SABANCI UNIVERSITY

MASTER THESIS

Fault Localization for a Series System When Tests Are Unreliable

Author:

Zahed Shahmoradi

Supervisor:

Tonguç Ünlüyurt

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of the requirements for the degree of

Master of Science

Sabanci University

January 2016

FAULT LOCALIZATION FOR SERIES SYSTEMS WHEN
TESTS ARE UNRELIABLE

APPROVED BY:

Assoc. Prof. Dr. Tonguç Ünlüyurt .... [illegible]
(Thesis Supervisor)

Prof. Dr. Barış Balcıoğlu .... [illegible]

Assoc. Prof. Dr. Erhun Kundakcıoğlu .... [illegible]

DATE OF APPROVAL: 22/12/2015

Fault Localization for a Series System When Tests Are Unreliable

ZAHED SHAHMORADI

IE, M.Sc. Thesis, 2015

Thesis Supervisor: Assoc. Prof. Dr. Tonguç Ünlüyurt

Keywords: Series System, Repeating Unreliable Tests, Fault Localization.

Abstract

In this study, we consider a failed series system in which any of the components in the system can be the cause of the failure with different independent probabilities. We are allowed to sequentially test the components in the system to localize the faulty one. Prior probability that a component is the cause of the failure as well as the cost of testing a component are known. We consider unreliable tests in which a test can identify a component as working when in reality it is down, and vice versa. Therefore, there are costs corresponding to misclassification of the components in the system and the total expected cost of diagnosing becomes sum of inspection costs and misclassification costs. In this study we propose a model in which the repetition of tests are allowed at most once to minimize the total expected cost. Therefore, the aim here is to not only determine the best test sequence, but also the best repetition strategy with minimum expected cost. The complete mathematical model which covers all repetition strategies is introduced. Different characteristics of the model are discussed and numerical results are presented to demonstrate the possible cost reductions through repetition of tests.

ÇALIŞMAYAN BIR SERI SISTEMDE HATALI BILEŞENIN GÜVENILIR OLMAYAN TESTLERLE BULUNMASI

ZAHED SHAHMORADI

IE, M.Sc. Tezi, 2015

Tez danışma: Doç. Dr. Tonguç Ünlüyurt

Anahtar Kelimeler: Seri sistem, Güvenilir olmayan testleri tekrar etme, Hatalı bileçen bulma.

ÖZET

Bu çalışmada her hangi bir bileşeninin çalışmaması sebebiyle çalışmayan bir seri sistem ele alınmaktadır. Bileşenler tek tek sınanarak hangi bileşenin hatalı olduğunu bulmak ana amaçtır. Ancak bileşenlerin sınanması maliyetlidir ve bileşenlerin sistemin çalışmamasına sebep olma ihtimalleri bilinmektedir. Sınama işlemi için kullanılan testler ise yanlış sonuçlar verebilmektedir. Bu çalışmada toplam beklenen maliyeti (sınama maliyeti ve çalışmayan bileşenin yanlış belirlenmesinin maliyeti) enazlayan stratejiyi bulmaya çalışıyoruz. Testler yanlış sonuç verebildikleri için en çok bir kez olmak kaydıyla testlerin tekrarına izin veriyoruz. Bu durumda bir strateji bileşenlerin bir sıralaması ve tekrar edilip edilmeyeceği bilgisinden oluşmaktadır. Bununla ilgili matematiksel model ortaya konmakta, modelin özellikleri çalışmakta, toplam maliyeti hesaplayan formüller türetilmekte ve testleri tekrar ettiğimiz durumda elde edilebilecek maliyet avantajı sayısal sonuçlarla gösterilmektedir.

Acknowledgements

First and foremost, I offer my sincerest gratitude to my adviser, Dr Tonguç Ünlüyurt, who has supported me throughout my thesis. I attribute the level of my Masters degree to his encouragement and effort and without him this thesis, too, would not have been completed or written. One simply could not wish for a better or friendlier adviser.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Barış Balcıoğlu, Assoc. Prof. Erhun Kundakcıoğlu, for their time and consideration.

Last but not the least important, I owe more than thanks to my family members which includes my parents and elder brothers and sisters, for their encouragement throughout my life. Without their support, it is impossible for me to finish my graduate education seamlessly.

Contents

Acknowledgements iv

Contents vi

List of Figures vii

List of Tables vii

Symbols viii

1 Introduction 1

2 Problem definition and Literature Review 3

2.1 Problem Definition 3

2.2 Literature Review 8

3 Models and Formulations 11

3.1 Models 11

3.1.1 Repetition After a *Positive* Outcome 11

3.1.2 Repetition After a *Negative* Outcome 22

4 Discussion 25

4.1 Complexity 25

4.2 Algorithms 26

4.3 Instance Generation 29

4.4 Computational Results 30

4.5 Conclusion and Future Research 36

References 38

List of Figures

2.1 Decision tree for basic sequential testing policy. 5

3.1 Decision tree for repetition policy when $V = \mathbf{1}$. 12
3.2 Examples of partial repetition for a system with $N = 3$ components: (A).$V = (0, 1, 1)$, (B).$V = (1, 0, 1)$, (C).$V = (1, 1, 0)$, (D).$V = (0, 1, 0)$. 20
3.3 Decision tree for negative repetition policy. 22

4.1 Percentage of repetitions for different parameters in both models . . 37

List of Tables

2.1 Type-I and type-II errors 4

4.1 Growth of the feasible space size by increasing N 26
4.2 Parameters of Experimental Design 30
4.3 Optimal results for the model of repeating after a *positive* outcome,($N = 8$) 31
4.4 Optimal results for the model of repeating after a *negative* outcome,($N = 8$) 31
4.5 Results of large instances for the model of repeating after a *positive* outcome, average of $N = 10, 25, 50$ 32
4.6 Results of large instances for the model of repeating after a *positive* outcome, average of $N = 75, 100$ 33
4.7 Results of large instances for the model of repeating after a *negative* outcome, average of $N = 10, 25, 50$ 33
4.8 Results of large instances for the model of repeating after a *negative* outcome, average of $N = 75, 100$ 34
4.9 Ratio algorithm performance in different instances 35

Symbols

$M_{[i]}$	Probability of performing test i for first time .
$M_{[i']}$	Probability of performing test i for second time.
Pr_{fp}	Probability of terminating the diagnosis with a false positive result.
Pr_{fn}	Probability of not finding the failed component (ending the diagnosis with a *negative* outcome for component N).
D_r	Cost of adverse consequences of a false positive error.
D_n	Cost of adverse consequences of a false negative error.
α_i	Pr(positive test for component i | component i is not failed).
β_i	Pr(negative test for component i | component i is failed).
P_i	Pr(component i has failed | system has failed).
C_i	Cost of testing component i.
S	$\{[1],[2],...,[N]\}$: vector of the sequence of testing.
V	Binary vector with size N corresponding to repetition of the components.
U	Set of all possible solutions as the form $u = \begin{bmatrix} S \\ V \end{bmatrix}$

Dedicated to my family.

Chapter 1

Introduction

Although it is possible to keep complex systems running for longer time periods by using methods like preventive maintenance, one cannot totally avoid failures. Once a complex system has failed, it is very important to find out what is causing the failure with minimum total cost (or in minimum time) in the long run. In order to find out the cause of the failure, one typically needs to conduct costly tests on certain sub-systems that make up the whole system. In practical situations, the tests are not perfect. Unreliable nature of the tests which is mostly due to incorrect setup, operator mistake, unsuitable environmental condition, or imprecise measurement instruments, is considered as an important issue in the field of systems reliability and maintenance. This feature of the tests introduces a stochasticity aspect to the diagnosis process. For instance, a test on a certain subsystem can provide a result of "failure" although that subsystem is in fact working. So it is possible that one can mistakenly conclude that a working subsystem is failed or a failed subsystem is working. In both cases, there can be costly consequences. In the former case, one may incur some costs due to trying to repair a working component and assuming wrongly that the system has been repaired. Whereas, in the latter case, one can try to run the whole system with a failed sub-system leading to costly problems. When the tests are imperfect, another possibility is to repeat certain tests. This way, it may be possible decrease the frequency of incorrect conclusions, hence decrease the total cost in the long run.

In this work, we consider a series-system that has just failed where tests are imperfect. We tackle the problem of identifying the cause of failure with the minimum expected cost. The total expected cost is the sum of expected inspection costs and misclassification costs. We consider two models where we allow repetition only after a positive result and only after a negative result. For this problem, a feasible solution is a permutation of the tests along with the information whether or not the test is repeated. For instance, for the case of allowing repetition after a positive result, we test the components one by one in a feasible strategy. In case, we decide to repeat a test, we conclude that it is the reason of failure if we get two positive results. Otherwise, we continue by executing the next test. Essentially, we allow a certain type of repetition. Imperfect tests have been considered in the literature in similar testing problems in [1] and [2]. Yet, these works do not allow repetition of the tests. The problem is already complicated for a series-system. A series system is a special case of many different types of systems including k-out-of-n systems, threshold systems, etc. So it is quite natural to start from investigating the problem for a series-system.

The main contributions of the thesis can be summarized as follows:

a) We introduce a model where we allow repetition of imperfect tests to decrease the total expected cost.

b) We show how to compute the expected cost of a given solution for two different repetition policies.

c) We propose a local search and a generic algorithm to find good solutions.

d) We conduct numerical experiments to demonstrate the possible gains by repeating tests and the effectiveness of the proposed algorithms.

The organization of the remainder of the thesis is as follows. We provide a formal problem definition and a literature review in chapter 2. We show how to compute the expected costs under two types of repetition policies in chapter 3. We provide the details of the local search and genetic algorithm in chapter 4. The results of the numerical experiments are presented in section 4.4 and we conclude by section 4.5 in chapter 4.

Chapter 2

Problem definition and Literature Review

2.1 Problem Definition

In this study we consider a repairable series system with N components which is not functional. Since failing at least one of the components in this system provides sufficient condition for the system to fail, we assume that cause of the failure is exactly one of the components which has recently failed. This is because the lifetime of the components are continuous random variables. We aim to locate the failed component in this system by applying a sequential test strategy. Executing a test on component j costs us an inspection cost C_j and the prior probability that component j is failed is shown with P_j. Clearly, depending on both the type of the system and the aim of diagnosis, C_j represent capital, time, pain, or depreciation of the component. Based on our assumption that there exist exactly one faulty component in the system we can clearly state that for a failed series system with N components $\sum_{i=1}^{N} P_i = 1$.

In this thesis, we attribute the terms *positive* and *negative* to discovery of down and up states of a component, respectively. Type-I ($false$ *positive*) error corresponds to the probability of deriving a positive state for component j when in reality it is active. For instance, a product on production line may fail to pass the quality control section because of a false positive alarm. On the other hand, we define type-II error ($false\ negative$) as a wrong $negative$ outcome of a test on a component. For example, this happens when a health monitoring system fails to detect the faulty part of a complex system when in reality there exist a non operative component. The following expressions as well as Table 2.1 shows the difference of two error types. Throughout this thesis we consider α and β as type-I and type-II errors respectively.

		Actual State	
		down	up
Test Result	*positive*	$1-\beta$	α
	negative	β	$1-\alpha$

Table 2.1: Type-I and type-II errors

$$\begin{aligned}\text{Type-I error} = \alpha = Pr\{\text{result positive} \mid \text{component up in reality}\},\\ \text{Type-II error} = \beta = Pr\{\text{result negative} \mid \text{component down in reality}\},\end{aligned}$$

Imprecision of a test can cause an increase in the overall inspection cost. On the one hand, repairing an active component is an unnecessary cost which is a result of a false positive. On the other hand, failing to find the faulty component is equivalent to putting the system into operation with a failed component. In this situation, the system fails to work and it might cause some more components to stop functioning.

In this study, we focus on two cost parameters other than inspection costs which corresponds to cost of adverse consequences of imprecision of the tests. We consider D_r and D_n as the constant cost coefficients for when we observe a false positive and negative outcome respectively. For example, if we assume that we are sequentially testing to locate the faulty component in our system explained previously, then at the end of the process for wrongly finding the failed component we have to pay D_r and if we fail to find the faulty component we will be paying D_n. Depending on the type of the system these two parameters can vary and take on different interpretations. For instance, in case of not locating the failed component and restarting the system, an explosion may occur or in cases in which human beings are the subject of the diagnosis, D_r might correspond to an unnecessary expensive operation.

A basic sequential testing policy for our system in which no repetition is allowed, is to start from an initial component and continue executing the tests until either we find a *positive* outcome which is an alarm for detecting the faulty component or ending the diagnosis process with a *negative* result for the N^{th} component. In the first case, the result might be correct or incorrect with certain probabilities. In the second case, we understand that we have failed to detect the target component. Without loss of generality if we consider the solution $S = \{[1], [2], [3], ..., [N]\}$ as the strategy in basic sequential testing policy, then Figure 2.1 represents the decision tree corresponding to that strategy for solution S.

Each node which has at least one successor represents a decision point in which we test the i^{th} component written inside the node. In these parent nodes, we either continue with testing $(i+1)^{th}$ component by observing a *negative* result or enter to a terminal node. All the terminal nodes with component number written inside a parenthesis are the termination of testing with detecting $(i)^{th}$ component as faulty. We enter these child nodes whenever we observe a *positive* outcome. There are N such leaf nodes that correspond to detecting any of the components as faulty. Leaf node (NF) represents a termination node in which we have observed N negative results and have failed to locate the non operative component. Numbers 0 and 1 written on the links show *negative* and *positive* results, respectively.

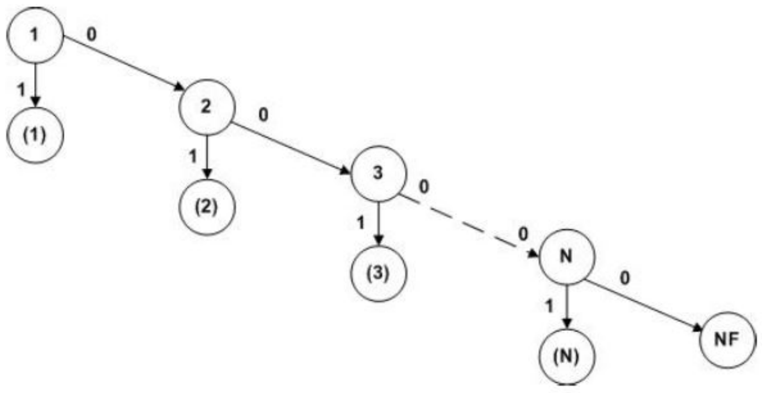

Figure 2.1: Decision tree for basic sequential testing policy.

In this thesis we aim to apply a new policy to detect the faulty component of an inoperative series system with N components in which it is assumed to possess exactly one failed component. We apply a sequential testing policy in which repetition of tests are allowed at most once immediately after performing the first test. If we look at the testing problem in the view of expected cost, we can divide the total cost into inspection cost and misclassification cost. Repeating the tests on the components of the system can improve the chance of accurate localization of the failed component and consequently it reduces the total cost of misclassification. It happens because the precision of detecting a fault correctly will increase in the second test. Therefore, the less mistake we make, the less misclassification cost we will pay.

Based on the presented argument, a trade off happens between inspection cost and misclassification cost while diagnosing a system. It provides us the opportunity to examine different test policies for discovering of the system. Repetition of tests can be established in different procedures. In a sequential testing strategy, the repetition of a test on component i can occur either right after doing the test or execution of tests on some other components can happen in between. In both of these cases, the repetition of the test happens under different circumstances. Depending on the system and the situation, one might decide to repeat the execution of all the tests for a predetermined number of times, or it can be the case for some tests with specific parameters. The repetition policy can be dynamically handled and applied. By this we mean one might be interested in repeating the tests when some sort of conditions is satisfied. Therefore the decision about repeating the test is made during the diagnosis process.

In this work we propose two repetition scenarios. In both cases, we only allow repetition right after performing the first test and it is allowed at most once. The main difference between our polices is that, in one of them repetition is allowed after observing a *positive* outcome and in the second one it is allowed when a *negative* result is observed. In order to show the repetition in our models, we consider binary row vector V with size N in which the $V(i) = 1$ or 0 represent whether to re-execute a test on component i or not, respectively.

Since we are sure about existence of a failed component in our system, our first model concentrates on improving the chance of detecting that down component. That leads us to consider a dynamic repetition policy in which we might repeat a test whenever we observe a *positive* outcome for a test. In this model, we start with an initial component and execute a test. If the result of the test turns out to be *negative* we continue the diagnosis process by testing next component. But if the outcome shows that component $[i]$ is in down state, we will check the value $v_{[i]}$ in V. If it has taken on the value 1 we repeat the test for the second time, and if it is otherwise we stop the diagnosis process by introducing element i as faulty. In the first case, where we are required to execute another test on component i, the continuation of the process will depend on the outcome of the second test. By this we mean, in the second test on the component if we again observe a *positive* outcome, we stop by determining the corresponding component as faulty. On the other hand if the result is *negative* we will move to component $[i+1]$ and continue until a component is located as failed or all the components are tested without localizing the failed element.

We believe that such a policy can be beneficial to apply in the systems which disposition of the components is highly expensive. Disposition mainly occurs as a consequence of a wrong positive outcome. When a component is detected as faulty while in reality it is in up state, we repair or replace it with a new working component and restart the system. But since our diagnosis was terminated with a false positive result, we conclude that the failed component is still in the system and the system will breakdown after it is restarted. As an example consider a system in which the repair cost of the components or the price of the components are high. In this system, one prefers to pay more in inspection process in order to prevent an unnecessary repair or replace.

In the second model that we propose, the repetition of tests are allowed at most once right after observing a *negative* outcome. Similar to other model, we start the diagnosis process by the first component in S. In any step, if a *positive* result is observed we stop diagnosing by introducing the corresponding component as faulty. On the other hand, if a *negative* result is detected for component $[i]$, we check the value of v_i in V. If the corresponding value is 0, we continue the process by testing component $[i+1]$ and if it is otherwise, the test is repeated on the component. Depending on the result of the second test, we either continue the process by testing component $[i+1]$, or stop the process by determining $[i]$ as failed. While the former case happens when a *negative* result is observed, the later one occur if the outcome is *positive*.

2.2 Literature Review

The most related study to our problem can be found in the work of Nachlas et al [1]. Our model is an extended version of testing model proposed in that study where repetition of the tests is not allowed but the tests are imperfect. They develop a basic sequential testing model and consider cost of negative consequences of test errors. They provide an algorithm which finds the optimal solution with minimum expected cost of diagnosis for small systems (less than 10 components).

Another very similar type of problems are called 'discrete sequential search problems' presented for the first time by Koopman [3]. There are a set of N boxes available and it is assumed that an item has been hidden in one of them. The prior probability that the item is located in box i is known to be p_i ($\sum p_i = 1$) and there are two costs associated with the problem. One is cost of inspection which can be interpreted as time, energy, or limitation of instruments, searchers or capital and other one is the cost of not finding the hidden object which take on be determined based on the application of the problem. An introduction to the literature of search problem is provided in L.H. Nunn's work [4] in which different aspects of the the search problem like type of the search region, movement of the target ..etc.

Unreliable tests are applied to stationary target search problems in which the parameter α_i is considered as type-II error corresponding to searching the box i [5]. α_i is the probability that the item will not be found in box i even if it is there in reality. An optimal strategy for this problem was proposed by Bellman [6] for the first time where the descending order of the ratio $(p_i(1-\alpha_i)/c_i)$ provides minimum expected cost of searching. Using this result, Glus [7] proposed an optimal detection strategy for the diagnosing model of complex multi-component systems. They consider a system with N modules containing $n(1), ..., n(N)$ components where a breakdown has occurred . The aim is to find the best test solution with minimum expected time taken for test and repairmen. Although the tests are not reliable, they repeatedly perform the tests until the fault point is isolated properly. Another study in which the cost of repairing is considered, can be found in [8]. The difference is that the tests are perfect. Wegener [9] has considered the case in which a search may fail to detect the target in box i in the first look and they may recheck the box to find the object. He showed that the problem of surly finding the object has a finite expected cost and then he provided a procedure to compute the optimal strategy.More fault detection problems with imperfect tests can be found in [10][11][12].

Kress et al. [13] provides an optimal greedy algorithm for the search problem when false positive detection of the tests are considered. They also argue that for some special case of the problem the proposed algorithm maximizes a probability objective. There are studies in discrete search problems where the imperfect assumption for the tests are ignored. Which means if a test provides a *positive* result for a box (component), the item (fault point) is localized. Song et al. [14], has considered perfect diagnosis of the boxes.

Raghvan et al. [15] consider a system with a single fault point and a set of imperfect tests available each of which can be performed on a subset of components. A threshold is specified as confidence of detecting a faulty component. The repetition of the tests are allowed to improve the confidence of the diagnosis. Performing the tests are costly and when a fault source is detected it will be repaired or replaced with a specified cost. Additionally they consider a penalty for a missed repair/replacement occurrence. The best policy is the one which minimizes total expected cost of diagnosis (including all the cost sections). The problem is behaved as a partially observed Markov decision problem and continuous dynamic programming (DP) is applied to solve it. Other works in which imperfect tests are considered are [16][17].

Ding et al.[18] have consider repetition of imperfect tests on products of a production line. The main aim of their work is answering the question whether to retest conforming or non-conforming products. The purpose of retesting rejected items is to reduce the scrapping problems while the goal of retesting the accepted items is improving the outgoing quality. The general model in their work is a minimization problem where the scrapping and testing costs are minimized and the outgoing quality as well as testing machine availability are considered as constraint of the problem. They compare two different repetition policies and conclude that under certain circumstances the retesting rejected items will result in lower utilization of the testing machine.

There are studies in the literature which consider sequential test problems with perfect tests to either determine the state of a system or localize a fault point of a failed system. A comprehensive review of different types of sequential testing problems can be found in Ünlüyurt [19].

Chapter 3

Models and Formulations

3.1 Models

3.1.1 Repetition After a *Positive* Outcome

In this section, we propose an optimization model that finds the best sequential testing strategy that allows repetition with the minimum expected cost. The model includes an objective function which is going to be minimized over a feasible space U. Set U includes all possible solutions $u = \begin{bmatrix} S \\ V \end{bmatrix}$ in which S corresponds to the vector of a permutation of size N and V represents the binary row vector in the set of all binary combinations of size N. By this we mean, a solution $u_{2,N} \in U$ includes a permutation strategy and a repetition policy which both together are considered as a feasible solution for our objective function. Therefore, since there are $N!$ possible permutations and 2^N possible binary combinations for a system with size N, the size of our feasible space U will be $|U| = N!.2^N$. In order to formulate our model, we first consider a case in which the tests are repeated whenever a *positive* outcome is observed. This means we focus on the case that $V = \mathbf{1}$ in which $\mathbf{1}$ is the vector with all components equal to one.

Without loss of generality, we consider the solution $S = \{[1], [2], [3], ..., [N]\}$ in which $[i]$ represents i^{th} position in the sequence. Figure 3.1 represents the decision tree for repetition policy in a system with three components. As we can see from the tree, number of nodes increase exponentially with the system size. In comparison to the basic sequential testing policy where no repetition is allowed, here we deal with a larger tree. In basic sequential testing strategy the decision tree contains $(N + 1)$ leaf nodes which corresponds to number of possible terminations of the diagnosis. While the decision tree for repetition case can terminate in $(2^{N+1} - 1)$ possible nodes. In $(2^N - 1)$ leaf nodes, we end the testing process by detecting a component as failed and in (2^N) cases we stop by a *negative* outcome for the last component.

As it is shown in Figure 3.1, we start with component [1] and depending on the result of the test we either enter node $C1$ or $A'1$ if a *negative* or *positive* outcome occurs respectively. Each node represents a testing station for a component while the links transfer the testing process to either next component or the second test on the same component. Binary numbers 0 and 1 written on the links show *negative* and *positive* results, respectively. If a node is a station in which a test is performed on a component for the second time, we use i' notation. While those leaf nodes with a number written in a parenthesis indicates the termination node in which the corresponding component is determined as failed, other termination nodes showed by (NF) are those in which the diagnosis process in ended without isolating any faulty component.

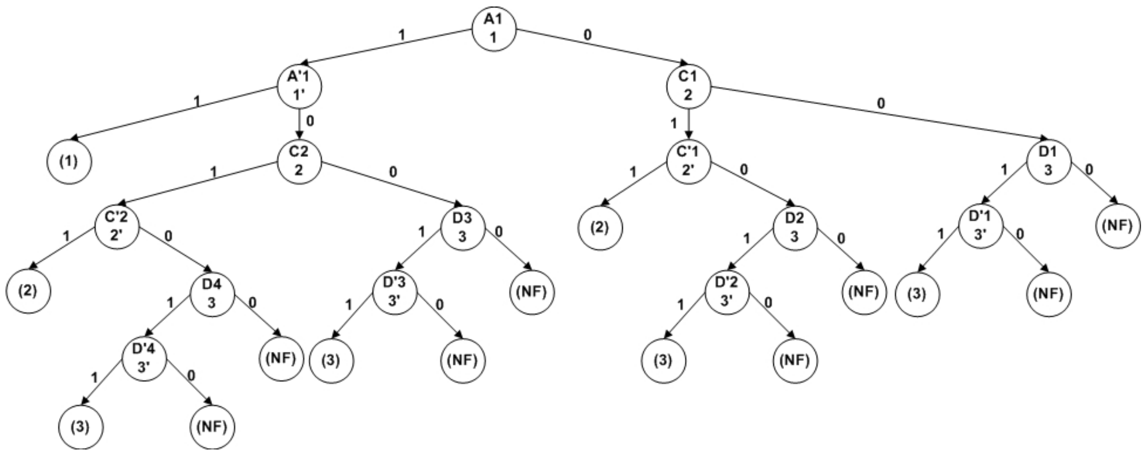

FIGURE 3.1: Decision tree for repetition policy when $V = \mathbf{1}$.

In order to explain the formulation of our problem, we first focus on inspection cost and then continue with misclassification costs. As we stated before, we consider cost C_i for executing a test on component i. On the other hand, testing component

i is an event which happens with a probability. Therefore, the inspection cost of a predetermined solution S is the expected cost of diagnosing the entire system. Equation (3.1) is the general form of inspection cost in which $H_{[i]}$ is the probability of executing test on the component tested in i^{th} order and it is the summation of all the probabilities in all the nodes in which component i is tested without noting whether it is the first or second test on the component. We show the total expected cost of inspection of S by:

$$E[IC(S)] = \sum_{i=1}^{N} C_{[i]} H_{[i]} \tag{3.1}$$

In order to make our job easier, we will have a different look at first and second test of a component. We consider $M_{[i]}$ and $M_{[i']}$ notation to show the total probability of doing test on component i for the first and second time respectively. Therefore we can rewrite the equation (3.1) as the following:

$$E[IC(S)] = \sum_{i=1}^{N} C_{[i]}(M_{[i]} + M_{[i']}) \tag{3.2}$$

As an example, in Figure 3.1, nodes $D1$ to $D4$ are those in which we execute a test for the first time and in $D^{'}1$ to $D^{'}4$ we perform the second test on the component positioned in place 3. Consequently the $M_{[3]}$ is the summation of all the probabilities of entering nodes $D1$, $D2$, $D3$, and $D4$. Therefore, if we concentrate on calculating the probabilities of occurring these events separately, we would be able to quantify the total amount of $M_{[3]}$. If we apply same calculation for all the components, we will be done with formulating the inspection cost of the complete system.

We start the diagnosis process by testing the component staying in the first position of the solution S. Thus, the probability of performing a test on component [1] is equal to 1. Performing test on component [2] occurs in nodes $C1$ and $C2$ and they depend on the results that has happened before. If we observe a *negative* outcome in node $A1$ we will enter node $C1$ which correspond to probability of doing test in that node. We calculate the probability of observing a *negative* result in any node by conditioning on the actual state of the component. A component is either in the *up* or *down* state which we can consider as two mutually exclusive events that form the sample space W. By applying Bayes theorem for two events A_1 and A_2, and event A from same sample space, (equation (3.3)) we will obtain the probability of entering node $C1$. (equation (3.4)).

$$Pr(A) = Pr(A_1)Pr(A|A_1) + Pr(A_2)Pr(A|A_2) \tag{3.3}$$

$$\begin{aligned} Pr(C1) &= Pr(''[A1] = negative'') \\ &= Pr([1] = down)Pr(''[A1] = negative''|[1] = down) \\ &+ Pr([1] = up)Pr(''[A1] = negative''|[1] = up) \\ &= \beta_{[1]}P_{[1]} + (1 - \alpha_{[1]})(1 - P_{[1]}) \end{aligned} \tag{3.4}$$

Where $Pr(C1)$ is the probability of entering node $C1$, the statement in the quotation represents the result of the test in node $A1$ and $([1] = up(down))$ indicates the actual state of first component. Next node in which a test is performed on the second component is $C2$. This event happens when a *positive* result following by a *negative* one happens for first component. The probability of entering node $A^{'}1$ is shown in equation 3.5. It is obtained by applying Bayes theorem and conditioning on the real state of component [1].

$$Pr(A^{'}1) = M_{[1^{'}]} = (1 - \beta_{[1]})P_{[1]} + \alpha_{[1]}(1 - P_{[1]}) \tag{3.5}$$

In order to calculate the probability of doing test in node $C1$, we need to obtain the probability of observing a *negative* result in node $A^{'}1$. In this situation we are given the information about the state of the first component. We are aware that one test has been performed on the first component and a *positive* outcome is observed. Thus, we can not directly apply equation (3.4) to calculate the probability of a *negative* outcome in node $A^{'}1$. Here we need to obtain a revised prior probability for component [1]. We simply condition on the the observed result to obtain the probability of real state of component [1] being down. (equation (3.6)).

$$
\begin{aligned}
P_{rev[1]inA'1} &= Pr([1] = down|''[1] = positive'' \text{ in A1}) \\
&= \frac{Pr(''[1] = positive'' \text{ in A1}|[1] = down)Pr([1] = down)}{Pr(''[1] = positive'' \text{ in A1})} \\
&= \frac{(1-\beta_{[1]})P_{[1]}}{Pr(A'1)} \\
&= \frac{(1-\beta_{[1]})P_{[1]}}{(1-\beta_{[1]})P_{[1]} + \alpha_{[1]}(1-P_{[1]})}
\end{aligned} \tag{3.6}
$$

Therefore, the probability of executing a test on component [2] in node $C2$ can be obtained as the following:

$$
\begin{aligned}
Pr(C2) &= Pr(A^{'}1)(\beta_{[1]}P_{rev[1]inA'1} + (1-\alpha_{[1]})(1-P_{rev[1]inA'1})) \\
&= \beta_{[1]}(1-\beta_{[1]})P_{[1]} + \alpha_{[1]}(1-\alpha_{[1]})(1-P_{[1]})
\end{aligned} \tag{3.7}
$$

The total probability of executing test on the component positioned in the second place for the first time is the summation of the probabilities of entering nodes $C1$ and $C2$.

$$
M_{[2]} = \beta_{[1]}P_{[1]}(2-\beta_{[1]}) + (1-P_{[1]})(1-\alpha_{[1]}^2) \tag{3.8}
$$

By applying the same technique, we would be able to determine a probability for entering each node. In each step, we should be careful about calculating the revised prior probability of component $[i]$. As we explained before, prior probability of each component to be the cause of the failure is known before starting the diagnosis process. But in each step that we perform a test and observe the result, we will be provided with more information about the system. Thus, we should dynamically re-evaluate a prior probability for the next upcoming component. This can be done by conditioning on the results that have occurred on the path starting from root node and ending at the predecessor of the corresponding node.

In the following equations we provide probabilities of first and second test on components [2] and [3].

$$M_{[2']} = \alpha_{[2]}M_{[2]} + P_{[2]}(1-\beta_{[2]}-\alpha_{[2]})(1-\alpha^2_{[1]})$$

$$\begin{aligned} M_{[3]} &= \beta_{[1]}P_{[1]}(2-\beta_{[1]})(1-\alpha^2_{[2]}) \\ &+ \beta_{[2]}P_{[2]}(2-\beta_{[2]})(1-\alpha^2_{[1]}) \\ &+ (1-P_{[1]}-P_{[2]})(1-\alpha^2_{[1]})(1-\alpha^2_{[2]}) \end{aligned} \tag{3.9}$$

$$M_{[3']} = \alpha_{[3]}M_{[3]} + P_{[3]}(1-\beta_{[3]}-\alpha_{[3]})(1-\alpha^2_{[1]})(1-\alpha^2_{[2]})$$

The total probability of executing test on component $[i]$ including both first and second tests $(H_{[i]})$ can be quantified by summing all the probabilities of entering those nodes in which a test is performed on that component. The inspection probabilities of components $[1], [2]$ and $[3]$ are provided in equation (3.10).

$$\begin{aligned} H_{[1]} &= M_{[1]} + M_{[1']} = (1+\alpha_{[1]}) + P_{[1]}(1-\alpha_{[1]}-\beta_{[1]}) \\ H_{[2]} &= (1+\alpha_{[2]})M_{[2]} + P_{[2]}(1-\beta_{[2]}-\alpha_{[2]})(1-\alpha^2_{[1]}) \\ H_{[3]} &= (1+\alpha_{[3]})M_{[3]} + P_{[3]}(1-\beta_{[3]}-\alpha_{[3]})(1-\alpha^2_{[1]})(1-\alpha^2_{[2]}) \end{aligned} \tag{3.10}$$

As we see in equations (3.9), $M_{[i']}$ is a function of $M_{[i]}$. On the other hand, we can formulate $M_{[i]}$ as a recursive function. Equation (3.11) provides both the general and recursive form of $M_{[i]}$ and also a general formulation of $M_{[i']}$ as a function of $M_{[i]}$ for a system of size N.

$$\begin{aligned} M_{[i]} &= \sum_{j=1}^{i-1} \beta_{[j]} P_{[j]} (2-\beta_{[j]}). \prod_{k=1,k\neq j}^{i-1} (1-\alpha_{[k]}^2) \\ &+ (1-\sum_{j=1}^{i-1} P_{[j]}). \prod_{k=1}^{i-1} (1-\alpha_{[k]}^2) \\ M_{[i+1]} &= (1-\alpha_{[i]}^2) M_{[i]} - P_{[i]}. \prod_{k=1}^{i} (1-\alpha_{[k]}^2) \\ &+ \beta_{[i]} P_{[i]} (2-\beta_{[i]}). \prod_{k=1}^{i-1} (1-\alpha_{[k]}^2) \\ M_{[i']} &= \alpha_{[i]} M_{[i]} + P_{[i]} (1-\alpha_{[i]}-\beta_{[i]}). \prod_{k=1}^{i-1} (1-\alpha_{[k]}^2) \end{aligned} \tag{3.11}$$

In order to quantify the total cost of misclassification, we need to find the probability of false outcomes in all termination nodes separately. We first concentrate on false *positive* outcome which might happen for component $[i]$ in a leaf node. If we consider the decision tree provided for $N = 3$ and $V = (1, 1, 1)$, in leaf nodes indicated as (i), the component $[i]$ is detected as failed which can be a true or false outcome. As we explained before, α_i is the proportion of times that the test results in *positive* if the component is in its up state. Thus, probability of a false *positive* result to be observed in a leaf node, is the product of α_i times probability that component $[i]$ is in up state. But this is the case for the situation that a test execution occurs for sure (in the root node of the decision tree). If we desire to quantify the probability of a false *positive* for a component which is tested in lower levels of the decision tree, we should also consider the probability of performing the test on component $[i]$. For example, if we consider leaf node (2) ($C'1$'s successor), we should multiply the probability of executing a test on component [2] in node $C'1$ by the probability of false *positive* outcome. Equation (3.12) provides the probability of false *positive* outcome for the entire system which is the summation over all the leaf nodes in which a false *positive* outcome can happen.

$$Pr_{fp} = \sum_{j=1}^{N} \alpha_{[j]} (M_{[j']} - P_{[j]} (1-\beta[j]). \prod_{k=1}^{j-1} (1-\alpha_{[k]}^2)) \tag{3.12}$$

By considering sample decision tree provided in Figure 3.1, the chance of not finding the failed component in the system corresponds to ending at the leaf nodes specified with (NF) notation. On the other hand, leaf nodes (NF) are the successors of those nodes in which N^{th} component is tested either for the first or second time. More importantly, they are the successors which happen after a *negative* result is discovered from testing N^{th} component. This means that if we know the probabilities of executing tests in the parent nodes of (NF) leaves, by calculating a revised probability for component $[N]$ in each node we can obtain the chance of entering those leaves. Equation (3.13) shows the total probability of ending the diagnosis process without providing any information about the faulty component for a system of size N.

$$Pr_{fn} = \sum_{j=1}^{N} \beta_{[j]} P_{[j]} (2 - \beta_{[j]}) \prod_{k=1, k \neq j}^{N} (1 - \alpha_{[k]}^2) \tag{3.13}$$

As we see in equation (3.13), Pr_{fn} is independent of the strategy. By this we mean that for any permutation solution in our problem, the chance of not localizing the failed component is a constant value.

Now we have formulated all the cost components of the objective function of complete $(V = \mathbf{1})$ repetition policy. For misclassification cost, we consider D_r and D_n which are two constant coefficients that respectively represent the cost of adverse consequences of false *positive* and false *negative* outcomes. Therefore, the total objective function for solution $u = \begin{bmatrix} S \\ V \end{bmatrix}$ in which $V = \mathbf{1}$ is the summation of total expected cost of inspection and misclassification shown in equation (3.14) which is going to be minimized over the set of all possible permutations of N.

As we stated at the beginning of the thesis, our main aim was to propose a general model which covers all the repetition policies. By this we mean, we derive a model from the one presented in equation (3.14) using which we will be able to determine not only the best permutation strategy but also the best repetition policy with minimum expected cost.

$$Z = Min\ E[C(S)] = Min\ \sum_{i=1}^{N} C_{[i]} H_{[i]} + D_r Pr_{fp} + D_n Pr_{fn}$$

$$\begin{aligned}
Z = Min \quad & \sum_{i=1}^{N} C_{[i]} \Bigg((1+\alpha_{[i]}) \Bigg(\sum_{j=1}^{i-1} \beta_{[j]} P_{[j]} (2-\beta_{[j]}). \prod_{k=1, k\neq j}^{i-1} (1-\alpha_{[k]}^2) \\
& + (1 - \sum_{j=1}^{i-1} P_{[j]}). \prod_{k=1}^{i-1} (1-\alpha_{[k]}^2) \Bigg) + \alpha_{[i]} P_{[i]} (1-\alpha_{[i]} - \beta_{[i]}). \prod_{k=1}^{i-1} (1-\alpha_{[k]}^2) \Bigg) \qquad (3.14) \\
& + D_r \Bigg(\sum_{j=1}^{N} \alpha_{[j]} (M_{[j']} - P_{[j]}(1-\beta[j]). \prod_{k=1}^{j-1} (1-\alpha_{[k]}^2)) \Bigg) \\
& + D_n \Bigg(\sum_{j=1}^{N} \beta_{[j]} P_{[j]} (2-\beta_{[j]}) \prod_{k=1, k\neq j}^{N} (1-\alpha_{[k]}^2) \Bigg) \\
& Where \;\; S \in \text{Set of all possible permutations of N}
\end{aligned}$$

In order to have better explanation about the model let us consider the Figure 3.2. As we explained before, our model provides us with a predetermined solution $u = \begin{bmatrix} S \\ V \end{bmatrix}$ in a way that both repetition policy (V) and the permutation strategy (S) will be known before the diagnosis process starts. In Figure 3.2 we provide the decision tree for different cases of V. Each time that an element of vector V takes on a 0 value, an specific part of the complete tree (when $V = \mathbf{1}$) is removed. Therefore, in total we will be provided with 2^N different trees for a fixed permutation strategy S. In order to quantify the model for each of these trees we are required to follow those steps which we applied for the model of a complete tree. Referring to the structure of the model presented in (3.14), and following the formulation steps we applied before, we can build a general binary model which covers all the repetition policies. Let us consider $V = (v_{[1]}, v_{[2]}, ..., v_{[N]})$ as the vector of N binary variables in which $v_{[i]}$ represent whether or not the i^{th} component in the permutation is repeated by taking on the values 1 and 0 respectively. Therefore, the formulation presented in (3.16) is the general optimization model for the repetition case after observing a *positive* outcome. As we can see, the presented model includes both models of no repetition policy presented in Nachlas's work as well as the every time repetition model shown in equation (3.14). These two cases can be obtained when the binary

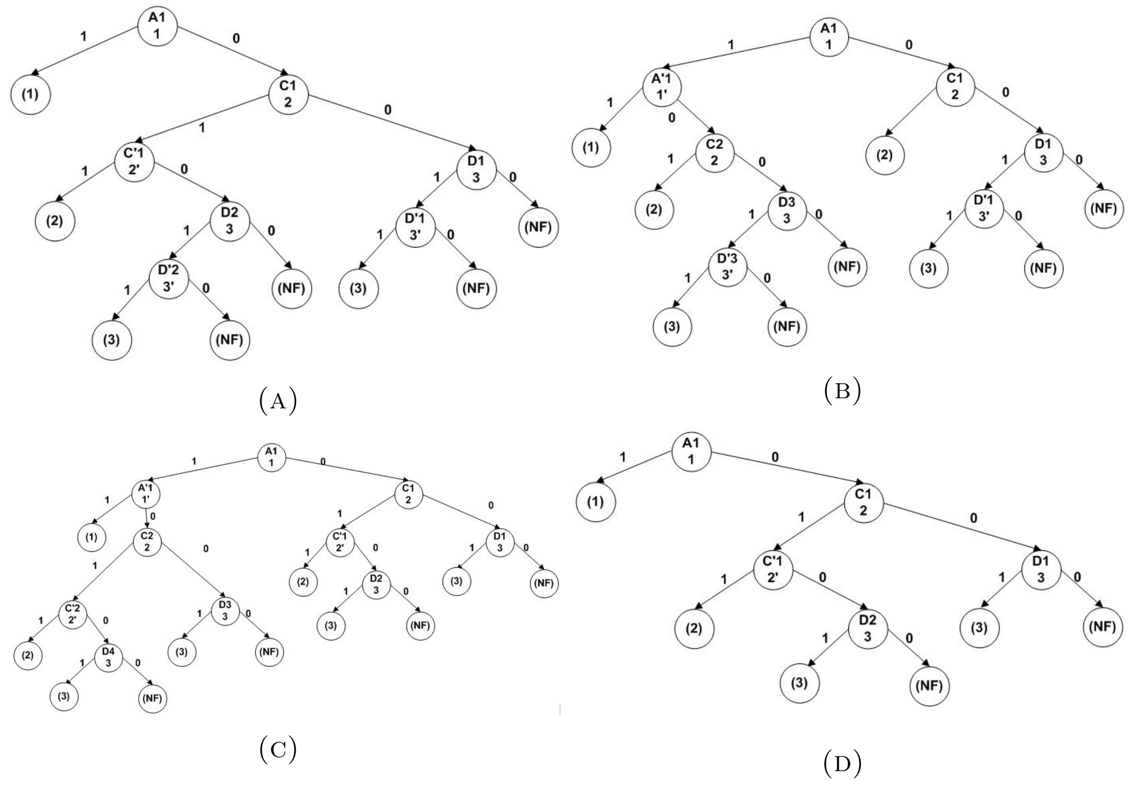

FIGURE 3.2: Examples of partial repetition for a system with $N = 3$ components: (A).$V = (0, 1, 1)$, (B).$V = (1, 0, 1)$, (C).$V = (1, 1, 0)$, (D).$V = (0, 1, 0)$.

vector $V = \mathbf{0}$ and $\mathbf{1}$ respectively.

$$Z = Min \quad E[C(S,V)] = Min \sum_{i=1}^{N} C_{[i]}(M_{[i]} + M_{[i']}) + D_r Pr_{fp} + D_n Pr_{fn}$$
$$Where \;\; u = \begin{bmatrix} S \\ V \end{bmatrix} \in U \tag{3.15}$$

Where $M_{[i]}$ and $M_{[i']}$ are the probabilities of first and second test execution on i^{th} component in the permutation strategy and:

$$
\begin{aligned}
M_{[i]} &= \sum_{j=1}^{i-1} \beta_{[j]} P_{[j]} (2-\beta_{[j]})^{v_{[j]}}. \prod_{k=1,k\neq j}^{i-1} (1-\alpha_{[k]})(1+\alpha_{[k]})^{v_{[k]}} \\
&+ (1-\sum_{k=1}^{i-1} P_{[k]}). \prod_{j=1}^{i-1} (1-\alpha_{[j]})(1+\alpha_{[j]})^{v_{[j]}} \\
M_{[i']} &= \begin{cases} \alpha_i M_{[i]} + \alpha_i (1-\sum_{j=1}^{i} P_j). \prod_{k=1}^{i-1} (1-\alpha_{[k]})(1+\alpha_{[k]})^{v_{[k]}} & \\ +P_i(1-\beta_i). \prod_{k=1}^{i-1}(1-\alpha_{[k]})(1+\alpha_{[k]})^{v_{[k]}}, & if\ v_{[i]} = 1. \\ 0, & otherwise. \end{cases} \\
Pr_{fp} &= \sum_{j=1}^{N} \alpha_j \alpha_j^{v_j} \big(M_{[i]} - P_{[j]}. \prod_{k=1}^{j-1} (1-\alpha_{[k]})(1+\alpha_{[k]})^{v_{[k]}} \\
Pr_{fn} &= \sum_{k=1}^{N} P_k \beta_k (2-\beta_k)^{v_k}. \prod_{j=1,j\neq k}^{N} (1-\alpha_j)(1+\alpha_j)^{v_j}
\end{aligned}
\tag{3.16}
$$

3.1.2 Repetition After a *Negative* Outcome

In this section we provide the details on how we formulated our second repetition model. In contrast to our first repetition policy in which a repetition of a test is performed at most once only if a *positive* result is detected, here we reiterate a test when a *negative* or *zero* outcome happens. The complete decision tree, in which a repetition of the test happens in all the cases, is presented in Figure 3.3. We use the term "negative repetition" for this policy to differentiate it from repetition after a *positive* outcome.

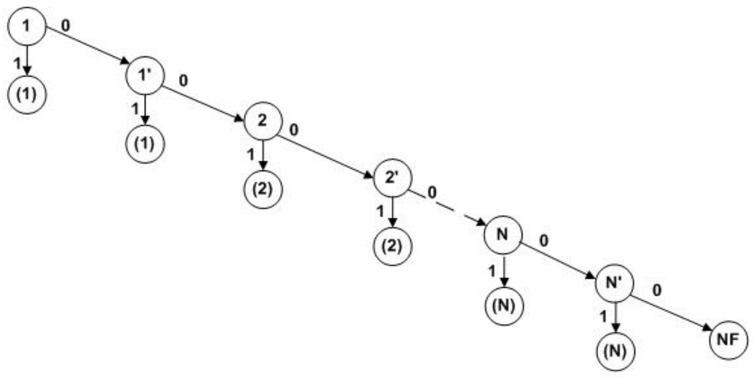

Figure 3.3: Decision tree for negative repetition policy.

Provided a permutation solution S and a repetition policy V for the the negative repetition policy, we start from first component and based on the result of the test we either terminate the diagnosis by attributing component $[i]$ as failed if a *positive* outcome occurs or continue the process by performing next test on the same component if both a *negative* outcome is observed and the corresponding binary value $v_{[i]}$ has taken on the value 1. Otherwise if $v_{[i]} = 0$, then we keep testing process by executing a test on component $[i+1]$ and continue until either we observe a *positive* result or terminate the process without isolating the failed element in the system. The i' notation in Figure 3.3 is applied to show the second (repetition of) test on component i and the termination nodes labeled with (i) notation indicate the end of process by reporting $[i]$ as faulty. In order to formulate the negative repetition policy as an optimization model we apply same technique that we used for the main repetition case. Similar to our main model, we consider three sections for the objective function. The inspection cost is the total expected cost of performing tests. In nodes labeled as (i), there exist the probability of ending with a false *positive* outcome which yields to a cost addition to the objective function. On the other hand, in node (NF) we terminate the process without isolating the faulty component which is definitely a wrong outcome as far as we have assumed that there exists exactly one failed component in the system. Therefor, we will be required to pay the cost of adverse consequences that can happen as a result of this detection.

Equation (3.17) is the general form of the objective function for the negative repetition policy. It looks for a solution u^* which minimizes the total expected cost of inspection and error costs. The feasible space for the problem is the set U including all the permutation of N, each corresponding to a strategy, as well as all possible binary combinations of N for each permutation solution S. Therefore, similar to our main model, $|U| = N!.2^N$.

$$\begin{aligned} &Z = Min \;\; E[C(S,V)] = Min \sum_{i=1}^{N} C_{[i]}(M_{[i]} + M_{[i']}) + D_r Pr_{fp} + D_n Pr_{fn} \\ &Where \;\; u = \begin{bmatrix} S \\ V \end{bmatrix} \in U \end{aligned} \tag{3.17}$$

Where similarly $M_{[i]}$ and $M_{[i']}$ represent the probabilities of performing test on i^{th} component for the first and second time respectively. Equation (3.18) shows the formulation of these parameters for a permutation solution S and binary vector V.

$$\begin{aligned}
M_{[i]} &= (1 - \sum_{j=1}^{i-1} P_{[j]}). \prod_{k=1}^{i-1} (1-\alpha_{[k]})(1-\alpha_{[k]})^{v_{[k]}} + \sum_{j=1}^{i-1} P_{[j]}\beta_{[j]}\beta_{[j]}^{v_{[j]}} \\
M_{[i']} &= \begin{cases} (1-\alpha_{[i]})M_{[i]} - P_{[i]}(1-\alpha_{[i]}-\beta_{[i]}). \prod_{k=1}^{i-1}(1-\alpha_{[k]})(1-\alpha_{[k]})^{v_{[k]}}, & if\ v_{[i]} = 1. \\ 0, & otherwise. \end{cases} \\
Pr_{fn} &= \sum_{j=1}^{N} P_j \beta_j \beta_j^{v_j} \\
Pr_{fp} &= \sum_{j=1}^{N} \alpha_{[j]}(2-\alpha_{[j]})^{v_{[j]}} \left(M_{[j]} - P_{[j]}. \prod_{k=1}^{j-1}(1-\alpha_{[k]})(1-\alpha_{[k]})^{v_{[k]}} \right)
\end{aligned} \tag{3.18}$$

Chapter 4

Discussion

4.1 Complexity

In this thesis we provided two models for two different repetition policies for the problem of diagnosing a failed series system when tests are imperfect. One of the common aspect of these two optimization models is the fact that they both search a same feasible space. In this work, although we did not provide the complexity proof for the our problem, we provide an argument on the size of the feasible region.

As we declared before, the feasible search space for our problem includes all the permutations of N and for each specific permutation solution S there exists 2^N possible binary vectors corresponding to all possible repetitions. Therefore, size of the feasible search space is $|U| = N!2^N$. In Table 4.1 we provide values for the feasible space size which shows how $|U|$ grows when N increases. In addition to the large search space of our problem, the objective functions of both our models are highly nonlinear which makes the evaluation process difficult for any solution method that directly calculates the value of the expected cost for each input solution u. These two aspects of the problem has limited us in a way that we are only able to calculate the optimal value for systems of size $N = 8$ within a reasonable time by enumerating all the possible solutions.

N	$N!$	2^N	$\|U\|$
2	2	4	8
3	6	8	48
4	24	16	384
5	120	32	3840
6	720	64	46080
7	5040	128	645120
8	40320	256	10321920
9	362880	512	185794560

Table 4.1: Growth of the feasible space size by increasing N

4.2 Algorithms

In this section we propose a Local Search (LS) and Genetic Algorithm (GA) to solve our problems. Additionally, we have applied an enumeration algorithm to find the optimal solution for small cases ($N = 8$) in order to be able to compare the performance of other algorithms by comparing their optimality gaps. Algorithm 1 is the pseudocode for our enumeration procedure.

Algorithm 1 Enumeration

```
1:  $BestCost \leftarrow$ Inf
2:  for all $S \in PermutationSet$ do
3:      for all $V \in BinaryCombinationSet$ do
4:          $u \leftarrow [S, V]'$
5:          if $ExpectedCost(u) < BestCost$ then
6:              $BestCost \leftarrow ExpectedCost(u)$
7:          end if
8:      end for
9:  end for
10: return $BestCost$
```

In order to have a better understanding of the performance of our algorithms for the instances with large values of N, we provide a simple greedy algorithm. As we previously explained the parameters of our problem, C_i and P_i are the cost of executing a test on component i and the prior probability that component i is failed respectively. For a relatively good permutation solution we normally expect to see components with less ratio C/P at the first positions. By this we mean, components with low inspection cost and high probability of being failed are preferred to be tested earlier and a permutation solution obtained by an increasing order of C/P can be interesting. Another reason why we consider this ratio is that they provide the optimal search sequence for series systems when the tests are assumed to be perfect and the only cost in the model is the inspection cost. On the other hand, α_i and β_i which are the probabilities of observing a false *positive* and *negative* results for i^{th} component respectively, are preferred to have small values. Therefore, one might be interested in the permutation solution provided by increasing order of α and β. In our greedy algorithm we arrange the components based on each of these ratios to form three permutation solutions.

In order to provide a binary vector for these permutation solution we consider the parameters associated with misclassification costs D_r and D_n. We believe that for a fixed permutation solution, the binary solution (repetition policy) which provides the minimum expected cost is highly dependent on the values of D_r and D_n. By this we mean that we expect to observe binary solutions with high density of 1 values if the value of misclassification costs are relatively high and vice versa. Therefore, we decided to determine probability values for each set of misclassification costs. This means that, any element v_i in V, takes on the value 1 with specified probability. Since we have generated three sets of values (low, medium, and high) for D_r and D_n, we also consider three probability values. For low, medium, and high misclassification costs we determined probability values 0.25, 0.5, and 0.75, respectively.

To solve our problem we first applied a local search algorithm under time limitation. In our LS we used swap moves to search the neighborhood. By this we mean that u_1 is a neighbor of u_0 if and only if they differ only in the positions of exactly two components in the solutions. As an example let us consider the case where $u_0(i) = \begin{bmatrix} m \\ 1 \end{bmatrix}$ and $u_0(j) = \begin{bmatrix} n \\ 0 \end{bmatrix}$. If a swap move is applied on u_0 then the new neighbor solution u_1 is equal to u_0 in all elements except that, $u_1(i) = \begin{bmatrix} n \\ 1 \end{bmatrix}$ and $u_1(j) = \begin{bmatrix} m \\ 0 \end{bmatrix}$. We see that a swap move not only changes the position of the components in the permutation solution, but it also changes the binary value to 0 if it is 1 and vice versa. LS starts with a random solution and by creating random swap moves continues until the provided time limit is finished.

In addition to a local search algorithm we applied a GA to solve our problem. GA is a well designed algorithm for solving different optimization problems. It is a simulation of evolutionary process of biological organisms in nature which begins with an initial population of the genes (gene pool) and aims to remove less fit genes from population and change with more qualified ones. The new generated genes which are result of crossover and mutation operators on highly fit members of population will supersede the less qualified members of population. Therefore after some iteration the population converges to optimal (best fit) solution. In this work we represent the solution (gene) as $u = \begin{bmatrix} S \\ V \end{bmatrix}$ where S and V are permutation and binary vectors with size N and i^{th} column is equal to $\begin{bmatrix} k \\ 1 \end{bmatrix}$ if component k is tested in i^{th} position and in case that repetition conditions are satisfied, the test will be repeated. Fitness value equals to the objective function value and since we are dealing with a minimization problem, solutions with less fitness value are more qualified.

We apply crossover and mutation operators to our problem as follows. We determined two values as crossover and mutation rates using which the number of generated solutions (children) in each iteration of the algorithm is calculated for each operator. According to our experiments we set 0.2 and 0.8 to be the rate of crossover and mutation operators, respectively. That means, for instance, the crossover operator will generate $0.2 * PS$ solutions in each iteration where PS is the size of the population. In all of our experiments we start with 100 solutions ($PS = 100$) as our initial gene pool. In order to explain how crossover operator produces new children from two parents we take advantage of the example with 5 components provided as follows. Let us consider $u_1 = \begin{bmatrix} 1\ 2\ 3\ 4\ 5 \\ 0\ 0\ 1\ 0\ 1 \end{bmatrix}$ and $u_2 = \begin{bmatrix} 4\ 2\ 1\ 5\ 3 \\ 1\ 1\ 0\ 1\ 0 \end{bmatrix}$. We randomly select a point for the crossover operator after which all the elements of the solutions u_1 and u_2 will change their places. New child solutions $u_1^{'} = \begin{bmatrix} 1\ 2\ 1\ 5\ 3 \\ 0\ 0\ 0\ 1\ 0 \end{bmatrix}$ and $u_2^{'} = \begin{bmatrix} 4\ 2\ 3\ 4\ 5 \\ 1\ 1\ 1\ 0\ 1 \end{bmatrix}$ show the results of the crossover operator when the crossover point was located on the second column. As we see in the permutation solution there are missing values and instead some values are observed more twice. In order to tackle this problem we generated a random permutation solution for each child solution and by starting from the first element of the solution we substitute the repeated values by first missing value from the random permutation vector. If we consider $r_1 = (3\ 4\ 1\ 2\ 5)$ and $r_2 = (5\ 4\ 1\ 3\ 2)$ as generated random vectors for $u_1^{'}$ and $u_2^{'}$, respectively, then the resulting child solutions are, $u_1^{'} = \begin{bmatrix} 1\,2\,4\,5\,3 \\ 0\,0\,0\,1\,0 \end{bmatrix}$ and $u_2^{'} = \begin{bmatrix} 4\,2\,3\,1\,5 \\ 1\,1\,1\,0\,1 \end{bmatrix}$.

The mutation operator in our GA generated new solutions in an almost similar way as LS. The swap moves change the place of two elements in the permutation vector and the binary values corresponding to them take on values 1 and 0 with probability 0.5. In order to have a better comparison with LS, we considered a same time limit for GA. Algorithm 2 provides the pseudocode for our GA.

Algorithm 2 Genetic Algorithm

```
 1: $TimeLimit \leftarrow$ t
 2: initialize the population $P := \{u_1, ..., u_N\}$
 3: while $time <= TimeLimit$ do
 4:     $PopFit \leftarrow fitness(P)$
 5:     $\{u_1^{'}, u_2^{'}\} \leftarrow BestTwoSolutions(P)$
 6:     $G_1 \leftarrow crossover(u_1^{'}, u_2^{'})$
 7:     $G_2 \leftarrow mutation(u_1^{'}, u_2^{'})$
 8:     $H \leftarrow \{P, G_1, G_2\}$
 9:     $Hfit \leftarrow sort(fitness(H))$
10:     $P \leftarrow$ first N members of H
11: end while
12: return $Best\ u \in P$
```

4.3 Instance Generation

We compare the performance of our algorithms on randomly generated instances. Firstly, we generate instances with $N = 8$ components for which we can find the optimal expected cost in a reasonable time (within 10 minutes) so that we have some idea on the observed optimality gap of the proposed algorithms. We conducted some initial experiments to determine appropriate values for the parameters of the random instances. There are in total six parameters which we should consider to generate our instances. The maximum size of a system that we consider is $N = 100$. The individual cost of performing tests (C_i) were generated uniformly between 0 and 20. For the parameters α and β, we generated three different values to see the effect of these parameters on the diagnosis process especially on percentage of repetition when they take on different values. On the other hand, the same rule was applied to generate misclassification costs and we determined three sets of values for them. Lastly, the prior probability that a component is failed (P_i) was generated uniformly between 0 and 1 but since the summation of them should be equal to 1, we normalized the results. By conducting some initial runs, the valued of the parameters were determined as shown in Table 4.2.

Factors	**Values**
N	8, 10, 25, 50, 75, 100
C	Uniform(0,20)
P	Normalize(Uniform(0,1))
α,β	Uniform(0,0.05), Uniform(0,0.15), Uniform(0,0.4)
(D_r, D_n)	(100,50), (2000,1500), (10000,8000)

Table 4.2: Parameters of Experimental Design

4.4 Computational Results

In this section, we present the results that we obtained from our experiments. All the algorithms are coded by using MATLAB. We provide the results for both of presented models and discuss them in details. First we compare solutions obtained by our algorithms for small instances $(N = 8)$ for which we have the optimal value. In this case, we focus on the optimality gaps of different algorithms.

Table 4.3 and 4.4 provide the optimal values as well as the performance of our algorithms for the *positive* and *negative* repetition policies. The values in each line are the averages of 5 instances generated with same parameter values. At the top of each parameter set the average of all instances with the corresponding parameter values are shown. For example, row $(50, 100)$ shows the average of 15 instances and each 5 instances are drawn from one specific misclassification cost value. Columns GA.Gap, LS.Gap, and Gr.Gap represents the optimality gaps GA, LS, and the Greedy algorithm, respectively. The optimality gaps, for example for GA, are calculated as the difference of GA and optimal cost over optimal cost. The column N.R.Gap provides the gap between the optimal cost and the cost value of the optimal permutation solution when there is no repetition ($V = \mathbf{0}$).

(Dn, Dr)	α, β	Opt.Cost	GA.Gap	N.R.Gap	LS.Gap	Gr.Gap
(50,100)		48.57	0.00%	3.23%	4.76%	10.23%
	U(0,0.05)	34.31	0.00%	4.24%	6.09%	9.93%
	U(0,0.15)	41.79	0.00%	4.19%	3.54%	13.18%
	U(0,0.4)	69.61	0.00%	1.26%	4.65%	7.58%
(1500,2000)		342.33	0.00%	59.51%	5.92%	34.08%
	U(0,0.05)	120.20	0.00%	95.83%	2.39%	44.31%
	U(0,0.15)	250.22	0.00%	49.02%	7.33%	33.51%
	U(0,0.4)	656.59	0.00%	33.69%	8.03%	24.41%
(10000,8000)		1562.67	0.00%	81.81%	6.28%	22.99%
	U(0,0.05)	469.34	0.00%	145.94%	2.97%	17.28%
	U(0,0.15)	1119.48	0.00%	62.27%	8.41%	31.04%
	U(0,0.4)	3099.18	0.00%	37.20%	7.48%	20.66%
Total		651.19	0.00%	48.18%	5.65%	22.43%

Table 4.3: Optimal results for the model of repeating after a *positive* outcome,($N = 8$)

(Dn, Dr)	α, β	Opt.Cost	GA.Gap	N.R.Gap	LS.Gap	Gr.Gap
(50,100)		54.71	0.00%	3.81%	7.51%	9.59%
	U(0,0.05)	35.89	0.00%	0.00%	9.82%	4.27%
	U(0,0.15)	45.37	0.00%	0.67%	9.60%	10.37%
	U(0,0.4)	82.87	0.00%	10.77%	3.13%	14.13%
(1500,2000)		512.40	0.00%	13.19%	7.18%	29.04%
	U(0,0.05)	204.92	0.00%	1.64%	6.11%	27.83%
	U(0,0.15)	348.59	0.00%	13.11%	6.05%	33.30%
	U(0,0.4)	983.69	0.00%	24.82%	9.36%	26.00%
(8000,10000)		2399.37	0.00%	18.52%	10.20%	35.08%
	U(0,0.05)	901.78	0.00%	2.98%	6.31%	42.94%
	U(0,0.15)	1589.54	0.00%	23.73%	10.43%	44.84%
	U(0,0.4)	4706.79	0.00%	28.84%	13.85%	17.46%
Total		988.83	0.00%	11.84%	8.30%	24.57%

Table 4.4: Optimal results for the model of repeating after a *negative* outcome,($N = 8$)

As we see, independent of the type of the repetition and also the parameters of the problem, GA has the best performance among all three algorithms. In all 90 instances (including both models) for which optimal value is available, the GA can obtain the optimal solution. On the other hand, by considering average values, the greedy ratio algorithm outperforms LS only in one case in the second model.

For the rest of our instances, we provide the results of each model in two tables. But since the optimal values for these cases are not available, in order to compare the performance of the algorithms we obtain the gaps of GA and LS with respect to the result of the greedy algorithm. For each model we provide two tables. Tables 4.5 and 4.7 show the results of both models for small values of N. In these tables, the values are the average of $N = 10, 25, 50$. Similarly, Tables 4.6 and 4.8 provide the average results of the instances with $N = 75, 100$ which are considered as large instances. The columns in all of these tables are defined similar to the optimal tables, except that, firstly, the gaps of GA and LS are obtained with respect to best cost obtained by the greedy algorithm and secondly, the column 'GA.N.R.Gap' is the percentage of gap between the cost of best solution obtained by GA and the cost of same solution when no repetition is allowed.

(Dn, Dr)	α, β	Gr.Cost	GA.Gap	GA.N.R.Gap	LS.Gap
(50,100)		90.08	-6.21%	-1.00%	-3.65%
	U(0,0.05)	87.09	-4.65%	-0.60%	-2.23%
	U(0,0.15)	91.04	-7.62%	-1.61%	-4.54%
	U(0,0.4)	92.10	-6.37%	-0.78%	-4.17%
(1500,2000)		824.95	-35.81%	-44.03%	-32.64%
	U(0,0.05)	372.41	-43.12%	-59.33%	-40.09%
	U(0,0.15)	788.58	-42.61%	-50.75%	-39.02%
	U(0,0.4)	1313.86	-21.70%	-22.01%	-18.80%
(8000,10000)		3301.30	-38.87%	-55.66%	-30.13%
	U(0,0.05)	1054.14	-43.65%	-71.39%	-37.15%
	U(0,0.15)	2946.49	-43.44%	-60.86%	-35.25%
	U(0,0.4)	5903.26	-29.54%	-34.74%	-18.00%
Total		1405.44	-26.97%	-33.56%	-22.14%

TABLE 4.5: Results of large instances for the model of repeating after a *positive* outcome, average of $N = 10, 25, 50$

(Dn, Dr)	α, β	Gr.Cost	GA.Gap	GA.N.R.Gap	LS.Gap
(50,100)		137.68	-9.53%	0.00%	-7.97%
	U(0,0.05)	177.25	-10.88%	0.01%	-8.58%
	U(0,0.15)	129.59	-12.41%	0.00%	-10.83%
	U(0,0.4)	106.20	-5.29%	0.00%	-4.50%
(1500,2000)		1353.91	-36.39%	-39.44%	-34.58%
	U(0,0.05)	865.28	-53.48%	-62.70%	-51.07%
	U(0,0.15)	1400.65	-41.48%	-44.64%	-39.17%
	U(0,0.4)	1795.81	-14.21%	-10.98%	-13.49%
(8000,10000)		5107.23	-41.72%	-53.31%	-39.39%
	U(0,0.05)	2371.15	-62.06%	-79.99%	-59.73%
	U(0,0.15)	5115.39	-46.41%	-59.18%	-43.25%
	U(0,0.4)	7835.14	-16.71%	-20.78%	-15.19%
Total		2199.61	-29.21%	-30.92%	-27.31%

Table 4.6: Results of large instances for the model of repeating after a *positive* outcome, average of $N = 75, 100$

(Dn, Dr)	α, β	Gr.Cost	GA.Gap	GA.N.R.Gap	LS.Gap
(50,100)		128.15	-19.15%	-10.61%	-15.14%
	U(0,0.05)	98.16	-12.39%	-0.56%	-7.12%
	U(0,0.15)	110.71	-12.97%	-8.04%	-9.80%
	U(0,0.4)	175.59	-32.09%	-23.22%	-28.50%
(1500,2000)		1486.47	-26.26%	-9.90%	-23.19%
	U(0,0.05)	665.05	-30.35%	-0.83%	-27.21%
	U(0,0.15)	1316.29	-20.99%	-5.32%	-18.83%
	U(0,0.4)	2478.06	-27.45%	-23.55%	-23.51%
(8000,10000)		6731.00	-24.23%	-10.80%	-21.13%
	U(0,0.05)	2883.38	-30.31%	-1.36%	-27.33%
	U(0,0.15)	6063.26	-20.15%	-6.50%	-17.66%
	U(0,0.4)	11246.37	-22.24%	-24.53%	-18.41%
Total		2781.87	-23.22%	-10.44%	-19.82%

Table 4.7: Results of large instances for the model of repeating after a *negative* outcome, average of $N = 10, 25, 50$

As we observe, no matter what type of repetition policy is considered, the GA performs better that LS in all the cases. On the other hand, another important

(Dn, Dr)	α, β	Gr.Cost	GA.Gap	GA.N.R.Gap	LS.Gap
(50 100)		237.24	-36.33%	-25.32%	-34.34%
	U(0,0.05)	188.08	-14.43%	-7.38%	-12.22%
	U(0,0.15)	201.12	-41.14%	-28.35%	-39.14%
	U(0,0.4)	322.52	-53.42%	-40.23%	-51.66%
(1500 2000)		2566.90	-28.63%	-16.79%	-26.45%
	U(0,0.05)	1388.11	-25.33%	-0.12%	-22.72%
	U(0,0.15)	2215.49	-22.84%	-16.40%	-21.06%
	U(0,0.4)	4097.11	-37.70%	-33.85%	-35.58%
(8000 10000)		11491.54	-27.70%	-18.31%	-25.33%
	U(0,0.05)	5956.47	-28.62%	-0.25%	-26.60%
	U(0,0.15)	9786.13	-20.64%	-19.60%	-17.95%
	U(0,0.4)	18732.03	-33.86%	-35.07%	-31.43%
Total		4765.23	-30.89%	-20.14%	-28.71%

TABLE 4.8: Results of large instances for the model of repeating after a *negative* outcome, average of $N = 75, 100$

thing in these tables is that greedy algorithm never performs better that other two algorithms. This shows how the stochastic aspect of the models has affected the results when we compare with the case that tests are perfect. The trade of between misclassification cost and inspection cost makes the total expected cost to be highly dependent on the parameters which in result prevents the ratio solutions to provide interesting solutions. The worst performance of GA and LS in the model of *positive* repetition happens mostly when misclassification costs take on their lowest value (when $(D_n, D_r) = (50, 100)$ and $\alpha, \beta \equiv U(0, 0.05)$). The reason for that can be explained as follows.

Since the value of misclassification costs are close to individual inspection costs (C_i), the trade of between inspection cost and misclassification cost is more balanced than other cases. By this we mean, the different solutions in the feasible space mostly provide the value of objective function in a tight interval and therefore, any search algorithms can hardly obtain better results. On the other hand, in these instances, the inspection cost plays a more important role in the total objective function in compare to other cases where misclassification costs take on larger values. Therefore, if we apply a suitable enough ratio which both considers C and P, we would probably find an interesting solution. As we see in Table 4.9, in the instances where misclassification costs are small, the ratio algorithm almost in all the cases returns C/P as the best solution which confirms our presented argument. As it is shown the Table 4.9, the more we increase the value of misclassification costs, the less we observe C/P solution as the best solution and the more error ratios result in better solutions. The values in Table 4.9 represent the number of times that a ratio shown in the column has been the best solution among others in specified instances in the corresponding rows.

As we stated before, both our LS and GA algorithms work under time limitation 60 seconds. But in order to compare their performances we saved the time of best solution that they find within the provided time limit. Considering all the instances, LS performs relatively faster that GA. In both models, GA spends on average 28 seconds to find the best solution while this number for LS is 9.7 seconds. For instances with $N \geq 75$, GA spends the whole time limit but the maximum time value for LS is 45 seconds which happens in same instances.

		positive			*negative*		
(Dn, Dr)	α, β	C/P	α	β	C/P	α	β
(50,100)		88	1	1	70	1	19
	U(0,0.05)	29	0	1	29	1	0
	U(0,0.15)	30	0	0	30	0	0
	U(0,0.4)	29	1	0	11	0	19
(1500,2000)		43	20	7	31	29	30
	U(0,0.05)	18	8	4	20	10	0
	U(0,0.15)	16	12	2	11	13	6
	U(0,0.4)	9	20	1	0	6	24
(8000,10000)		31	46	13	8	60	22
	U(0,0.05)	10	12	8	1	29	0
	U(0,0.15)	12	15	3	5	21	4
	U(0,0.4)	9	19	2	2	10	18

Table 4.9: Ratio algorithm performance in different instances

The main contribution of this thesis was to propose the idea of repetition of tests in the diagnosing process of a failed series system with unreliable tests. By considering the results of our experiments we provide an argument that how repetition provides better results in such problems.

In order to explain this, we first consider the column 'N.R.Gap' provided in Tables 4.3 and 4.4. As we explained, the values in these column are the cost values of the solutions of the form $u = \begin{bmatrix} S^* \\ \mathbf{0} \end{bmatrix}$, where S^* is the best permutation solution found by enumeration. As we see, even for small values of misclassification cost and error rates, no-repetition solution does not provide better results. On the other hand, we see that for the rest of our instances the more we increase misclassification cost, the more optimality gap is observed for no-repetition solution. The same behavior is observed for the other model in Table 4.4 where again the performance of no-repetition solution gets worse by increasing misclassification costs and error rates.

In the Tables 4.5 to 4.8, the column 'GA.N.R.Gap' is the gap of the no-repetition solution with respect to the cost obtained by GA. As we see again, almost in all of the instances, repeating the tests in both models results in lower expected cost on average.

In addition to the argument presented above, we provide the graph in Figure 4.1 in which the rate of repetition is shown for solutions that GA has found for both models. In Figure 4.1, labels as the form $(x, y)P$ and $(x, y)N$ show the error rates for the *positive* and *negative* repetition model, respectively. The x axis shows three values of misclassification cost and the values on the y axis represent the percentage of ones in the solutions found by GA. This value is obtained by counting the number of times that tests are repeated in a solution over the size of the system $(\sum_{i=1}^{N} v(i)/N)$.

As we observe in Figure 4.1, among all the cases where neither misclassification cost nor error rates are in their minimum level, the minimum repetition that happens is 50%. Even among the instances with minimum misclassification cost and error rates, on average 10% repetition is observed.

4.5 Conclusion and Future Research

In this thesis, we developed two new models for the problem of fault localization of series system when tests are unreliable. In our models, we allow repetition of the tests under some specific conditions. As we observed, repetition of the tests can provide considerable decrease in the overall expected cost of diagnosis when we compare with the case that no repetition is allowed. Depending on the parameters of the problem (system), one may take advantage of any of these models in order to diagnose a system.

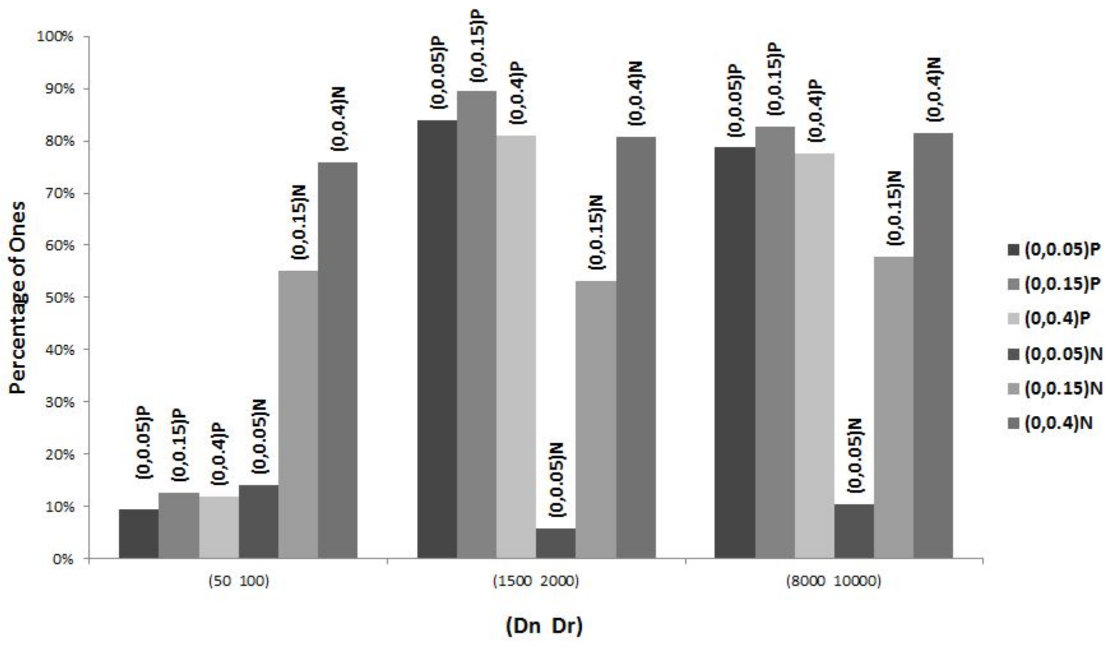

FIGURE 4.1: Percentage of repetitions for different parameters in both models

An important extension to this problem that can be stated as future work is considering the case where other types of repetition are allowed. This means that, one can either allow the repetition of the tests more than once, or a repetition of a test may not be performed right after the first test. In these cases, the size of tree will becomes very large and it results in more complicated computations. Another worthwhile extension can be the case where repetitions are allowed after both *positive* and *negative* outcomes. In this case, the stopping condition should be defined properly. Again in this model the size of the decision tree grows fast and even for small instances we face large trees.

Applying a repetition policy in diagnosing more general systems like k out-of-n and threshold can be considered as a future work. Some assumptions of the problem, like the single fault assumption, as well as the size of the decision tree may change depending on the type of the system. Furthermore, in these systems, it may not be possible to describe the solutions as permutations and therefore it makes the computation more difficult.

References

[1] Nachlas, Joel, Susan R. Loney, and Blair Binney. Diagnostic-strategy selection for series systems. *Reliability, IEEE Transactions on*, 39(3):273–280, 1990.

[2] W. Wenchao, F. N. Talla, and R. Leus. Sequential diagnosis of k-out-of-n systems with imperfect tests. *Available at SSRN 2381951*, 2013.

[3] B. O. Koopman. *Search and Screening*. 1946.

[4] Laura H. Nunn. An introduction to the literature of search theory. *CENTER FOR NAVAL ANALYSES ALEXANDRIA VA OPERATIONS EVALUATION GROUP*, (CNA-PP-305), 1981.

[5] D. Matula. A periodic optimal search. *American Mathematical Monthly*, 142(1-3):15–21, 1964.

[6] R. Bellman. The theory of dynamic programming. *RAND CORP SANTA MONICA CA*, (RAND-P-550), 1954.

[7] B. Gluss. An optimum policy for detecting a fault in a complex system. *Operations Research*, 7(4):468–477, 1959.

[8] M. Y. Kovalyov, M. C.Portmann, and A. Oulamara. Optimal testing and repairing a failed series system. *Journal of combinatorial optimization*, 12(3):279–295, 2006.

[9] I. Wegener. The discrete sequential search problem with nonrandom cost and overlook probabilities. *Mathematics of Operations Research*, 5(3):373–380, 1980.

[10] A. Balakrishnan and T. Semmelbauer. Circuit diagnosis support system for electronics assembly operations. *Decision support systems*, 25(4):251–269, 1999.

[11] S. Ruan and et al. Dynamic multiple-fault diagnosis with imperfect tests. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(6):1224–1236, 2009.

[12] B. Wang and et al. Fault localization using passive end-to-end measurements and sequential testing for wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 11(3):439–452, 2012.

[13] M. Kress, K. Y. Lin, and R. Szechtman. Optimal discrete search with imperfect specificity. *Mathematical methods of operations research*, 68(3):539–549, 2008.

[14] N. O. Song and D. Teneketzis. Discrete search with multiple sensors. *Mathematical Methods of Operations Research*, 60(1):1–13, 2004.

[15] V. Raghavan, M. Shakeri, and K. Pattipati. Test sequencing algorithms with unreliable tests. *Systems and Humans, IEEE Transactions on*, 29(4):347–357, 1999.

[16] S. V. Amari, H. Pham, and G. Dill. Optimal design of k-out-of-n: G subsystems subjected to imperfect fault-coverage. *Reliability, IEEE Transactions on*, 53(4):567–575, 2004.

[17] W. Wang, Y. Daren, and H. Qinghua. Test sequencing strategy with imperfect test. *Electronic Measurement and Instruments*, (ICEMI'07), 2007.

[18] J. Ding, S. Betsy, Greenberg, and H. Matsuo. Repetitive testing strategies when the testing process is imperfect. *Management Science*, 4(10):1367–1378, 1998.

[19] T. Ünlüyurt. Sequential testing of complex systems: a review. *Discrete Applied Mathematics*, 142(1):189–205, 2004.