition 4.** ***Path Probability:*** *A path probability* $Pr(\pi)$ *of a DTMC path* $\pi$ *defined as* $Pr(\pi) = \prod Pr(s_i, s_{i+1}) > 0$ *such that* $Pr(s_i, s_{i+1}) \geqslant 0 \; \forall i \geqslant 0$*.*

So, the system reliability can be calculated using the DTMC as follows:

- Find all the paths from initial node(state) to final node(state)
- Find the reliability of each of the path
- Sum the reliability of each of the path

Sum of the reliability of each of the path of DTMC represents the system reliability. Now, we arrived at calculation of system reliability using DTMC in which each of the component associated with single reliabilty option. We extend the same to the system in which each component is associated with a set of reliability options.

## 3.4 Optimization Of System Reliability As MINLP

The optimization of a system in which each component is associated with a set of reliability and cost options can be modelled as a MINLP problem as follows:

## 3.5 Problem Formulation

Let there be n subsystems (components) $SS_i$, i = 1,..,n, each with reliability $R_j$ and cost $C_j$, j = 1,...,m options. Let $C_S$ and $R_T$ represent the total system cost and the target reliability. If all the subsystems are essential to the system and if their failures are statistically independent, the system can be modeled as a series system. The reliability maximization problem can be stated as:

$n$ – Number of Components
$m$ – Number of implementation options for each component
$C_{i1}$, $C_{i2}$, ..., $C_{im}$ – Cost options of component $i$
$R_{i1}$, $R_{i2}$, ..., $R_{im}$ – Reliability options of component $i$

Now the formula is:

$$\text{minimize } C_T = \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} x_{ij}$$

subject to

$$R_T \leqslant \prod_{i=1}^{n} \sum_{j=1}^{m} R_{ij} x_{ij}$$

$$\sum x_{i1} + x_{i2} + \ldots + x_{im} = 1 \text{ for all } i = 1,..,n$$

$$x_{ij} = 0 \text{ } or \text{ } 1 \text{ for all } i = 1,..,n \text{ and for all } j = 1,..,m$$

The goal is to maximize $R_S$, the system reliability while minimizing the $C_T$, system cost.

In the given specification above, each statement is modelled as a new component. But in general, a statement may be occured multiple times as if a function can be called from different places in the software.. So a component that represent a function call may repeat in the specification. In the DTMC specified in fig 3.2, each node has a unique number. So, same component may represent multiple nodes in the DTMC. Therefore, a mapping from node to component shall be done. If there are 'c' components and n nodes such that c <= n, these nodes must be mapped to the components and there reliability options shall be considered in the calculation of the system reliability.

**Definition 5.** ***Node-Componet mapping function:*** *A node to component mapping function 'f' is defined $f(n) \rightarrow i$. This function takes a node number as input and returns the corresponding component number.*

Assume there are $p$ paths, and each path is having at most $n$ nodes, the reliability is calculated as:

$$R_T \leqslant \sum_{r=1}^{p} PI(\pi_r) \prod_{z \in \pi_r, f(z) \to i} \sum_{j=1}^{m} R_{ij} x_{ij}$$

where

- $PI(\pi_r)$ – the path index of 'r'th path
- $f(z)$ – the node to component mapping function

The value $x_{ij}$ is either '0' or '1' that represents whether the component 'i' is there on that path or not. An ILP solver can be used to solve the MINLP problem using above formulation. The output of the solver is the configuration of the system that provides the desired system reliability.

**Definition 6.** ***Configuration of the System:*** *A sequence of selected 'n' reliability options for every component in the system. It is denoted by a vector,* $\Gamma$ *of dimension n where n is the total number of components in the system,* $1 \leqslant i \leqslant n$.

**Definition 7.** ***Best Configuration of the System:*** *A configuration of system* $\Gamma$ *that provides the maximum reliability at minimum cost*

## 3.6 Optimization Sequence

The work-flow of the proposed system that reads the speicification file and generates the best configuration of the systeym is shown in fig 3.3. The input system specification with reliability constructs is parsed and partial DTMC model is developed as specified in the introduction. Then component and node mapping is done. The cost and reliability options are read and completed DTMC model is developed. All the paths from node 1 to node n can be found by using DTMC and reliability along these paths is calculated. Then the problem is converted into ILP formulation according to the specification of the solvers (SCIP, BONMIN, and Matlab GA) used. The ILP specification in the format of Matlab are executed in Matlab interface and results are documented.

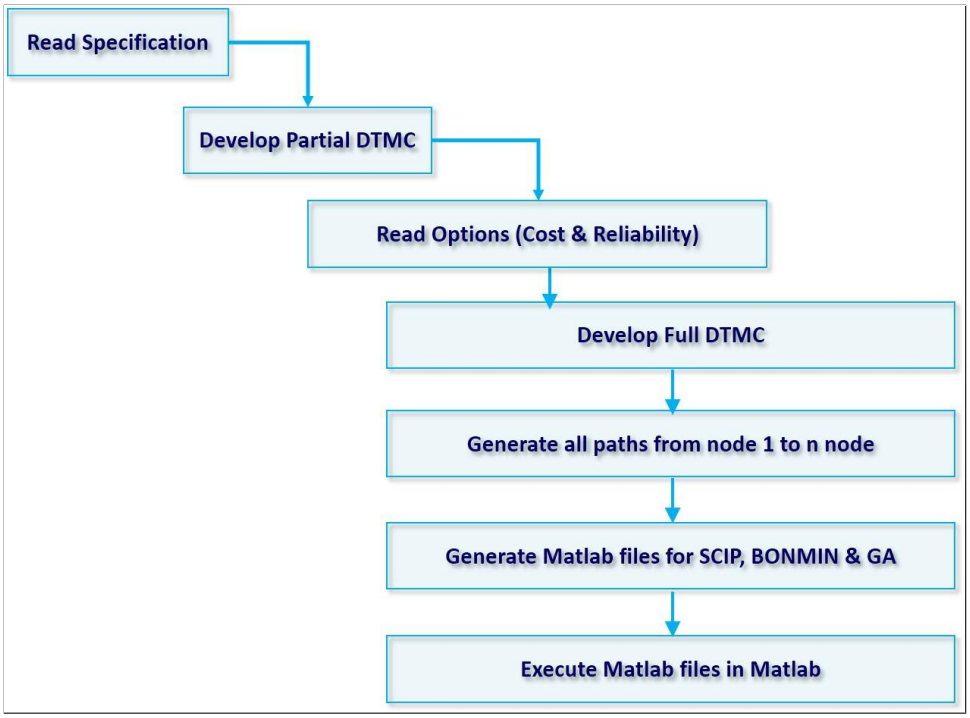


Figure 3.3: Optimization Methodology

## 3.7 Various Approaches

The three Mixed Integer Non-Linear Problem solvers used are:

1. BONMIN
2. SCIP
3. Matlab GA

Matlab provides default algorithm called Genetic algorithm to solve MINLP problems. Other two methods are supported by OPTI toolbox with its academic version. Each of the approach takes the input in a specific format. The optimization software has to generate the Matlab files in specific formats for each of the solver specified above. These files are to be executed in the Matlab interface to get the optimized values if possible.

The OPTI tool box of academic version has to be downloaded from internet and has to be integrated with the Matlab software. OPTimization Interface (OPTI) Toolbox is a free MATLAB toolbox for constructing and solving linear, nonlinear, continuous and discrete optimization problems for Windows users. The graphical view of the ILP solvers is given below:

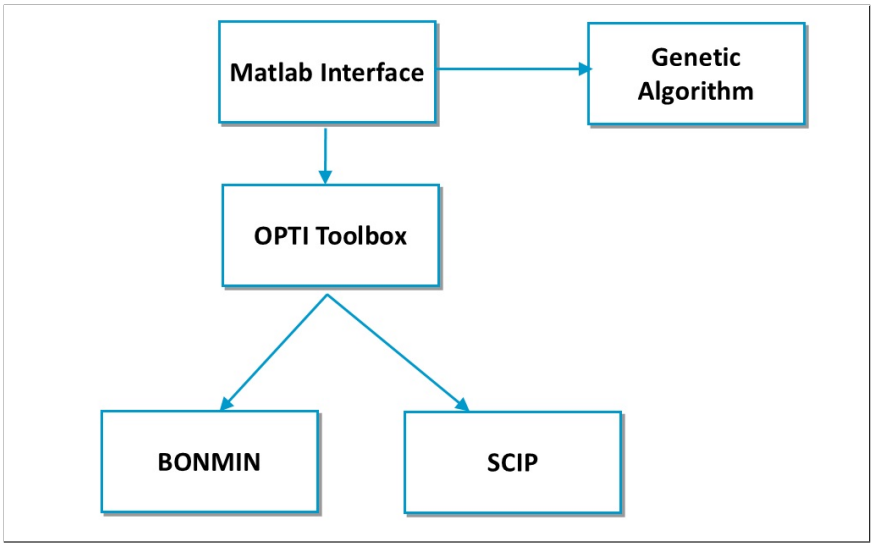


Figure 3.4: Optimization Approaches

### 3.7.1 BONMIN (Basic Open-source Nonlinear Mixed INteger programming)

BONMIN (Basic Open-source Nonlinear Mixed INteger programming) is an open-source code for solving general MINLP (Mixed Integer Nonlinear Programming) problems. It is distributed on COIN-OR under the CPL (Common Public License). BONMIN is OSI Certified Open Source Software.

BONMIN implements six different algorithms for solving MINLPs:

- B-BB: a simple branch-and-bound algorithm based on solving a continuous nonlinear program at each node of the search tree and branching on variables
- B-OA: an outer-approximation based decomposition algorithm
- B-QG: an outer-approximation based branch-and-cut algorithm
- B-Hyb: a hybrid outer-approximation/nonlinear programming based branch-and-cut algorithm
- B-Ecp: another outer-approximation based branch-and-cut
- B-iFP: an iterated feasibility pump algorithm

### 3.7.2 SCIP (Solving Constraint Integer Programs)

SCIP is currently one of the fastest non-commercial solvers for mixed integer programming (MIP) and mixed integer nonlinear programming (MINLP). It is also a framework for constraint integer programming and branch-cut-and-price. It allows for total control of the solution process and the access of detailed information down to the guts of the solver. SCIP is a framework for Constraint Integer

Programming oriented towards the needs of Mathematical Programming experts who want to have total control of the solution process and access detailed information down to the guts of the solver. SCIP can also be used as a pure MIP solver or as a framework for branch-cut-and-price.

## 3.8 Implementation

### 3.8.1 Example Problem

The implementation of the problem is explained for the given example above. The software specification parsed and a graph representation of the specification is done. It basically represents an adjacency matrix in which every edge is association with one value representing the probability of success and failure. The adjacency matrix is read using an initialization file "AdjMatrix.ini" and the component cost and reliability options are read using another intialization file "ComponentInFile.ini". A sample of "AdjMatrix.ini" is given below that contains the adjacency matrix for the above model It contains 15 nodes and 15 edges:

| Edge | Reliability |
|---|---|
| 1 -> 2 | 0.10 |
| 1 -> 3 | 0.90 |
| 3 -> 4 | 0.09 |
| 3 -> 5 | 0.91 |
| 5 -> 6 | 0.06 |
| 5 -> 7 | 0.94 |
| 7 -> 8 | 0.01 |
| 7 -> 9 | 0.99 |
| 9 -> 10 | 0.56 |
| 9 -> 13 | 0.44 |
| 10 -> 11 | 0.07 |
| 10 -> 12 | 0.93 |
| 12 -> 15 | 1.00 |
| 13 -> 14 | 0.03 |
| 13 -> 15 | 0.97 |

Table 3.1: Adjacency Matrix

Column 1 represent node numbers that connect the edge in the DTMC and column 2 represent probability value.A samle of "ComponentInFile.ini" file that contains the component reliability information is given as:

| Component Num | Num Of Options | Reliablity Cost Pairs |
|---|---|---|
| 1 | 4 | <0.98 10.00> <0.985 20.00> <0.99 40.00> <0.995 65.00> |
| 2 | 5 | <0.97 6> <0.98 12> <0.99 20> <0.993 30> <0.995 45> |
| 3 | 3 | <0.99 25.00> <0.991 40.00> <0.992 60.00> |
| 4 | 2 | <0.99 40.00> <0.995 60.00> |
| 5 | 2 | <0.995 70.00> <0.998 80.00> |
| 6 | 2 | <0.997 40.00> <0.998 50.00> |

Table 3.2: Component Information

This file also contains the start node and end node information. The adjacency matrix and the component information comprise the input to the optimization software. The values between the '<' and '>' specifies the reliability of the component and cost associated with that component. The Optimization software shall read the above information and find all the paths between the start node and end node. The output of the optimization software shall be the ILP formulation of the specification in specific format that is suitable to each of the algorithm mentioned above. ILP formulation for each of the solver shall be done separately and shall be saved in the respective matlab files.

Once the ILP formulation is done, these files shall be executed in the Matlab interface and results shall be noted. The following sections describe the each of the approach for optimizing the ILP formulation of the specification.

A sample matlab program for the specified problem above is provided below. The text in figure 3.5 and 3.6 is to be entered in MATLAB sequentially. The figure 3.5 defines the objective function, linear and nonlinear equations, the bounds and integer variables. The figure 3.6 calls the respective solver in MATLAB.

```
%objective function & constraints
fun = @(x) ...
        10 * x(1) +     20 * x(2) +     40 * x(3) +     65 * x(4) + ...
         6 * x(5) +     12 * x(6) +     20 * x(7) +     30 * x(8) +     45 * x(9) + ...
        25 * x(10) +    40 * x(11) +    60 * x(12) + ...
        40 * x(13) +    60 * x(14) + ...
        70 * x(15) +    80 * x(16) + ...
        40 * x(17) +    50 * x(18);

%Linear Constraints
A = [    1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 ;
        -1 -1 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0 ;
         0  0  0  0  1  1  1  1  1  0  0  0  0  0  0  0  0  0 ;
         0  0  0  0 -1 -1 -1 -1 -1  0  0  0  0  0  0  0  0  0 ;
         0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0 ;
         0  0  0  0  0  0  0  0  0 -1 -1 -1  0  0  0  0  0  0 ;
         0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0 ;
         0  0  0  0  0  0  0  0  0  0  0  0 -1 -1  0  0  0  0 ;
         0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0 ;
         0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1  0  0 ;
         0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1 ;
         0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1 ;
        ];

b =  [1; -1; 1; -1; 1; -1; 1; -1; 1; -1; 1; -1; ];

% Nonlinear Constraint
nlcon = @(x) ( 0.95 - ...
                ( ( 0.980 * x(1) + 0.985 * x(2) + 0.990 * x(3) + 0.995 * x(4) )  * ...
                ( 0.970 * x(5) + 0.980 * x(6) + 0.990 * x(7) + 0.993 * x(8) + 0.995 * x(9) )  * ...
                ( 0.990 * x(10) + 0.991 * x(11) + 0.992 * x(12) )  * ...
                ( 0.990 * x(13) + 0.995 * x(14) )  * ...
                ( 0.440000 )  * ...
                ( 0.997 * x(17) + 0.998 * x(18) )  + ...
                ( 0.980 * x(1) + 0.985 * x(2) + 0.990 * x(3) + 0.995 * x(4) )  * ...
                ( 0.970 * x(5) + 0.980 * x(6) + 0.990 * x(7) + 0.993 * x(8) + 0.995 * x(9) )  * ...
                ( 0.990 * x(10) + 0.991 * x(11) + 0.992 * x(12) )  * ...
                ( 0.990 * x(13) + 0.995 * x(14) )  * ...
                ( 0.560000 )  * ...
                ( 0.995 * x(15) + 0.998 * x(16) ) )  );

nlrhs = 0; nle = -1;

%bounds
lb =  [0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ];
ub =  [1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ];

%Integer constraints & Initial Guess
xtype =  'BBBBBBBBBBBBBBBBBB';
x0 =  [0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; ];
```

Figure 3.5: MATLAB program for BONMIN Solver

To solve the problem:

```
% Create OPTI Object & Solve the MINLP problem
opts = optiset('solver','bonmin','display','iter');
Opt = opti('fun',fun,'nlmix',nlcon,nlrhs,nle,'ineq',A,b,'bounds',lb,ub,'xtype',xtype, 'options', opts)
```

Figure 3.6: Calling the Solver in MATLAB

## 3.9 Optimization Using BONMIN Algorithm

We have to execute the Matlab file that represents the ILP formulation of the problem. The Matlab file generated by the optimization software is `optibonmin_relcal.m`. The execution of the file in Matlab interface is done by entering the filename without extension as command in interface. The output generated by the algorithm is shown below for desired reliability of 0.975:

```
x =  0     0     0     1     0     0     0     0     1     0     0     1     0     1     0     1     0     1

fval =    360

exitflag =   1

info =

        Iterations: 172
        Nodes: 0
        AbsGap: 5.5806
        RelGap: 0.0155
        Time: 1.9238
        Algorithm: 'BONMIN: Branch & Bound using IPOPT & CBC'
        Status: 'Success'
```

Figure 3.7: Output of BONMIN Solver

The $fval$ represents the cost of the system returned by the solver. The value changes according to the desired reliability. The variable *Time* represents the time taken by the solver to get the result. The *status* field specifies whehter the solver is able to solve the problem or not.

## 3.10 Optimization Using SCIP Algorithm

We have to execute the Matlab file for the SCIP slover that represents the ILP formulation of the problem. The Matlab file generated by the optimization software is "`optiscip_relcal.m`". The filename without extention of *m* is the command to the matlab. The execution of the file in Matlab interface as command starts the respective solver to solve the problem. The output generated by the algorithm is shown below figure for desired reliability of 0.975. The output of the solver can be saved in to a file also.

```
x =  0     0     0     1     0     0     0     0     1     0     0     1     0     1     0     1     0     1

fval =    360

exitflag =   1

info =

         BBNodes: 33
         BBGap: 0
         Time: 0.2784
         Algorithm: 'SCIP: Spatial Branch and Bound using IPOPT and SoPlex'
        | Status: 'Globally Integer Optimal'
```

Figure 3.8: Output of SCIP Solver

The *fval* represents the cost of the system returned by the solver. The value changes according to the desired reliability. The variable *Time* represents the time taken by the solver to get the result. The *status* field specifies whehter the solver is able to solve the problem or not.

## 3.11 Optimization Using GA Algorithm

We have to execute the Matlab file that represents the ILP formulation of the problem. The Matlab files generated by the optimization software for Genetic algorithm are `ga_objfun.m` and "`ga_constraints.m`". We need to use a tool called optimtool to execute these files.

The procedure for execution of Matlab files for Genetic algorithm is given below:

1. Open matlab;
2. Type optimtool in the command interface;
3. It opens a window as shown below.

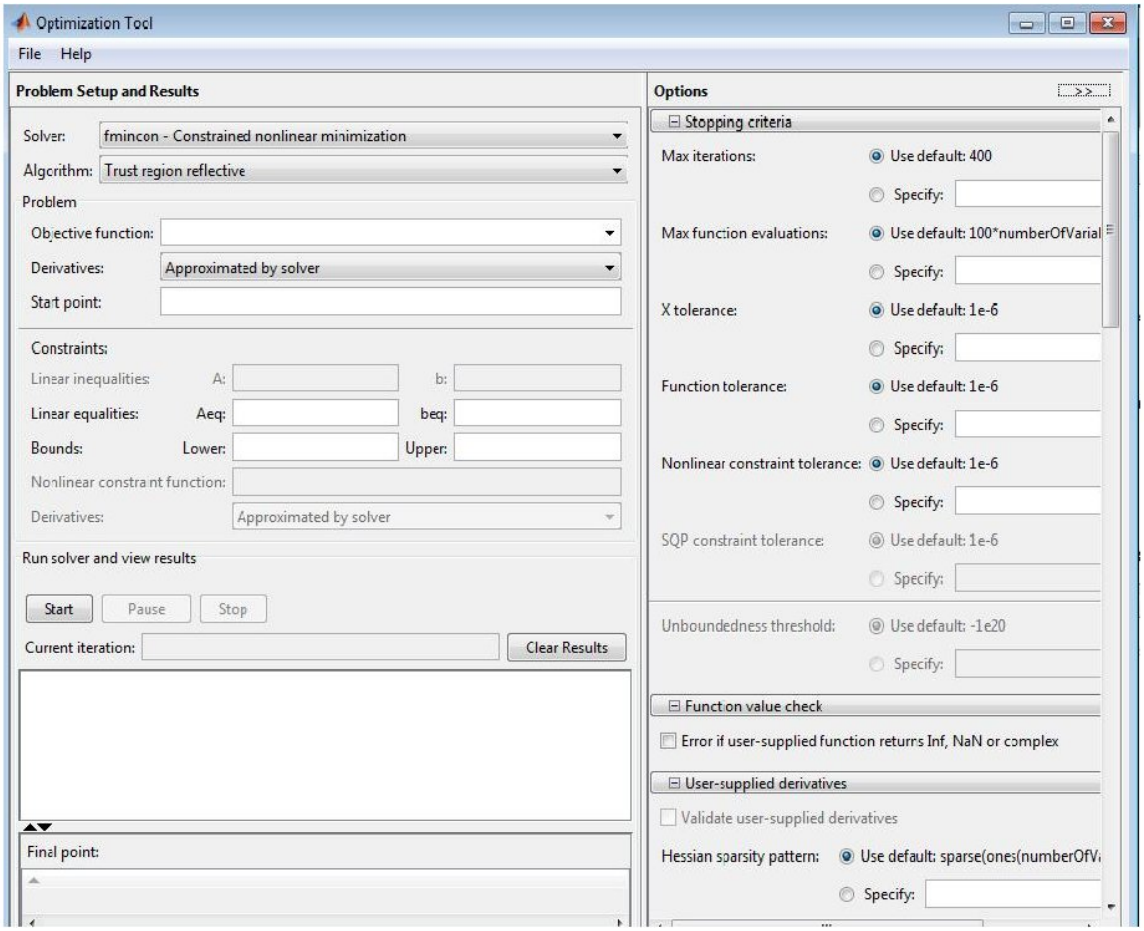


Figure 3.9: Matlab GA Solver Window

4. Select the following options in the optimtool window:

   Solver : ga
   Number of Variables : 18
   Fitness Function : `@ga_objfun`
   LowerBound : [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0]
   UpperBound : [1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1]
   Nonlinear constrain function : `@ga_constraints`
   Integer variable indices : [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18]

5. Run the solver

The output generated by the algorithm is shown below for desired reliability of 0.955:

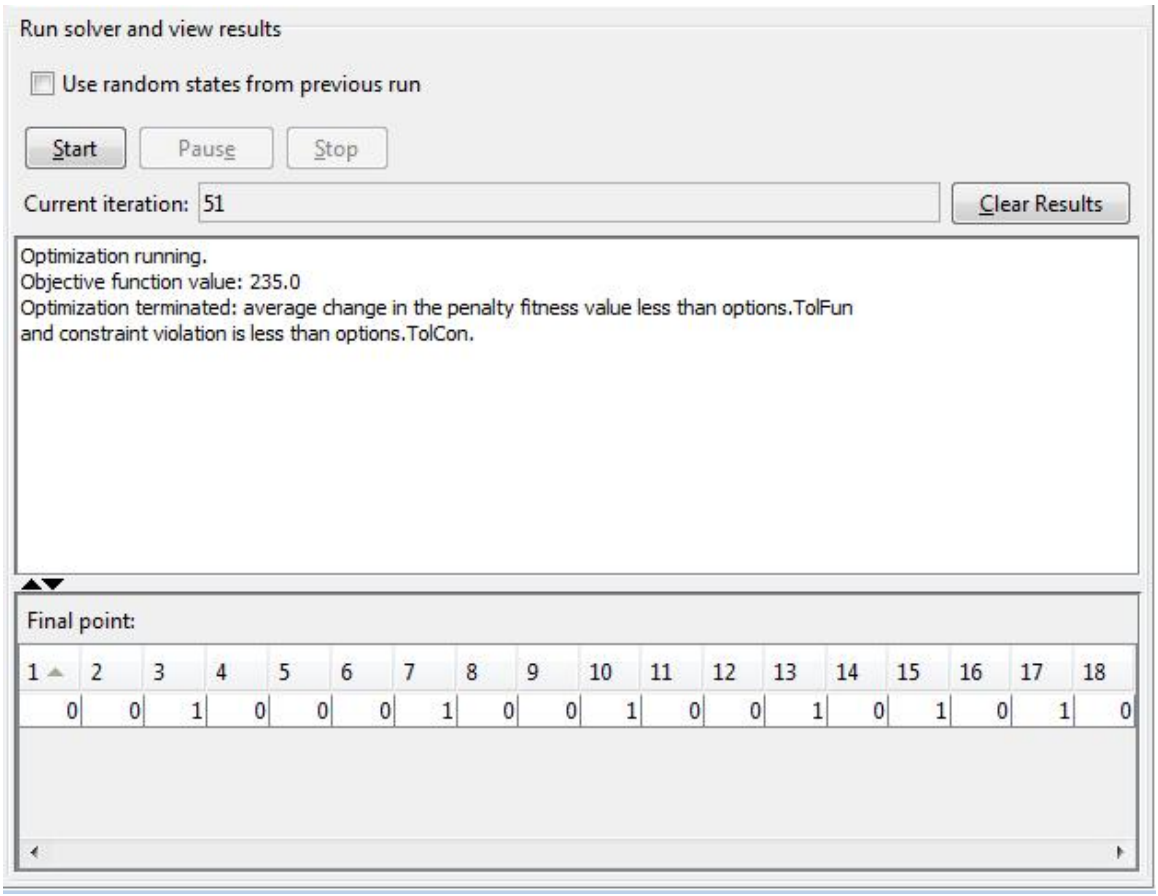


Figure 3.10: Output of Matlab GA Solver

## 3.12 Comparison Of Results

Three MINLP solvers are used for the reliability optimization. The comparison of results shows that SCIP framework outperformed the other two MINLP solvers. The comparision of the results are provided below:

| Solver Name | Desired Reliablity | | | | |
|---|---|---|---|---|---|
| | 0.95 | 0.96 | 0.965 | 0.97 | 0.975 |
| BONMIN | 215/26/0.5 | 255/733/2.98 | 275/909/3.05 | 300/230/2.08 | 360/172/1.9 |
| SCIP | 215/218/0.25 | 255/293/0.26 | 275/312/0.2 | 300/300/0.26 | 360/284/0.35 |

Table 3.3: Result of SCIP and BONMIN

The values 0.1 to 0.5 represents the desired reliability values. Three numbers in the column specifies cost incurred, number iterations respective solver took and the time taken by the respective solver.

The GA algorithm produces different output each time we run the algorithm. The output may not be a best solution. The series of solutions given by GA algorithm for the specified desired reliabilities are given below:

| Desired Reliability | Sequence Of Optimal Values By GA |
|---|---|
| 0.95 | 191, 211, 211, 191, 201, 201, 201, 211, 211, 216 |
| 0.96 | 230, 225, 225, 222, 220, 227, 215, 232, 215, 235 |
| 0.965 | 255, 255, 265, 250, 250, 275, 250, 270, 255, 255 |
| 0.97 | 300, 275, 295, 305, 305, 300, 295, 300, 310, 300 |
| 0.975 | 300, 335, 335, 295, 325, 340, 340, 295, 325, 350 |

Table 3.4: Result of GA

The SCIP algorithm is one of fastest open source MINLP solver. SCIP ensures global optimal solutions for convex and non-convex MINLPs. It implements a spatial branch and bound algorithm that utilizes LPs for the bounding step.

SCIP is implemented as C callable library and provides C++ wrapper classes for user plug-ins. It has its own dynamic memory management to perform better. SCIP uses SoPlex as LP solver, the COIN-OR Interior Point Optimizer (IPOPT) as nonlinear solver, and CppAD to compute derivatives of nonlinear functions. SCIP supports continuous, binary, integer, semi-continuous, and branching priorities for discrete variables.

SoPlex is a Linear Programming (LP) solver based on the revised simplex algorithm. It features preprocessing techniques, exploits sparsity, and offers primal and dual solving routines. SoPlex is completely implemented in C++ using object oriented concepts.

IPOPT (Interior Point OPTimizer) is a software library for large-scale nonlinear optimization. IPOPT implements an interior-point algorithm for continuous, nonlinear, non-convex, constrained optimization problems. It is meant to be a general purpose nonlinear programming (NLP) solver. However, it is mainly written for large-scale problems with up to millions of variables and constraints (For such large problems, it is assumed that the derivative matrices are sparse).

SCIP follows the plug-in based architecture. SCIP distinguishes itself through the modular design of its solution algorithm, which allows the combination of general-purpose and specialized solving

techniques in a flexible fashion. The central objects of SCIP are constraint handlers. A constraint handler must be able to decide whether a given solution is feasible for all constraints of its type. To speed up the solution process, it may provide supplementary algorithms like constraint-specific pre-solving, domain propagation, separation methods, or a linear representation of its constraints. Additional plug-ins such as branching rules, propagators, and primal heuristics allow for an efficient solution process. Altogether, SCIP 3.0 contains 126 plug-ins. The interaction among them is organized by the core of SCIP.

The BONMIN algorithm uses branch and bound method whereas SCIP uses spatial branch and bound method. The algorithms in BONMIN are exact when the problem is convex, otherwise they are heuristics. BONMIN uses IPOPT (Interior Point Optimizer) for solving relaxed problems and CBC (COIN-OR branch and cut) as the mixed integer solver. SCIP uses IPOPT for solving relaxed problems and SoPlex as the LP solver.

The performance parameters for the GA algorithm are not provided by MATLAB. But, it is inclined to produce sub optimal solutions, it cannot be compared with the above two solvers as they are global optimization solvers. The commercial solvers like Gurobi and Baron outperform open source solvers as they includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor.

## 3.13 Conclusion

The BONMIN and SCIP algorithms performance is almost similar for the given example. It may vary as the number of components and nodes increases. Genetic Algorithm gives suboptimal solutions in every execution of the problem. The sequence of 10 executions is shown in the comparison of results for Genetic Algorithm.

# Chapter 4

# Optimization of Software Reliability Using Component Ranking & Cost Rate Function Approach

---

A system is generally designed as a set of interconnected subsystems, each with its own reliability and associated cost attributes. The overall system reliability is a function of the subsystem reliability metrics. The cost of the system is the sum of the costs for all the subsystems. The objective here is either to achieve the target reliability while minimizing the total cost, or maximize the reliability while using only the specified cost.

An exhaustive search of all reliability options for the subsystems to optimize the system reliability under the specified cost may not be feasible at all the times. Many systems are implemented by assembling a set of subsystems. While the architecture of the overall system can often be fixed, individual subsystems may be implemented differently. Intuitively, some of the components that contribute more to overall system reliability may need special attention to raise the overall system reliability. Using this point, here we discuss an approximate near solution to the global solution of the optimization problem by employing following approaches:

1. **Ranking the components**
2. **Use of Cost Rate Function**

These are the greedy methods and may not give global solutions always. This chapter explains an approach to allocate the reliability values using component (subsystem) ranking and cost functions such that the total cost of the system is minimized. The problem involves expressing the system reliability in

terms of the subsystem reliability values and specifying the cost function for each subsystem which allows setting up an optimization problem.

## 4.1 Component Ranking Approach

We consider a small example of software specification for Anti Lock Braking System is given below.

### 4.1.1 Example Problem

**Algorithm 1** ABS() //Anti Lock Braking System

```
1:  [1] sense_wheel_speed();
2:  [2] sense_wheel_speed();
3:  [3] calculate(slip);
4:  if [4,0.5] (slip > slip_max) then
5:      [3] calculate(release_delay);
6:      [5] ABS_HOLD (release_delay);
7:      [3] calculate(release_time);
8:      [6] ABS_DECREASE (release_time);
9:  else if [7,0.5](slip > slip_min) then
10:     [3] calculate_secondary_apply_delay();
11:     [5] ABS_HOLD (secondary_apply_delay);
12:     [3] calculate_secondary_increase_time();
13:     [8] ABS_INCREASE (secondary_increase_time);
14: else
15:     [3] calculate(primary_apply_delay);
16:     [5] ABS_HOLD(primary_apply_delay);
17:     [3] calculate(primary_increase_time);
18:     [8] ABS_INCREASE (primary_increase_time);
19: end if
```

The Anti lock braking(ABS)[14] system prevents the wheels of the vehicle from locking up thereby stopping the vehicle from sliding and enables the driver to retain directional control. The ABS example given above modelled using 8 components. The values between the '[' and ']' specifies the component number. For branching statements, the other number specifies the reliability of the branching statement. The component reliabilities and associated costs are given as input. It is evident that some of the components are repeated (example: function calls) in the above program. We try to rank the component in order of contribution to the overall system reliability.

### 4.1.2 Ranking of Components

The Idea is we initially calculate the frequency of each component. Here, we specify the term “Path Reliability of the component". The path reliability is calculated as the reliability of the path between the starting node to the component node in consideration. Each component may appear in different paths from the start node to the end node in the DTMC. We assume reliability 1 for the simple statements and for the branching statements, specified reliabilities in the software specification are taken. We calculate the path reliability of each instance of a given component in all the paths it appears in the DTMC. The individual path reliability of each instance of the component is summed up and the result is the path reliability of that particular component. This process is repeated for every component in the system. For example, the component ‘3’ appears 7 times in the given example. The path reliability for each of the instance of the component ‘3’ is calculated and sum of all these reliabilities is considered as the path reliability of the component. The component that has highest path reliability has the highest rank. We sort the components according to their path reliability.

### 4.1.3 Optimization Algorithm Using Component Ranking

The sequence of steps in the algorithm for optimization is given below:

1. Perform the component ranking(using path reliability of the component)
2. Start from the basic configuration( initial reliability options )
3. Increment the option of the highest ranked component in circular fashion
4. Calculate the reliability for present configuration
5. Calculate the cost for present configuration
6. Repeat the steps 3 to 5 until the cost crosses the target cost

The pseudo code of the algorithm is presented here. The function rankComponents() calculates the path reliability of each component as explained above and sorts the components based on the path reliability values. The component that has the highest path reliability shall be given rank one, second highest as rank two and so on. The getNextRankedComponent() returns the selected next highest ranked component. If the reliability option of the present component is already the last option, then it returns the next highest ranked component. The changeCurrentConfiguration() function changes the current configuration by advancing the option of the current highest ranked component. Based on the present configuration, reliability of the system and cost of the system is calculated. The algorithm terminates when the current cost exceeds the target cost.

**Notation:**

$D$ – DTMC Model

$n$ – Number of Components

$m$ – Number of implementation options for each component

$V_C$ – Component vector

$V_C[i].cont$ – $i^{th}$ Component Contribution

$V_{1,n}$ – Node Vector, contains nodes corresponding to each component

$F_{1,n}$ – Feasibility Vector, contains feasibility values of each component for all reliability options

$O_C$ – Current Configuration

$O_B$ – Best Configuration

$C_{1,n}$ – Cost Vector, cost options

$R_{1,n}$ – Reliability Vector, reliability options

$C_T$ – Target Cost

$R_S$ – System Reliability

$C_S$ – Total cost of the system

**Algorithm 2** Ranking Components

**Input:**
The DTMC of system $D$, Component Vector $V_C$, Node Vector $V_{1,n}$
**Output:**
Ranked Component Vector $V_C$
**rankComponents()**
**{**
**for** $i = 1 \rightarrow n$ **do**
    rel $\leftarrow 0$;
    **for** $j = 1 \rightarrow count(V_n[i])$ **do**
        rel $\leftarrow$ rel + pathReliablity(1, $V_n[i, j]$)
    **end for**
    $V_c[i].cont \leftarrow$ rel;
**end for**
sort($V_c$); //based on contribution of each component
**}**

**Algorithm 3** Component Ranking Algorithm

**Input:**
The DTMC of system $D$, Current Configuration $O_C$ , Best Configuration $O_B$
Component Vector $V_C$, Node Vector $V_{1,n}$
Cost Vector $C_{1,n}$ , Reliability Vector $R_{1,n}$
Target Cost $C_T$
**Output:**
Best Configuration, $R_S$ , $C_S$
**Initialization:** $O_C \leftarrow O_B \leftarrow \{0\}$, $R_S \leftarrow C_S \leftarrow 0$
**optimizeWithRanking()**
**{**
rankComponents();
**while** true **do**
    component ← getNextRankedComponent();
    changeCurrentConfiguration(component, $O_C$ );
    $R_{cur} \leftarrow$ calculateReliability($O_C$);
    $C_{cur} \leftarrow$ calculateCost($O_C$)
    **if** $C_S \leqslant C_T$ **then**
        $O_B \leftarrow O_C$ ;
        $R_S \leftarrow R_{cur}$ , $C_S \leftarrow C_{cur}$ ;
    **else**
        return;
    **end if**
**end while**
**}**

## 4.2 Cost Rate Function Approach

There can be several choices for many of the subsystems providing the same functionality, but differently reliability levels. There is always a cost associated with reliability level because of changing a design due to change of vendors, use of higher-quality materials, administrative fees, etc. Developing the "cost of reliability" relationship will enable the user an understanding of which components to improve. The first step is to obtain a relationship between the cost of improvement and reliability.

The approach would be to formulate the cost function from actual cost data. The general (default) behavior model of the cost versus the component's reliability was developed for performing reliability

optimization. The proposed idea is cost rate function. The objective of this function is to model an overall cost behavior for all types of components.

### 4.2.1 Quantifying the Cost Rate Function

The cost rate function for each component in terms of the reliability, $R_i$ and Cost, $C_i$, of each component is specified as:

$$CostRateFunction = f(R_i, C_i)$$

This function should:

- Look at the current reliability of the component,
- Look at the previous reliability of the component, .

The cost rate function $f$ would satisfy these three conditions:

1. $f$ is a positive definite function
2. $f$ is non-increasing
3. $f$ decreases at a higher rate for higher values of $R_i$ and $C_i$.

The cost rate function is defined as:

$$f_{ij} = \frac{R_{ij} - R_{i-1\,j-1}}{C_{ij} - C_{i-1\,j-1}}$$

where:

- $f_{ij}$ is the cost rate function as a function of component reliability and cost.
- $R_{ij}$ and $R_{i-1\,j-1}$ are the current and previous reliability values of $i$th component and $C_{ij}$ and $C_{i-1\,j-1}$ are the current and previous cost values of $i$th component

### 4.2.2 The Feasibility Value, $f$

The feasibility term in above equation is a constant that represents the difficulty in increasing a component's reliability from the current reliability. Depending on the design complexity, technological limitations, etc., certain components can be very hard to improve. Clearly, the more difficult it is to improve the reliability of the component, the greater the cost.

### 4.2.3 Relaxation of the Feasibility

As some components repeat in the software system (function calls at different places), it is useful to select the component that has lower feasibility than the component that has the higher feasibility if that component is repeatedly occurs. For example, Assume component A occurs 5 times and component B occurs 2 times. If component A is having feasibility value 0.4 and component B is having feasibility value 0.5, selecting component A reliability option my contribute more to the overall system reliability. So, we relax the feasibility values according to the rank of the components that appears in the system. The feasibility values of a component are multiplied with its contribution. The resulted feasibility values are considered as the final feasibility values.

### 4.2.4 Optimization Algorithm Using Cost Function Approach

The sequence of steps in the algorithm for optimization is given below:

1. Perform the component ranking(using frequency)
2. Calculate the feasibility values for each of the reliability and cost options of all components
3. Relax the feasibility values
4. Start from the basic configuration ( all zero options )
5. Increment the option of a component that has highest feasibility value
6. Calculate the reliability and cost
7. Repeat the steps 5 and 6 until the cost crosses the target cost

The Cost Rate Function algorithm is presented below. The function rankComponents() calculates the path reliability of each component and ranks the components based on the path reliability values. Then the feasibility values for all of reliability options of each component are calculated. These are relaxed based on the path reliability values of the corresponding components. The getNextComponentWithHighestFeasibility() returns the selected next highest feasible component. This is done by increasiong the option of present configuration to next level option temporarily and comparing the feasibility values for this temporary configuration. The changeCurrentConfiguration() function changes the current configuration by advancing the option of the current selected component. Based on the present configuration, reliability of the system and cost of the system is calculated. The algorithm terminates when the current cost exceeds the target cost.

**Algorithm 4** CostRate Function Algorithm

**Input:**
The DTMC of system $D$, Current Configuration$O_C$ , Best Configuration $O_B$
Component Vector $V_C$, Node Vector $V_{1,n}$
Feasibility Vector $F_{1,n}$
Cost Vector $C_{1,n}$ , Reliability Vector $R_{1,n}$
Target Cost $C_T$
**Output:**
Best Configuration, $R_S$ , $C_S$
**Initialization:** $O_C \leftarrow O_B \leftarrow \{0\}$, $R_S \leftarrow C_S \leftarrow 0$
**optimizeWithCostFuntion()**
**{**
rankComponents();
**for** $i = 1 \rightarrow n$ **do** // calculates using cost rate function given above
    **for** $j = 1 \rightarrow m$ **do**
        $F_{1,n}[i,j] \leftarrow$ costRateFunction($C_{1,n}$, $R_{1,n}$, $i$, $j$);
        $F_{1,n}[i,j] \leftarrow F_{1,n}[i,j] * V_c[i]$.rel); //relax the feasibility value
    **end for**
**end for**
**while** true **do**
    component $\leftarrow$ getNextComponentWithHighestFeasibility();
    changeCurrentConfiguration(component, $O_C$ );
    $R_{cur} \leftarrow$ calculateReliability($O_C$);
    $C_{cur} \leftarrow$ calculateCost($O_C$)
    **if** $C_S \leqslant C_T$ **then**
        $O_B \leftarrow O_C$ ;
        $R_S \leftarrow R_{cur}$ , $C_S \leftarrow C_{cur}$ ;
    **else**
        return;
    **end if**
**end while**
**}**

## 4.3 Results & Discussion

The algorithms are implemented for the ABS program. The cost and reliability options are specified in the following tables and final results are shown in the graphs given below. There are 8 components and each component can have at the most 10 options. The option value zero specifies a invalid option.

| Component Num | Cost Options | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 10 | 20 | 35 | 55 | 80 | 110 | 145 | 185 | 235 | 290 |
| 2 | 6 | 11 | 18 | 28 | 42 | 60 | 86 | 120 | 160 | 220 |
| 3 | 5 | 13 | 25 | 40 | 60 | 85 | 115 | 150 | 0 | 0 |
| 4 | 7 | 15 | 30 | 55 | 65 | 90 | 130 | 180 | 240 | 310 |
| 5 | 15 | 30 | 50 | 90 | 150 | 230 | 0 | 0 | 0 | 0 |
| 6 | 20 | 50 | 90 | 150 | 230 | 310 | 0 | 0 | 0 | 0 |
| 7 | 15 | 25 | 40 | 65 | 100 | 150 | 0 | 0 | 0 | 0 |
| 8 | 10 | 25 | 50 | 85 | 130 | 200 | 0 | 0 | 0 | 0 |

Table 4.1: Cost Values

| Component | Reliability Options | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 0.9988 | 0.9989 | 0.999 | 0.9991 | 0.9992 | 0.9993 | 0.9994 | 0.9995 | 0.9996 | 0.9997 |
| 2 | 0.9990 | 0.9991 | 0.9992 | 0.9993 | 0.9994 | 0.9995 | 0.9996 | 0.9997 | 0.9998 | 0.9999 |
| 3 | 0.9985 | 0.9986 | 0.9987 | 0.9988 | 0.9989 | 0.999 | 0.9991 | 0.9992 | 0 | 0 |
| 4 | 0.9987 | 0.9988 | 0.9989 | 0.999 | 0.9991 | 0.9992 | 0.9993 | 0.9994 | 0.9995 | 0.9996 |
| 5 | 0.9984 | 0.9985 | 0.9986 | 0.9987 | 0.9988 | 0.9989 | 0 | 0 | 0 | 0 |
| 6 | 0.9981 | 0.9982 | 0.9983 | 0.9984 | 0.9985 | 0.9986 | 0 | 0 | 0 | 0 |
| 7 | 0.99915 | 0.99925 | 0.99935 | 0.99945 | 0.99955 | 0.99965 | 0 | 0 | 0 | 0 |
| 8 | 0.99945 | 0.99955 | 0.99965 | 0.99975 | 0.99985 | 0.99995 | 0 | 0 | 0 | 0 |

Table 4.2: Reliability Values

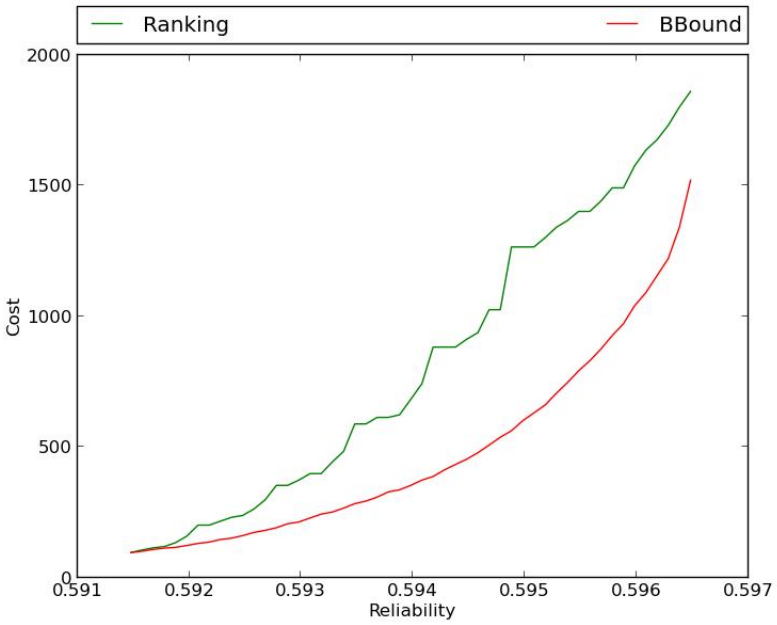


Figure 4.1: comparision of Component Ranking and Branch&Bound

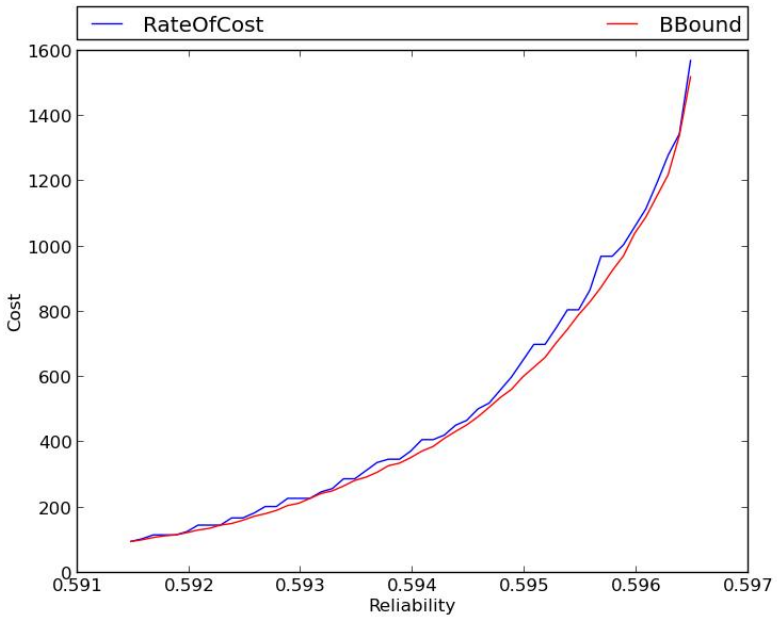


Figure 4.2: comparision of CostRate Function and Branch&Bound

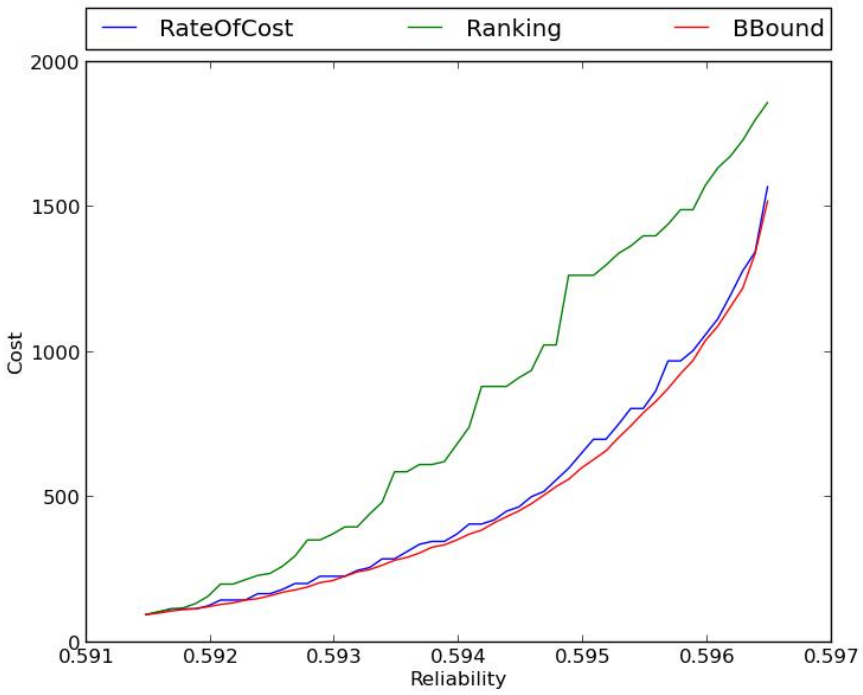


Figure 4.3: comparision of Three Algorithms

The branch and bound algorithm is exhaustive search algorithm which gives the global solution. The other two algorithms are compared with the performance of branch and bound algorithm. As we see from the figure, the cost function algorithm performs better initial part whereas the ranking algorithm performs better at end.

## 4.4 Conclusion

As we see from the results, the cost function approach and the component ranking algorithm are providing near optimal solutions at the higher target costs, the two algorithm performance is almost equal to the branch and bound algorithm. The best part of these two algorithms is the time they took for convergence of the solution.

# Chapter 5

# Optimization Based On Reliability Degradation Over Time

An underlying assumption in previous sections was, we considered the reliability of a component over a time period is constant. For example, the system reliability at time 500 hours will be same as the system reliability at time 1000 hours and so on. But in practical scenarios, as we know, the reliability of a component (either software or hardware component) decreases over the time. All equipment deteriorates with usage and progression in age. All hardware components are susceptible for the wear and tear and other characteristics which degrade the performance, reliability of those components. Whereas software components exhibit the degraded performance over a time period due to the exposure to the different types input data, underlying architecture etc. Consequently, the system comprised of different components exhibits the degraded performance as the time progresses. An example of a reliability of a system over time period with constant and degraded performance is shown in fig 5.1.

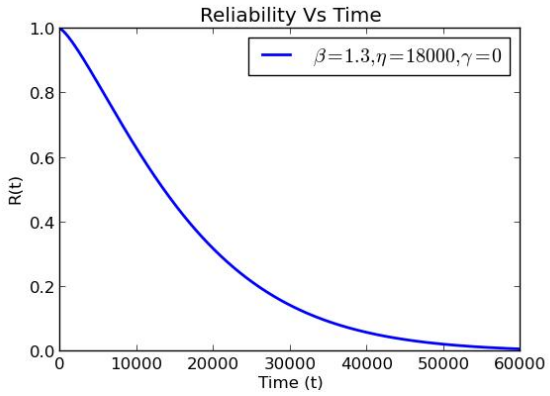


Figure 5.1: Reliability Curve

A component or subsystem or system reliability can be represented as a model. A Degradation model represents the behavior of a component in terms of performance or reliability over a period of time. There are different types of degradation models such as Exponential, Weibull, and Log-Normal distributions that are used to model component reliability. In this section we discuss a traditional degradation model called Weibull distribution, apply the model for each of the instance of the components of the subsystem and finally we perform optimization process to achieve the target reliability while minimizing the total cost of the system at different time interval slots. The final section of this chapter describes how the optimization process performed considering the replenishment of the present configuration of the system to achieve the specified target reliability at specific time for the given system.

## 5.1 Weibull Distribution

The performance or reliability of a product or component over a period of time defines the life time behavior of that product. Life data analysis of a component predicts the future performance of a certain component. Reliability Life Data Analysis refers to the study and modelling of components live under observation. The term "life data" means the measurements of component life. A component life can be measured using different metrics such as hours, miles, cycles etc. Life data refers to a measurement that specifies the period (time) of successful operation of a particular component. As our main criteria is of component reliability, here a life data point refers to "times-to-failure" or reliability of a component. For example, the component reliability at time zero is 1 whereas the same component reliability at time of 500 hours is 0.918 due to degraded performance. Each measurement at particular time represents a point in in the life data analysis. These points are used by statisticians, mathematicians and engineers to mathematically model the behavior of the component reliability to make future predictions about life of the component using a parameterized statistical distribution model. The parameterized distribution for the life data helps in estimating the important characteristics of the component such as reliability or probability of failure at a specific time, and the failure rate. So, the basic steps involved in life data analysis are:

1. Collect the life data of the component.
2. Select a statistical lifetime distribution that will fit the data for the component.
3. Estimate the parameters that will fit the distribution to the data.
4. Estimate the future data points or life time characteristics of the component by generating graphs.

The Weibull Distribution defines a probability density function (pdf) that represents the life data of a component in a mathematical model. A pdf function is a mathematical function that describes the distribution. The pdf can be represented mathematically or on a plot where the x-axis represents time. The Weibull distribution is widely used in reliability and life data analysis due to its versatility. The

3-parameter Weibull distribution given below represent life data analysis and is a popular distribution for analyzing the life data of a component. Weibull distribution is formulated by Professor Waloddi Weibull and given below:

$\eta$ = scale parameter, or characteristic life

$\beta$ = shape parameter (or slope)

$\gamma$ = location parameter (or failure free life)

$$f(t) = \frac{\beta}{\eta}(\frac{t-\gamma}{\eta})^{\beta-1} e^{-(\frac{t-\gamma}{\eta})^{\beta}}$$

where

$$f(t) \geqslant 0, t \geqslant or \gamma$$

$$\beta > 0, \eta > 0$$

$$-\infty < \gamma < +\infty$$

The Weibull model can be applied in a variety of forms (including 1-parameter, 2-parameter, 3-parameter or mixed Weibull) by fixing the one or more of parameters constant.

The scale parameter, $\eta$, defines where the bulk of the distribution lies. The shape parameter, $\beta$, defines the slope or shape of the distribution and the location parameter, $\gamma$, defines the location of the distribution in time. Estimation of these three parameters is performed to fit the given life data into the statistical distribution.

### 5.1.1 Weibull Reliability

Reliability is defined as the failure free operation of a component up to some specified time period. We assume that the component under consideration is non-repairable (it can be replaceable at the time it fails). The purpose of life data analysis of component reliability is to indicate the probability of success for a specified time. This probability is called as the component reliability, and is always associated with a given time. For example, a component specification may call for a 95% reliability at 500 hours of operation. This means that the component has a 95% probability of running for 500 hours without failure.

So, the reliability of a component is a function of time. It can be represented as continuous function as R(t) where R is the dependent variable reliability and t is the independent variable time. The reliability measure over the time is the fundamental measure in life data analysis of a component. The Weibull pdf can be used to derive reliability function of the component. The derivations are provided in [11]. The Weibull reliability function is given by:

$$R(t) = e^{-(\frac{t-\gamma}{\eta})^{\beta}}$$

where

$R(t)$ = reliability at time 't'

### 5.1.2 Characteristics of the Weibull Distribution

Selection of the three parameters in different ways result into different characteristics of the Weibull distribution. If the location parameter is made zero, the pdf equation reduces to that of the two-parameter Weibull distribution. If we fix the value of β also, the pdf equation reduces to that of the one-parameter Weibull distribution.

The Weibull distribution can be used to model a variety of life behaviors by using on the different values of the parameters. Some of the examples are given below that shows how the values of the scale parameter, η, the shape parameter, β, and affect such distribution characteristics as the shape of the pdf curve, the reliability.

#### Weibull Slope Parameter, β

The Weibull shape parameter, β, represents the slope of the reliability curve of a component. It is just a number. Different behaviors of the distribution can be specified by taking different values of the shape parameter. Some of the values of slope parameter can reduce the behavior of the distribution to those of other distributions. For example, when β = 1, the pdf of the three-parameter Weibull reduces to that of the two-parameter exponential distribution. The following figure shows the effect of different values of the shape parameter, β, on the shape of the pdf (while keeping γ constant).

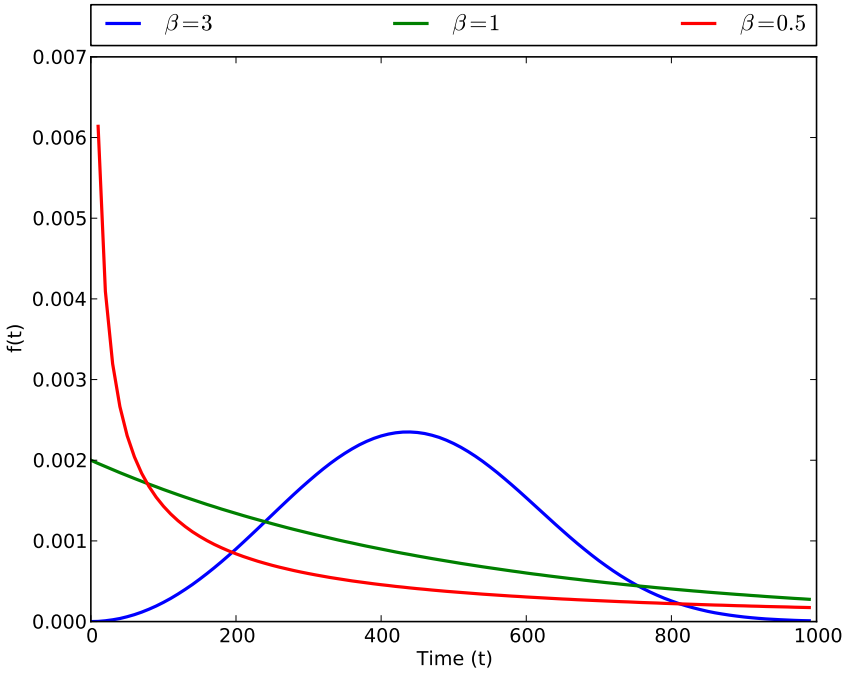


Figure 5.2: Slope parameter characteristics

**Weibull Scale Parameter, η**

The Weibull Scale parameter, β, represents the time of the reliability curve of a component. It is just a number. Increasing the value of the scale parameter while keeping the other two parameters constats is like stretching the probability distribution curve. While keeping the β and γ constant, increase of η value streches the curve to the right with decreasing height of the curve and decrease of η value streches the curve to the left with increasing height of the curve.

The following figure shows the effect of different values of the scale parameter, η, on the scale of the pdf (while keeping γ zero and β constant).

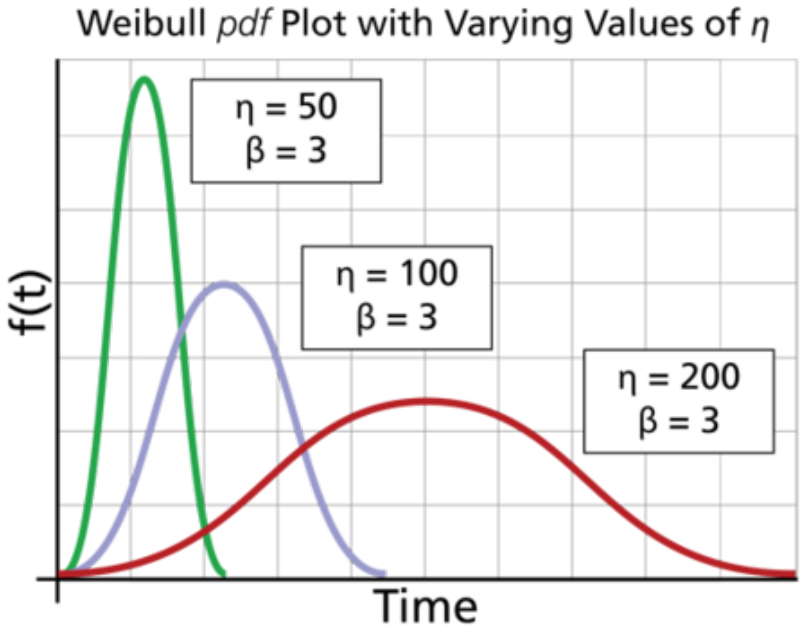


Figure 5.3: Scale parameter characteristics [12]

**Weibull Location Parameter, γ**

The Weibull shape parameter, γ, represents the location of the reliability curve of a component. It is just a number. The value of γ other than zero specifies the shift of the distribution from the origin. If the γ value is nagative, it shifts the distribution to the left of the origin and it shifts the distribution to the right of the origin on positive values of γ. The positive values of γ parameter specifies the failure free behavior over the period from zero to γ value.

The following figure shows the effect of different values of the shape parameter, β, on the shape of the pdf (while keeping γ constant).

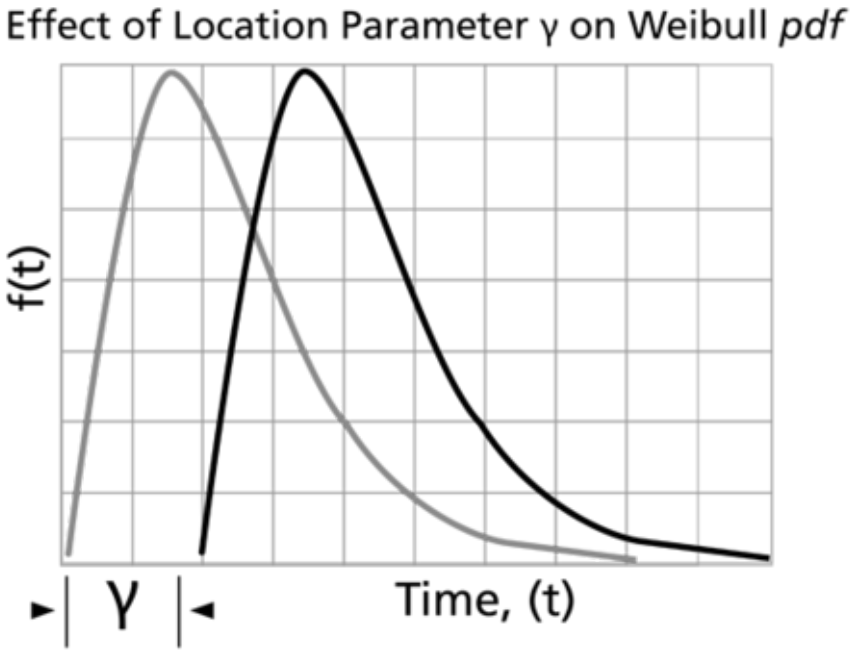


Figure 5.4: Location parameter characteristics [12]

### 5.1.3 Problem Definition and Formulation using Weibull Distribution

Now the problem definition changes. Let there be n subsystems (components) $SS_i$, i = 1,..,n, each with reliability $R_j$ and cost $C_j$, j = 1,...,m options. Let $C_S$ and $R_T$ represent the total system cost and the target reliability. Now the reliability values of the options of the components are specified by Weibull Distribution, these values degrades over a time period. So, if $j$ th reliability option of $i$ th component is 0.98 at time 100 hours, it may be 0.95 at 200 hours. So, Now the overall system reliability is a function of time. The objective is to get $R_T$, a target reliability of system at time $t_1$ or $R_T$, a target reliability of system at time $t_2$ etc. at the minimum cost. The problem formulation for a system reliability at time $t$ can be stated as:

**Notation:**

$n$ – Number of Components
$m$ – Number of implementation options for each component
$C_{i1}, C_{i1}, ..., C_{im}$ – Cost options of component $i$ at time $t = 0$
$R_{i1}, R_{i1}, ..., R_{im}$ – Reliability options of component $i$ at time $t$, which should be calculated using Weibull Reliability formula
$\beta_{i1}, \beta_{i1}, ..., \beta_{im}$ – $\beta$ options of component $i$ at time $t = 0$
$\gamma_{i1}, \gamma_{i1}, ..., \gamma_{im}$ – $\gamma$ options of component $i$ at time $t = 0$
$\eta_{i1}, \eta_{i1}, ..., \eta_{im}$ – $\eta$ options of component $i$ at time $t = 0$

$\pi_r$ – the path probability of 'r'th path,
$f(z) \rightarrow i$ – the mapping function from node to component

Now the formula is:

$$\text{minimize } C_T(t) = \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} x_{ij}$$

$$\text{subject to } R_T(t) \leqslant \sum_{r=1}^{p} \pi_r \prod_{z \in \pi_r, f(z) \rightarrow i} \sum_{j=1}^{m} R_{ij}(t) x_{ij}$$

$$\sum x_{i1} + x_{i2} + ... + x_{im} = 1 \text{ for all } i = 1,..,n$$

$$x_{ij} = 0 \text{ or } 1 \text{ for all } i = 1,..,n \text{ and for all } j = 1,..,m$$

The variable $x_i$ is a decision variable that denotes whether the option is chosen. The goal is to find the target reliability $R_T$ at minimum cost. Here $r$ represents the number paths from the starting node to the end node in the DTMC specification of the software system.

### 5.1.4 Random Allocation of Weibull Parameters

For our example specification problem illustrated in section 1, we consider random values to the three parameters of the Weibull Distribution. The cost and weibull parameter values for 8 components with 6 number of options for each are given below. Each row represents six options for one component:

The cost values are:

| Component Num | Cost Options | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 10 | 20 | 35 | 55 | 90 | 140 |
| 2 | 6 | 11 | 18 | 28 | 52 | 80 |
| 3 | 5 | 13 | 25 | 40 | 70 | 110 |
| 4 | 7 | 15 | 30 | 55 | 75 | 105 |
| 5 | 15 | 30 | 50 | 90 | 150 | 230 |
| 6 | 20 | 50 | 90 | 150 | 230 | 310 |
| 7 | 15 | 25 | 40 | 65 | 100 | 150 |
| 8 | 10 | 25 | 50 | 85 | 130 | 200 |

Table 5.1: Cost Values

The β parameter values are:

| Component Num | β Options | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 3.0 | 3.2 | 3.6 | 3.8 | 4.0 | 4.2 |
| 2 | 1.0 | 1.3 | 1.6 | 2.1 | 2.2 | 2.4 |
| 3 | 1.9 | 2.5 | 2.8 | 2.85 | 2.9 | 2.95 |
| 4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
| 5 | 1.5 | 1.8 | 2.0 | 2.4 | 2.8 | 3.0 |
| 6 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 |
| 7 | 2.1 | 2.4 | 2.6 | 2.7 | 2.9 | 3.29 |
| 8 | 1.8 | 2.0 | 2.1 | 2.2 | 2.3 | 2.8 |

Table 5.2: β Values

The γ parameter values are:

| Component Num | γ Options | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 100.0 | 1250.0 | 1500.0 | 80.0 | 95.0 | 1000.0 |
| 2 | 0 | 1000 | 500.0 | 0 | 1500 | 2000 |
| 3 | 1850 | 0 | 1300 | 500 | 3000 | 1500 |
| 4 | 1600 | 1500 | 800 | 1200 | 700 | 1650 |
| 5 | 1550 | 500 | 2400 | 1100 | 0 | 300 |
| 6 | 350 | 1100 | 2200 | 130 | 2000 | 250 |
| 7 | 0 | 1540 | 100 | 1143 | 134 | 1450 |
| 8 | 1100 | 1500 | 920 | 1000 | 10 | 2000 |

Table 5.3: γ Values

The η parameter value is taken as constant. The value is 10000 hours. Now, the objective is to find the minimum cost of system for different target reliabilities at different time intervals.

### 5.1.5 Optimization Procedure

The sequence of steps in the algorithm for optimization is given below:

1. Read the input parameters
2. Generate the reliability option values using Weibull Reliability function for the specified time 't'.
3. Apply the branch and bound algorithm, $EstOptConf(...)$ [1]
4. Output the result

The first step reads the input parameters such as cost options, $\gamma$, $\beta$, and $\eta$ values. The second step generates the reliability options for each of the component implementation and stores in the respective data structures. The $EstOptConf(...)$ function applies the branch and bound algorithm that finds the best configuration that gives maximum reliability with minimum cost. If it succeeds, it gives the best configuration as the output, otherwise it reports infeasible solution.

### 5.1.6 Results

The algorithms are implemented and the output is generated by the implementation program. The program is executed for the ABS system example discussed in the last chapter with the initial values given in the above tables for specific time intervals is given in the following table. Each row in the table specifies the best configuration resulted for the given desired reliability at the specified time. The first row in the table specifies that the system can provide reliability of 0.896935 up to the time 500 hours with the given best configuraion. Beyond 500 hours, it may or may not provide the required target reliability.

| Reliability | Time | Best Configuration |
|---|---|---|
| 0.896935 | 500 | 0 2 1 4 0 0 0 0 |
| 0.837457 | 1000 | 1 3 1 5 1 0 0 0 |
| 0.769903 | 1500 | 0 3 2 5 3 1 1 0 |
| 0.693545 | 2000 | 0 5 2 5 3 2 1 1 |
| 0.608944 | 2500 | 2 5 2 5 4 2 1 1 |

Table 5.4: Result

### 5.1.7 Reliability Optimization for Set Of Target Reliabilities

We have seen the problem formulation for minimizing the cost for the required target relaibility at some time $t$. Now, the problem statement can be extended to satisfy a set of target reliabilities at different incremental time slots. It means that a system configuration that satifies $R_{t_1}$ reliability at time $t_1$ and $R_{t_2}$

reliability at time $t_2$ and so on up to $k$ set of combinations.

The formula for the above description is:

$$\text{minimize } C_T(t=0) = \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} x_{ij}$$

subject to:

$$R_T(t_1) \leqslant \sum_{r=1}^{p} \pi_r \prod_{z \in \pi_r, f(z) \to i} \sum_{j=1}^{m} R_{ij}(t_1) x_{ij}$$

$$R_T(t_2) \leqslant \sum_{r=1}^{p} \pi_r \prod_{z \in \pi_r, f(z) \to i} \sum_{j=1}^{m} R_{ij}(t_2) x_{ij}$$

$$R_T(t_k) \leqslant \sum_{r=1}^{p} \pi_r \prod_{z \in \pi_r, f(z) \to i} \sum_{j=1}^{m} R_{ij}(t_k) x_{ij}$$

$$\sum x_{i1} + x_{i2} + \ldots + x_{im} = 1 \text{ for all } i = 1,..,n$$

$$x_{ij} = 0 \text{ or } 1 \text{ for all } i = 1,..,n \text{ and for all } j = 1,..,m$$

In this particular scenario, the sysem may provide a independent solutions for $R_{t_1}$ reliability at time $t_1$ and $R_{t_2}$ reliability at time $t_2$ and so on but may not provide a solution which satisfies all the desired reliabilities for specified times.

## 5.2 Reliability Optimization With Replenishment Of Components

Many complex critical systems like aircrafts undergo periodic maintenance process at fixed intervals of time. At each level of maintenance activity some of the components needs to replaced or replenished to maintain the reliability of the system to the desired level. The goal of this section is to minimize the periodic maintenance cost of the system over a period of time.

Now, We add the replenishment concept for our earliar model. At each level of maintenance actiity, some (all) of the components may be replenished to maintain the desired reliability.

### 5.2.1 Problem Description

Assume, we have selected one configuration that provides $R_S^{t_1}$ reliability upto time $t_1$. Suppose our configuration contains $n$ number of components each with $m$ number of implementation options. The present configuration selected and represented as say $\Gamma_C$ . The cost of this configuration is initial cost. Assume the model consists of $k$ number of maintenance activities. Now, Our goal is to get the system reliability to be $R_S^{t_2}$ upto time $t_2$ and so on until $R_S^{t_k}$ upto time $t_k$. We want to get the desired reliability by

either using current selected configuration for time$t_1$ or by replenishing the some of the component(s) to reach the desired reliability $R_S^{t_2}$ from time $t_1$ upto time $t_2$. The cost of the maintenance activity is calculated and added to the cost of the previous configuration. The procedure continues at each level of maintenance activity. The final cost is the sum of maintenance cost incurred at each level plus the initial cost.

For example, Assume we have 6 components and 6 number of cost and reliability options. The initial configuration is {1 , 4 , 5, 3, 2 , 6} that provides reliability $R_S^{t_1} = 0.938943$ up to time $t_1$ (250 hours) from zero hours as shown in fig 5.5. It means that the option number 1 is selected for component 1 and the option number 4 is selected for component 2 and so on. Now, to get $R_S^{t_2} = 0.9$ reliability upto time $t_2$(500 hours) from $t_1$, we can use the same configuration if it satisfies the desired reliabilty $R_S^{t_2}$. But at time $t_2$, the system can provide reliability at the most 0.778022. Now, we have replenish some of the components to achieve the desired reliability. So, we have a new set of 6 options for 6 components for replenishment along with the current availble configuration. The option of current configuration is called as dummy option or carry forward option for the next configuration. We have to select new configuration using these new set of options.

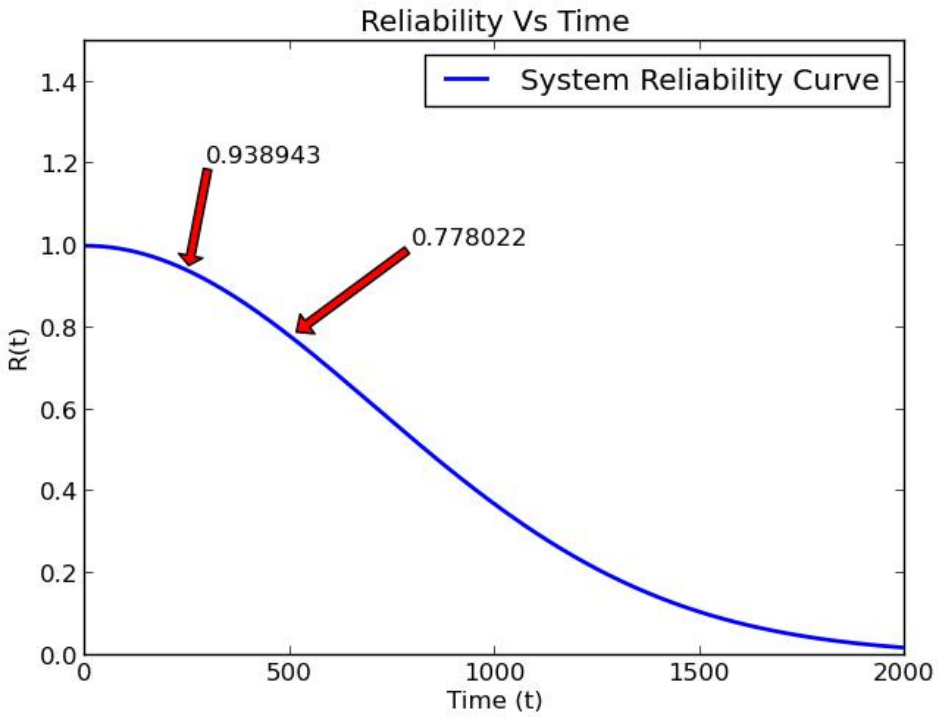


Figure 5.5: system reliability at different points

The problem here is:

1. What are the components to be replenished
2. What is the replenishment cost
3. How many components are to be replenished

The problem cannot be formulated directly into a MINLP problem, because everytime we select a configuration, we have to decide whether we have to continue with the used component or replenished component.

### 5.2.2 Solution Methodology

The solution methodology is provided here. Let there be n subsystems (components) $SS_i$, i = 1,..,n, each with a set of weibull parameters $\gamma_j$, $\beta_j$, $\eta_j$ and cost $C_j$, j = 1,...,m options. Let $C_S$ represent the total system cost. The reliability values of the options of the components are calculated by Weibull Distribution. The objective is to get $R_S^{t_1}$, a target reliability of system at time $t_1$ and $R_S^{t_2}$, a target reliability of system at time $t_2$ and so on using component replenishment. The problem formulation for a system reliability at time $t$ can be stated as:

**Notation:**

$n$ – Number of Components

$m$ – Number of implementation options for each component

$C_{i1}, C_{i1},...,C_{im}$: - Cost options of component $i$ at time $t = 0$ (Cost is constant)

$R_{i1}^t, R_{i1}^t, ..., R_{im}^t$: - Reliability options of component $i$ at time $t$, which should be calculated using Weibull Reliability formula

$\beta_{i1}, \beta_{i1}, ..., \beta_{im}$: - $\beta$ options of component $i$ at time $t = 0$

$\gamma_{i1}, \gamma_{i1}, ..., \gamma_{im}$: - $\gamma$ options of component $i$ at time $t = 0$ and these values are zero beacause intitial time t=0

$\eta_{i1}, \eta_{i1}, ..., \eta_{im}$: - $\eta$ options of component $i$ at time $t = 0$ and is taken as constart for all components

$\pi_r$ - the path probability of 'r'th path,

$f(z) \rightarrow i$ - the mapping function from node to component

The initial configuration for a target reliability of system $R_{t_1}$, up to time $t_1$, can be obtained by using the formulation we have done earliar and is represented as $\Gamma^0$. But for the next configuration, we have to choose whether the component option that has been selected already can be used or replenished. It means that, there are total $m+1$ options for each of the component now, where $m$ new options and one option from the previous configuration. The $m+1$ option is taken as dummy option for the initial configuration with the cost and reliability value as zero. So, Now the formulation can be done as follows:

**Definition 8.** ***Configuration for time*** $t_k$**,** $\Gamma^{t_k}$**:** *The configuration that provides at least the system reliability of* $R_S^{t_k}$ *from time* $t_{k-1}$ *to* $t_k$.

Now, Let

$\Gamma^{t_k}$ = Configuration for time $t_k$

$\Gamma_i^{t_k}$ = '$i$'th component option of configuration for time $t_k$

$\gamma_{i,j}^{t_k} = t_{k-1}$, The $\gamma$ for the new options for time $t_k$, $1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$

$\gamma_{i,m+1}^{t_k} = \gamma_i^{t_{k-1}}(\Gamma_i^{t_{k-1}})$ The $\gamma$ of carried over option of component from previous configuration for time $t_{k-1}$

$\beta_{i,m+1}^{t_k} = \beta_i^{t_{k-1}}(\Gamma_i^{t_{k-1}})$ The $\beta$ of carried over option of component from previous configuration for time $t_{k-1}$

$\gamma_{i,m+1}^{t_1} = \beta_{i,m+1}^{t_1} = \eta_{i,m+1}^{t_1} = 0$, The dummy options of configuration at time $t_1$, initial configuration

$C_{i,m+1} = 0$, The cost of the carried over option

The minimization formulation is:

$$\text{minimize } C_T^{t_k} = \sum_{i=1}^{n} \sum_{j=1}^{m+1} C_{ij} x_{ij}$$

subject to

$$R_S^t \leqslant \sum_{r=1}^{p} \pi_r \prod_{z \in \pi_r, f(z) \to i} \sum_{j=1}^{m+1} R_{ij}^{t_k} x_{ij}$$

$$\sum x_{i1} + x_{i2} + \ldots + x_{im+1} = 1 \text{ for all } i = 1,..,n$$

$$x_{ij} = 0 \text{ or } 1 \text{ for all } i = 1,..,n \text{ and for all } j = 1,..,m+1$$

By using the above formula, The result configuration we get is represented as $\Gamma^{temp}$. If the result configuration contains any option that is equal to $m+1$ for any component, then it is clear that the option is the carried over option from the previous configuration and the particular component is not replenished. So, a mapping has to be done from the $m+1$ option to its original represented option in the previous configuration. The mapping function is defined as:

$$\Gamma_i^{t_k} = g(\Gamma_i^{temp}, \Gamma_i^{t_{k-1}})$$

where $g$ is a function that maps the $m+1$ option of $\Gamma_i^{temp}$ to the equivalent option from the configuration $\Gamma_i^{t-1}$ and $\Gamma_i^{t_k}$ represents the final result configuration for the time $t_k$. The function $g$ definition is given below:

**Definition 9.** ***Replenishment mapping function*** *g: A mapping function whose functionality is provided below:*

$$g(\Gamma_i^{t_k}, \Gamma_i^{t_{k-1}}) \implies if(\Gamma_i^{t_k} == m+1) then \\ \Gamma_i^{t_k} = \Gamma_i^{k-1}$$

The function must be called with $\Gamma_i^{temp}$ as first parameter and $\Gamma_i^{t_{k-1}}$ as the second parameter.
The above procedure shall be done for $k$ iterations and the total maintenance cost is defined as:

$$C_T = \sum_{i=1}^{k} C_T^{t_i}$$

### 5.2.3 Replenishment Algorithm

The Algorithmic representation of the above methodology is provided below. The statements from 1 to 10 defines the input and output of the algorithm. The function $generateWeibullReliability(...)$ generates the reliability values using the weibull distribution. The $EstOptConf(...)$ function applies the branch and bound algorithm that finds the best configuration that gives maximum reliability with minimum cost. The statement number 26 performs the replenishment mapping functionality. The output is the total system cost for the specified maintenance periods.

**Algorithm 5** Optimization With Replenishment Algorithm

**Input:**
The DTMC of system $D$, Temp Configuration$\Gamma^{temp}$ , Best Configuration $\Gamma^{t}B$
Component Vector $V_C$, Node Vector $V_N$
Beta Vector $\beta_{n,m+1}$ , Gamma Vector $\gamma_{n,m+1}$, $eta$
Cost Vector $C_{n,m+1}$ , Reliability Vector $R_{n,m+1}$
System Cost Vector $C^k$, Total System Cost $C^T$
Target Reliabilitity Vector $R_T^k, 1 \leqslant k \leqslant p$
Time Vector $t_k, 1 \leqslant i \leqslant p$
**Output:**
Total System Cost $C^T$
**Initialization:** $C^T \leftarrow 0$
**optimizationWithReplenshment()**
{
**for** $k = 1 \rightarrow p$ **do** // $p$ maintenance times
    **if** $k = 1$ **then**
        $\gamma_{i,m+1}^{t_k} = \beta_{i,m+1}^{t_k} = C_{i,m+1} = 0$;
        $generateWeibullReliability(R, \gamma^{t_k}, \beta^{t_k}, \eta)$;
        $R_{i,m+1} = 0$;
    **else**
        $\beta_{i,m+1}^{t_k} = \beta_i^{t_{k-1}}(\Gamma_i^{t_{k-1}})$;
        $\gamma_{i,m+1}^{t_k} = \gamma_i^{t_{k-1}}(\Gamma_i^{t_{k-1}})$;
        $generateWeibullReliability(R, \gamma^{t_k}, \beta^{t_k}, \eta)$;
        $C_{i,m+1} = 0$;
    **end if**
    $C^k \leftarrow EstOptConf(D, V_C, V_N, R, C, R_T^k, \Gamma^{temp})$;
    $\Gamma_i^{t_k} = g(\Gamma_i^{temp}, \Gamma_i^{t_{k-1}}), 1 \leqslant i \leqslant n$
    $C^T \leftarrow C^T + C^k$;
**end for**
}

### 5.2.4 Results and Conclusion

The algorithm for above method is implemented for Adaptive Cruise Control(ACC) system. A Adaptive Cruise Control (ACC) is an automotive feature that automatically maintains the speed of a vehicle at a constant value set by the driver by taking control of the throttle and brakes. We run the algorithm for k = 5 iterations and the results are provided in the following tables.

The table given below modelled for ACC system with 8 components and 6 options for each of the component. The implementation of the ACC function is 28 lines long with 6 conditional statements. The DTMC representation of this model contains 63 states and 94 transitions. The number 6 in the best configuration column represents the component that is not replenished. All other components that have other than 6 as the option are replenished. The

| Reliability | Time | Best Configuration With $\gamma$ Zero |
|---|---|---|
| 0.385000 | 1000 | 0 1 1 3 0 0 0 0 |
| 0.384000 | 2000 | 6 1 1 4 6 6 6 0 |
| 0.383000 | 3000 | 6 2 1 4 0 6 6 0 |
| 0.382000 | 4000 | 0 2 0 4 6 6 0 0 |
| 0.381000 | 5000 | 6 2 0 4 0 6 6 0 |

Table 5.5: Result with replenishment - 8 components

The table below shows the results with varied $\gamma$ values as specified in the table 5.3. The varied $\gamma$ values may indicate that the components may have been already used.

| Reliability | Time | Best Configuration With Varied $\gamma$ |
|---|---|---|
| 0.385000 | 1000 | 3 3 1 4 4 3 2 4 |
| 0.384000 | 2000 | 0 3 1 4 4 3 2 4 |
| 0.383000 | 3000 | 3 3 1 4 6 3 2 4 |
| 0.382000 | 4000 | 0 3 1 4 4 0 2 4 |
| 0.381000 | 5000 | 4 3 1 4 6 0 2 4 |

Table 5.6: Result with replenishment - 8 components

The table given below modelled for ACC system with 10 components and 6 options for each of the component. The implementation of the ACC function same as described above. The number 10 in the best configuration column represents the component that is not replenished. All other components that have other than 10 as the option are replenished.

| Reliability | Time | Best Configuration |
|---|---|---|
| 0.50000 | 1000 | 0 5 2 3 0 2 0 5 0 5 |
| 0.49000 | 2000 | 0 5 2 2 0 6 0 4 0 5 |
| 0.48000 | 3000 | 0 4 1 2 6 1 6 4 0 5 |
| 0.47000 | 4000 | 1 3 1 1 0 6 0 4 1 4 |
| 0.46000 | 5000 | 0 3 1 1 6 0 6 3 0 4 |

Table 5.7: Result with replenishment - 10 components

We explored a system with different combinations of components and options such as 10 components with 10 options and 20 components with 6 options etc. But even one iteration could not be completed after 2 days in the second case because of the huge state space to be explored. So we couldnot proceed further.

To conclude, the proposed methodology finds the total cost of the system for a given number of maintenance periods using exhastive search of state space. However, the nature of the search space is amenable to more focused search strategies. The future work is to investigate such search strategies.

# Bibliography


[1] “RELSPEC: A framework Early Reliability Refinement of Embedded Applications”, Saurav Kumar Ghosh, Aritra Hazra, Dr Soumyajit Dey, CSE Department, IIT Kharagpur

[2] Chao-Jung Hsu and Chin-Yu Huang. An adaptive reliability analysis using path testing for complex component-based software systems. Reliability, IEEE Transactions on, 60(1):158–170, 2011.

[3] [mettas00] Mettas A, Reliability allocation and optimization for complex systems. Proceedings Annual Reliability and Maintainability Symposium, Los Angeles, CA, January 2000, 216-221

[4] Cristiana Bolchini, Antonio Miele, Reliability-Driven System-Level Synthesis for Mixed-Critical Embedded Systems, IEEE Transactions on computers,VOL.62,NO.12,December 2013

[5] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. John Wiley & Sons, 2006.

[6] Coremen, Introduction to Algorithms, 3rd edition

[7] I. Koren and C. M. Krishna. Fault-Tolerant Systems. Morgan Kaufmann, 2007

[8] https://projects.coin-or.org/Bonmin

[9] http://www.mathworks.in/help/gads/constrained-minimization-using-ga.html

[10] http://www.i2c2.aut.ac.nz/Wiki/OPTI/index.php/Probs/MINLP

[11] http://scip.zib.de/

[12] http://www.weibull.com

[13] http://reliawiki.org/index.php/The_Weibull_Distribution

[14] T. D. Day and S. G. Roberts. A simulation model for vehicle braking systems fitted with abs. Technical report, SAE Technical Paper, 2002.

[15] W. J. Chundrlik Jr and P. I. Labuhn. Adaptive cruise control, October 3 1995. US Patent 5,454,442.

[16] Oded Berman and Noushin Ashrafi, Optimization Models for Reliability of Modular Software Systems, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 19, NO. 11, NOVEMBER 1993

[17] Oded Berman and Noushin Ashrafi, Optimization Models for Selection of Programs, Considering Cost & Reliability, IEEE TRANSACTIONS ON RELIABILITY, VOL. 41, NO. 2, 1992 JUNE