TESTING STRATEGIES FOR k-out-of-n SYSTEMS UNDER GENERAL TYPE PRECEDENCE CONSTRAINTS

by

ELİF ÖZDEMİR

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
July 2011

TESTING STRATEGIES FOR k-out-of-n SYSTEMS PRECEDENCE CONSTRAINTS

APPROVED BY

Assoc. Prof. Tonguç Ünlüyurt (Thesis Supervisor)	[illegible]
Assoc. Prof. Özgür Özlük	[illegible]
Assoc. Prof. Bülent Çatay	[illegible]
Assoc. Prof. Ş. İlker Birbil	[illegible]
Assist. Prof. Murat Kaya	[illegible]

DATE OF APPROVAL: 27 July 2011

to my family

Acknowledgments

I would like to express my deepest gratitude to my thesis advisor Assoc. Prof. Tonguç Ünlüyurt for his invaluable supervision, patience, motivation, enthusiasm throughout my thesis project. I would also want to thank Assoc. Prof. Özgür Özlük for his concern and encouragement.

I am grateful to all my friends for their caring and support. Very special thanks to my dear friends Ezgi Yıldız, Gizem Kılıçaslan, Gökmen Polat, Mahir Umman Yıldırım, Merve Şeker, Nimet Aksoy, Nükte Şahin, Özlem Çoban, Pınar Efe and Semih Yalçındağ. Without their encouragement and understanding it would have been impossible for me to finish this work.

Lastly, I offer my special regards and blessings to my family for their concern, love and support they provided throughout my life.

TESTING STRATEGIES FOR k-out-of-n SYSTEMS UNDER GENERAL TYPE PRECEDENCE CONSTRAINTS

Elif Özdemir

Industrial Engineering, Master of Science Thesis, 2011

Thesis Supervisor: Assoc. Prof. Tonguç Ünlüyurt

Keywords: k-out-of-n system, tabu search, simulated annealing, general type precedence constraint, sequential testing

Abstract

This thesis investigates diagnosis strategies for k-out-of-n systems under the general type precedence constraints. Given the testing costs and the prior working probabilities, the problem is to devise strategies that minimizes the total expected cost of finding the correct state of the system. The true state of the system is determined by sequential inspection of these n components. We try to find good strategies for the problem under general type precedence constraints by adapting an optimal algorithm that works when there are no precedence constraints. We refer to this algorithm Intersection-Precedence and represent the strategy that we obtain efficiently by a Block-Walking Diagram structure. Since no computational results are reported in the literature for this particular problem, in order to benchmark the performance of the Intersection-Precedence algorithm, we develop Tabu Search and Simulated Annealing algorithms that find permutation strategies.We conduct an extensive computational study to compare the results obtained by the alternative algorithms and we observe that Intersection-Precedence algorithm, in general, outperforms the other algorithms.

Elif Özdemir

Endüstri Mühendisliği, Yüksek Lisans Tezi, 2011

Tez Danışman: Doç. Dr. Tonguç Ünlüyurt

Anahtar Kelimeler: n'nin k'lısı sistemler, tabu arama, benzetilmiş tavlama, genel tipte öncelik kısıtları, sıralı test etme

Özet

Bu tez nin klısı (k-out-of-n) sistemlerde genel tipte öncelik kısıtları oduğu zaman, tanılama stratejilerini araştırmaktadır. Başarılı çalışma olasılıkları ve test etme maliyetleri önceden belli n tane bağımsız bileşenden oluşan bu problem sistemin doğru durumunu olurlu bir strateji ile belirlemenin beklenen maliyetini en aza indirmeyi hedeflemektedir. Sistemin gerçek durumu bileşenlerinin sırayla test edilmesiyle tespit edilir. Öncelik kısıtlarının olmadığı durumda en iyi çalışan bir algoritma, genel tipte öncelik kısıtlarının olduğu duruma uyarlanmıştır. Bu algoritma Kesişim-Öncelik olarak isimlendirilmiş ve elde edilen strateji etkili bir biçimde Block-Walking Diyagram yapısı ile gösterilmiştir. Literatürde bu problem için sayısal çalışmalar bulunmadığı için algoritmanın performansını kıyaslamak adına, permütasyon stratejileri bulmak için Tabu Arama ve Benzetilmiş Tavlama algoritmaları oluşturulmuştur. Önerilen alternatif algoritmaları analiz etmek ve önerilen çözüm yöntemlerinin hesaplama etkinliğini göstermek amacıyla kapsamlı bir sayısal çalışma yapılmıştır.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION AND MOTIVATION

We deal with very complex structures in almost all service and production systems and our daily lives. Typical examples are telecommunication systems, manufacturing systems, mechanical and/or electronic products etc. These systems typically consist of subsystems or components. The state of the whole system is described by a function of the states of these subsystems. For instance, the state of a telecommunications system can be defined by the existence of a functional path between two specific nodes of the network. In this particular example, the state of the system is *working* if there exists a path that consists of working components and *failure*, otherwise. Often, it is necessary to diagnose these systems and find out the correct state of the system with the minimum cost. In order to do this, we need to learn the states of the subsystems. We assume that in order to learn the correct state of the subsystems, we need to conduct costly tests. In most cases, it is not necessary to learn the correct state of all the components or subsystems. For instance, in the above example, once we detect a path that consists of functioning components, we do not need to learn the states of the remaining components. So it is important to develop a strategy to carry out the diagnosis procedure with a minimum total expected cost.

The variations of the general testing problem have many application areas such as classification of pattern vectors [8], file screening/searching applications [18], maintenance operations [4], plant pathology, medical diagnosis, decision table programming, computerized banking, pattern recognition, nuclear power plant control [12, 21, 22], testing incoming patients against some rare but dangerous disease [9], discriminant analysis of test data, reliability analysis of coherent systems, research and development planning, communication networks, speech/voice recognition, distributed computing, and in the design of interactive expert systems [11],wafer probe testing in electrical engineering [6], best value, or satisfying search algorithms in artificial intelligence [17], organization and criterion of an applied research project [19]. As an example for application areas of testing problem, Bert and Roel [3] examined to maximize the net present value in project scheduling when the activities have failure probabilities. They determined the overall project termination criteria depended on the failure probabilities of activities. They showed that the problem is NP-hard.

In this study, we work on a related problem, namely, the sequential testing problem of k-out-of-n systems under general precedence constraints. A variation of this problem in electronic and electro mechanic systems is shown to be NP-complete [21]. A k-out-of-n system consists of n components which are either faulty or fault-free. This system is functional when at least k of the components are fault-free and it is not functional when at least $(n - k + 1)$ components are faulty. The system functionality depends only on the number of working and faulty components. Given the cost of testing each component and the prior probability of being fault-free for each component, the problem is to devise an optimal strategy that minimizes the total expected cost of finding the correct state of the system.

Variations of optimal test sequencing problem without precedence constraints have been extensively studied under various assumptions [2, 6]. On the other hand, in the existence of precedence constraints, there are only a few analytical results for the sequential testing problem. In [7], parallel chain precedence constraints was studied. In [13], an optimal algorithm was developed when precedence constraints satisfy some certain conditions. These conditions are given below:

- The precedence graph is a forest type precedence graph.
- Each tree in the forest is either an out-tree or in-tree.

As a special case of the result given by Garey in [13], an algorithm was developed for the series system when the precedence graph is a special forest in [7]. In these studies, there are not any computational results. The proposed algorithm in [7] can also be adapted for general k-out-of-n systems.

1.1 Contributions

The main purpose of this study is to minimize the expected cost of k-out-of-n sequential testing problem under general precedence constraints. The contributions of this study can be summarized as follows:

- We adapt the intersection algorithm (which is optimal for k-out-of-n systems when there are no precedence constraints) for the case of precedence constraints so that it is still possible to store the resulting strategy efficiently.
- We propose tabu search and simulated annealing algorithms to solve proposed problem.
- We conduct an extensive computational study to investigate the performances of the proposed methods.

1.2 Outline

The problem description and the literature review are presented in Chapter 2. We apply and improve intersection algorithm proposed by Ben-Dov [2] for our problem under precedence constraints in Chapter 3. Chapter 4 presents the proposed test sequencing problem with precedence constraints. We develop tabu search and simulated annealing algorithms for this test sequencing problem. We present numerical results in Chapter 5 to demonstrate the computational efficiency of the implemented heuristics and developed Intersection-Precedence Algorithm and to comparatively analyze all methods according to cost and time efficiency measures. Finally, in Chapter 6 we conclude and discuss future research directions.

CHAPTER 2

PROBLEM DESCRIPTION AND LITERATURE REVIEW

2.1 Problem Description

We consider a system that consists of n components whose functionalities are not known yet. The set of the individual components is denoted by N = $\{u_1$, u_2, ..., $u_n\}$. Each component of this set is functional or not and x_i describes the functionality of component i. x_i is 1 if the component i is functional, 0 if it is not functional.

$\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n})$ is a boolean vector describing the states of individual components where the i^{th} element of that vector shows the functionality of component i. The states of the components are not known by the decision maker, but the prior working probabilities of the components are known. We denote by p_i, the probability that component i functions. In order to determine the correct the state of the whole system, we need to test some of the components. The testing procedure terminates when the actual state of the system is determined. It is assumed that the individual components function or fail independent of each other. It is costly to test individual components (i.e. to learn the correct state of individual components). We denote by c_i the cost of testing component i. This could also correspond to the time required to test component i.

The whole system can be in either working state or failure state. In this particular study, we consider k-out-of-n systems, where the system is in working state if and only if at least k components are functioning. In other words, the testing procedure is terminated once we find k functioning components or $n - k + 1$ failing components. The k-out-of-n systems are the generalization of the 1-out-of-n (parallel) systems and n-out-of-n (serial) systems.In certain applications, it is not possible to test the components in any order. Due to physical or technological constraints, there can be precedence constraints among the components. This precedence relationship can naturally be described by an acyclic directed graph. The nodes of the graph correspond to the components of the system. An arc from node i to node j means that one can test component j only if component i is already tested.

A testing strategy S is a rule that specifies which component will be tested next, given the states of the inspected components. Each strategy will have a certain expected cost. An optimal inspection strategy is the one that has minimum total expected time or cost among all strategies.

The notation is summarized below:

N: set of nodes, N = $\{u_1, u_2, ..., u_n\}$

p_i: priori probability that u_i is working

q_i: priori probability that u_i is not working, q_i = (1-p_i)

c_i: testing cost of node u_i

While, testing strategies for the simple parallel and simple series systems are represented by permutations of components, in general, the strategies for k-out-of-n systems can be represented by a binary decision tree. An example binary decision tree of 3-out-of-5 system diagnosis procedure is given in figure 3.1.The nodes of the binary decision tree corresponds to the components to be tested. If the component is faulty, then the next component to be tested is the component corresponding to left child of the previously tested node. If the component is fault-free, then the next component to test is the right child. The leaf nodes are ”success” or ”fail” nodes that show the state of the system.

For instance, in the example in Figure 3.1, the first component to be tested is component 3. If component 3 functions then the next component to be tested is 2, otherwise the next component to be tested is 4. There is a unique path from the root to each leaf node. If the leaf node is success node, then there are k right arcs and less than $(n - k + 1)$ left arcs and for the fail nodes there are $(n - k + 1)$ left arcs and less than k right arcs on the path from the root to that leaf node. On each path from root to a leaf node we can observe the state of each component on this path and we can find the cost and probability of the path. The cost of a path can be calculated by summing up all the costs associated to the nodes on this path and the probability of a path can be calculated by product of p_i's for the working state components multiplied by the product of $1 - p_i$'s for the faulty components. In this framework, we can calculate the expected cost of a path by multiplying path cost with the path probability. By summing up all paths' expected cost, we will find the expected cost of decision tree or expected cost of testing strategy S. If we describe a strategy by an explicit binary decision tree, the size of this binary decision tree can be exponentially large in terms of the problem size, which is described by k, n, $\mathbf{C}$ and $\mathbf{P}$. So it would not be possible to store the solution strategy in a compact way. One can overcome this difficulty by describing an algorithm that outputs the next component to be tested given the results of the tests conducted so far. In this way, it is possible to use this algorithm to diagnose a system by running it until k functioning or $n - k + 1$ failing components have been determined. On the other hand, we will not be able to compute the expected cost of the strategy in this way in polynomial time. It turns out that, when there are no precedence constraints, it is possible to describe an optimal strategy in a compact way by using a data structure called Block-Walking Diagram. We explain the details of this data structure in Chapter 3. Unfortunately, when there are precedence constraints, to describe an optimal strategy in this manner is no longer possible. Since this method is optimal when there are no precedence case, we try to adapt the same logic of choosing the next component to inspect and using Block-Walking Diagram by hoping that this will produce satisfactory results.

Since it is very difficult to store a solution strategy as an explicit binary decision tree, due to the exponential growth in the size of the tree, it could be of interest to consider a subset of the strategies that are easy to represent and try to find a good solution among these strategies. An alternative is to consider permutation strategies, where the next component to test is the next component in the permutation. In this case, a strategy is just described by a permutation, but we still need to be able to compute the expected cost of such a strategy efficiently. It turns out that this is possible. We show how to compute the expected cost of a permutation strategy in Chapter 4 and propose tabu search and simulating annealing algorithms that find good permutation strategies.

2.1.1 An Example k-out-of-n System Under Precedence Constraints

A 3-out-of-5 system example is given below with data in Table 3.1. The precedence constraints are given in Table 2.2 by using $P(i, j)$'s and in Figure 2.5 by using arcs between components which has precedence relationship. $P(i,j) = 1$ means there is an arc from node i to j. We calculate the total expected testing cost of some strategies by using given data under precedence constraints.

i	u_1	u_2	u_3	u_4	u_5
p_i	0.95	0.9	0.7	0.82	0.6
c_i	2	2.5	2	4	3

Table 2.1: Data for an Example of 3-out-of-5 System

P(i , j)	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	1
3	0	1	0	1	0
4	1	0	0	0	0
5	0	0	0	0	0

Table 2.2: Precedence Constraints Data for an Example of 3-out-of-5 System

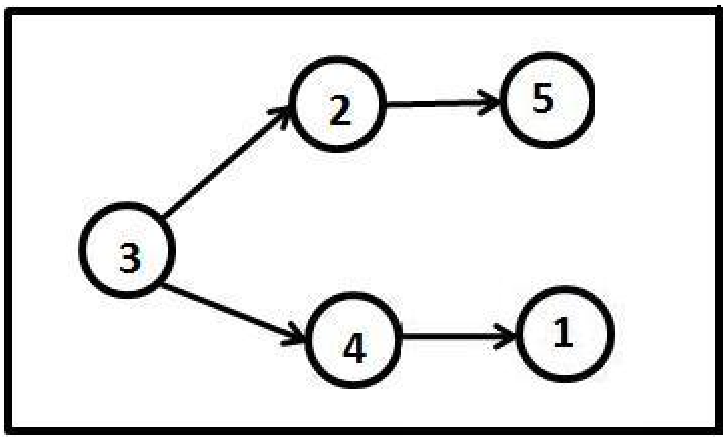


Figure 2.1: Precedence Constraints for Example of 3-out-of-5 System

In the first strategy, we consider a feasible permutation strategy according to precedence constraints which is (3-2-4-5-1). We start to test components according to this given order to determine the state of the whole system. 3-out-of-5 system is functional if at least 3 of the components are fault-free or the system is not functional if 3 of the components are faulty. In Figure 2.2, binary decision tree of this procedure is given. As can be seen in Figure 2.2, no matter what the state of already inspected components are, as long as we need to inspect a component, it is the next component in the permutation. The expected cost of given order can be calculated as:

TC = $1.c_3 + (q_3 + p_3).c_2 + [q_3.(p_2 + q_2) + p_3.(p_2 + q_2)].c_4 + [q_3.(q_2.p_4 + p_2.q_4 + p_2.p_4) + p_3.(q_2.p_4 + q_2.q_4 + p_2.q_4)].c_5 + [q_3.(q_2.p_4.p_5 + p_2.q_4.p_5 + p_2.p_4.q_5) + p_3.(q_2.q_4.p_5 + q_2.p_4.q_5 + p_2.q_4.q_5)].c_1$

For this example total cost is calculated 10.35072.

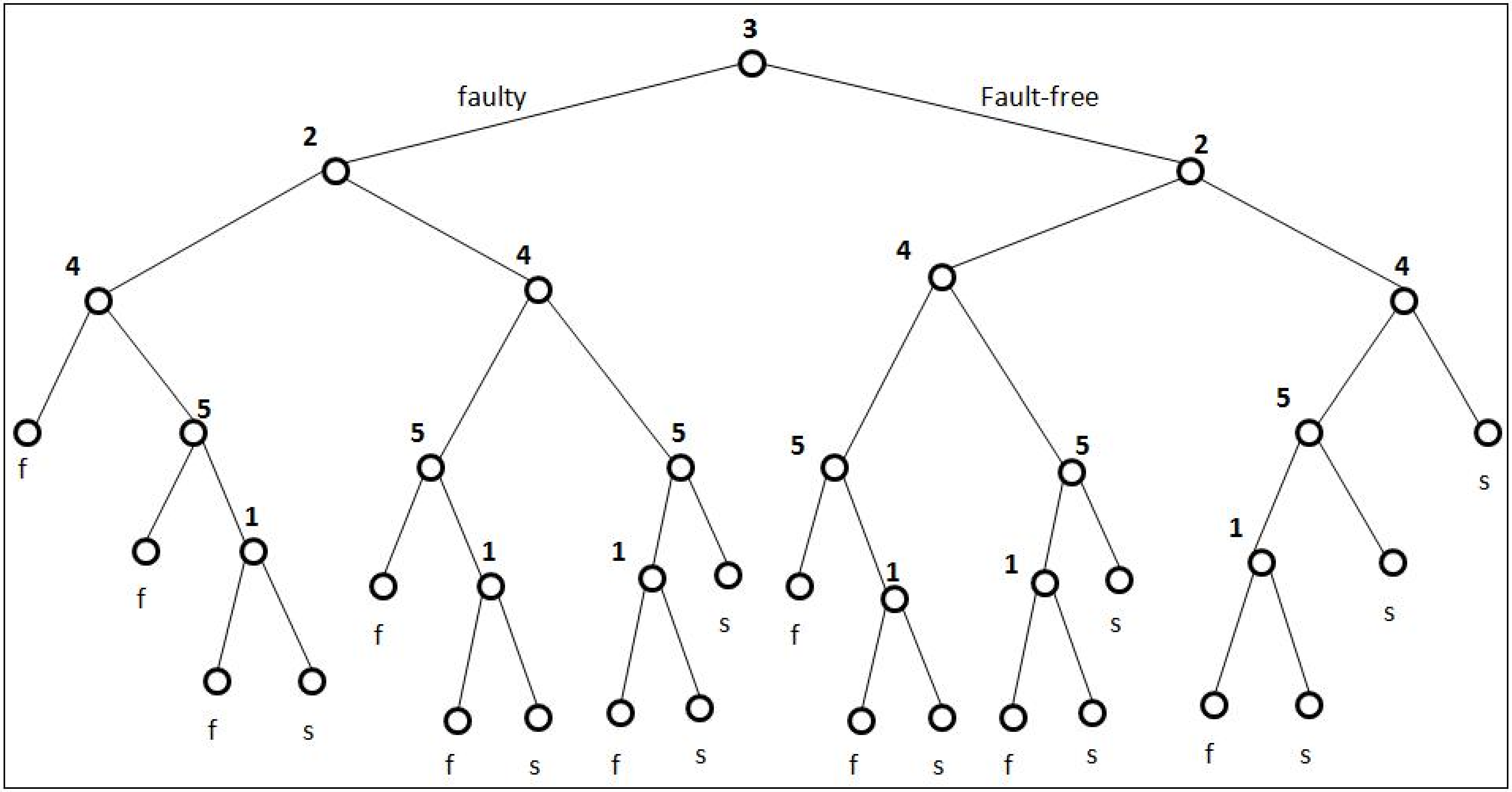


Figure 2.2: An Example Procedure for (3-2-4-5-1) Order

When we have a permutation strategy, we can calculate total expected cost for given testing order computationally easily. The calculation steps can be represented by using a matrix which is given below:

$$\begin{array}{llll} C_{(0,0)} & C_{(0,1)} & ... & C_{(0,k)} \\ C_{(1,0)} & C_{(1,1)} & ... & C_{(1,k)} \\ ... & ... & ... & ... \\ C_{(n-k+1,0)} & C_{(n-k+1,1)} & ... & C_{(n-k+1,k)} \end{array}$$

Then we can calculate the expected testing cost by using the following recursion in a bottom-up fashion:

- C(i,j)= $C(a_{i+j+1}) + p(a_{i+j+1}).C(i, j+1) + [(1 - p(a_{i+j+1})).C(i+1, j)$ if $j < k$ and $i < (n-k+1)$
- C(i,j) = 0 if $j = k$ and $i = (n-k+1)$

where

- a_i; i^{th} component in the given permutation $a = a_1, a_2, ..., a_n$
- $C(a_i)$: Cost of testing a_i
- $p(a_i)$: Prior success probability of a_i

When we have a permutation strategy, we can compute the total expected testing cost in polynomial time. For k-out-of-n problems, the optimal testing strategy is not always a permutation strategy. We mention about binary tree representation. We calculate the objective function value by using a permutation testing strategy for the example of 3-out-of-5 system under precedence constraints. An example binary decision tree strategy is given in Figure 2.3 for this problem whose data is given in Table 3.1.

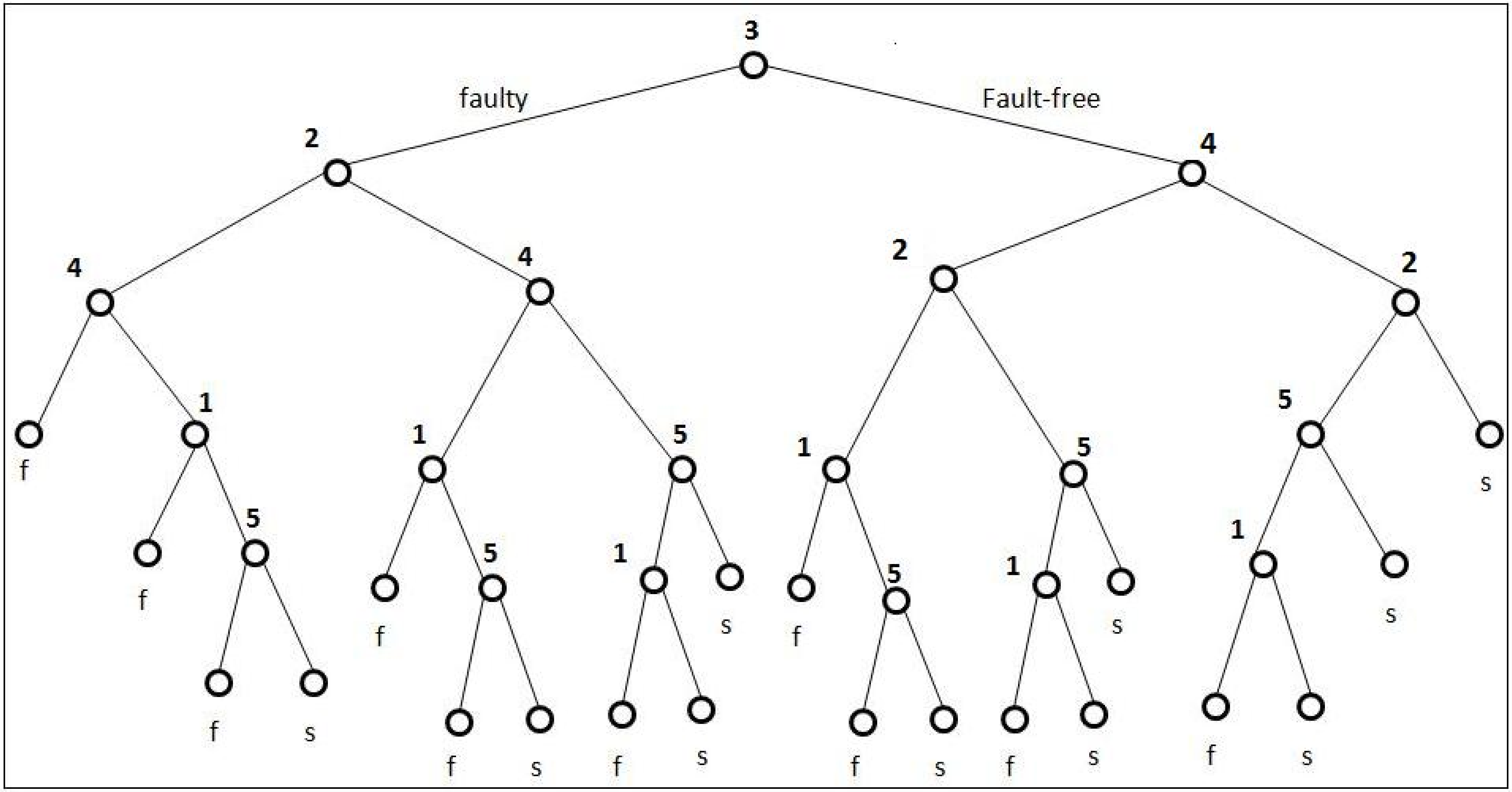


Figure 2.3: An Example Binary Tree Procedure 3-out-of-5 System

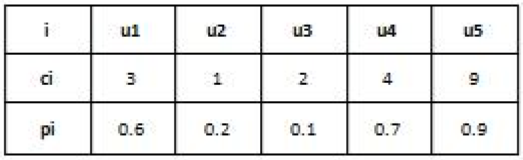
The expected cost of the binary tree example given in Figure 2.3 is calculated as follows:

TC = $1.c_3 + [p_3 + (q_3.p_2) + (q_3.q_2)].c_4 + [q_3 + (p_3.q_4) + (p_3.p_4)].c_2 + [(q_3.q_2.p_4) + (q_3.p_2.q_4) + (p_3.q_4.q_2) + (q_3.p_2.p_4.q_5) + (p_3.q_4.p_2.q_5) + (p_3.p_4.q_2.q_5)].c_1 + [(q_3.p_2.p_4) + (p_3.q_4.p_2) + (p_3.p_2.q_4) + (q_3.q_4.p_2.p_1) + (q_3.p_4.q_2.p_1) + (p_3.q_2.q_4.p_1)].c_5$

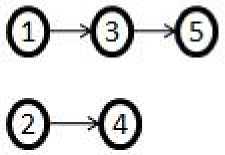
For this example total cost is calculated as 10.57449. For small size problems, it is computationally easy to calculate the total expected cost, when the testing strategy is not a permutation strategy. Unfortunately, when n and k get larger, it is not possible to represent binary decision tree and to calculate the expected cost of the tree. In Chapter 3 we propose an algorithm to find testing procedure and a matrix representation to calculate the total expected testing cost. To compare these results, then in Chapter 4 we start with a good feasible solution and apply a search procedure based on some well-known heuristics and find better permutation solutions.

2.2 Literature Review

Chiu et al. [7] study k-out-of-n sequential testing problem with special precedence constraints referred to parallel chain type precedence constraints. This precedence type consists disjoint subsets of components set. These disjoint subsets have the precedence constraints only within each of themselves. This precedence type constraints means that all items can be partitioned into subsets and each block has a precedence constraints which is defined by unique inspection order. Chui et al give an example of this type of precedence constraints for a series system (inspecting the components until a failure is found or all of the components have been inspected) which is given in Figure 2.4:



i	u1	u2	u3	u4	u5
ci	3	1	2	4	9
pi	0.6	0.2	0.1	0.7	0.9



$$\begin{tikzcd} 1 \arrow[r] & 3 \arrow[r] & 5 \\ 2 \arrow[r] & 4 & \end{tikzcd}$$

Figure 2.4: A series system with parallel chain precedence constraint

They determined testing cost, testing priori probability, testing states by using the Block Walking Algorithm which is developed by Chang, Chi and Fuchs [6]. The following notation is used in their algorithms, also, they determined the working probability, testing cost and some ratios which will be used in solution methodology, of a set $I = (i_1, i_2, ..., i_j)$ such as:

- Working probability of the set I: $P(I) = p_{i_1}.p_{i_2}....p_{i_j}$
- Testing cost of the set I with respect to series structure: $C(I) = c_{i_1} + p_{i_1}.c_{i_2} + ... + p_{i_1}....p_{i_{j-1}}.c_{i_j}$
- Failure probability of the set I: $Q(I) = q_{i_1}.q_{i_2}....q_{i_j}$
- Testing cost of the set I with respect to parallel structure (inspecting the components until a success is found or all of the components have been inspected): $D(I) = c_{i_1} + q_{i_1}.c_{i_2} + ... + q_{i_1}....q_{i_{j-1}}.c_{i_j}$
- R-ratio: $R(I) = \frac{C(I)}{1-P(I)}$
- S-ratio: $S(I) = \frac{D(I)}{1-Q(I)}$

They proved theorems that give an optimal testing strategy for series and parallel systems by using R-ratio and S-ratio which are given below:

- For a series system problem with a parallel-chain precedence constraints, if the blocks are arranged in order of increasing R-ratio according to the precedence constraints; then the resulting sequence is optimal.
- For a parallel system problem with a parallel-chain precedence constraints, if the blocks are arranged in order of increasing S-ratio according to the precedence constraints; then the resulting sequence is optimal.

Using the results for series and parallel systems they developed the optimal testing strategy for $k - out - of - n$ systems. If an inspection procedure satisfies two conditions which are given below, then it is optimal:

- Condition 1: All of the blocks in the sequence has a S-ratio which is less than the blocks before them comes from a different chain.
- All of the blocks in the sequence has a R-ratio which is less than the blocks before them comes from a different chain.

In summary, Chui et al. developed more general results by using the study of Ben-Dov [2]. Also, they pointed out that the optimal inspection rule for the general k-out-of-n systems under parallel-chain precedence constraints is not yet known.

Also, Garey in [13] studied simple series (parallel) systems sequential testing problem with forest type precedence constraints where in each precedence graph either no component has more than one immediate predecessor, or no task in that component has more than one immediate successor. An example of forest type precedence constraints is given in Figure 2.5.

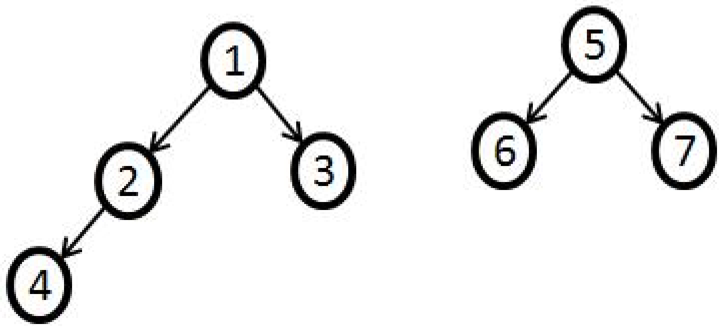


Figure 2.5: An example of forest type precedence constraints

Garey [13] provided some reduction rules like Chui et al. [7] that turn the precedence graph into a graph without any arcs. Essentially, the reduction rules combine certain nodes or delete some arcs in the precedence graph. Then they used their reduction rules to find a series and parallel systems under forest type precedence constraints.

Ünlüyurt [24] described and analyzed a framework for sequential testing problem. In this review paper, different related applications are described such as distributed computing, artificial intelligence, manufacturing and telecommunications. The mathematical framework, variations and extensions of this problem are given. Series, parallel, k-out-of-n, Series-Parallel systems are mentioned and strategies, binary decision trees, solution methodologies about these problem types are reviewed. Catay et al [5] develop an ant colony based algorithm for testing series (parallel) systems under precedence constraints.

Tanrıverdi [23] studied k-out-of$n$ testing problem under forest type precedence constraints. In this study, optimal inspection strategies are obtained for series and parallel systems. Then these strategies are adapted for k-out-of-n systems.

CHAPTER 3

SOLUTION METHODOLOGY UNDER PRECEDENCE CONSTRAINTS

In the Chapter 2, we mentioned some strategies that are optimal for the sequential testing problem of k-out-of-n system without precedence constraints, namely the Intersection Algorithm. In this section, we develop algorithms for the general k-out-of-n system problem by using Intersection Algorithm. We adapt the Intersection Algorithm proposed by [2] for the case of precedence constraints. Also, we describe how to store this strategy by using Block-Walking representation. In this chapter, first we describe the Intersection Algorithm and Block-Walking representation in detail. Then we describe how we adapt this algorithm when general precedence constraints exist.

3.1 Intersection Algorithm and Block Walking Representation without Precedence Constraints

For k-out-of-n structures without any precedence constraints, an optimal diagnosis procedure is proposed by Ben-Dov [2]. This diagnosis procedure can also be described by a binary decision tree. Each leaf node of this binary decision tree shows testing result. If a node is tested and its result is success, then the right subtree is taken else its result is failure, then the left subtree is taken. The objective is to find the diagnosis procedure which gives the minimum expected testing cost.

Firstly, two sets are defined as U and V. The set $U_i = \{\tau(j)|1 \leq j \leq i\}$ is utilized by the permutation which is labeled as,

$$\frac{c_1}{p_1} \leq \frac{c_2}{p_2} \leq ... \leq \frac{c_n}{p_n}$$

The set $V_i = \{\pi(j)|1 \leq j \leq i\}$ consists an order of nodes which is utilized by a permutation so that,

$$\frac{c_\pi(1)}{p_\pi(1)} \leq \frac{c_\pi(2)}{p_\pi(2)} \leq ... \leq \frac{c_\pi(n)}{p_\pi(n)}$$

Firstly, intersection algorithm that is proposed by Ben-Dov [2], takes the intersection of the sets U_k and V_{n-k+1}. To obtain an optimal strategy for $k - out - of - n$ structure without any precedence constraint, the tested unit is any of the elements of this intersection. If the first item is faulty, then the problem becomes a $k - out - of - (n-1)$ system and if the first item is fault-free the problem becomes $(k-1) - out - of - (n-1)$ system. And the intersection procedure is implemented for these new systems. This procedure is stopped when the correct state for the whole system is found.

Chang, Chi and Fuchs [6] recompute and save the optimal diagnosis procedure by a compact representation by using Block-Walking Algorithm. They represent the binary decision tree by this block-walking representation in $O(n^2)$ space.

Notations and definitions for this representation are given below:

$TU(v)$: Tested unit set, for any vertex $v4$ in a binary decision tree, it is the set of units tested along the path from the root to v, including v.

$TS(v)$: Test state, for any vertex v in a binary decision tree, it is defined to be an ordered pair (i,j), where i and j are the number of fault-free and faulty units tested along the path from root to v, excluding v.

G: Set of intermediate states, $G = \{(i,j)|0 \leq i \leq k-1, 0 \leq j \leq n-k\}$.

S: Set of success states, $S = \{(k,j)|0 \leq j \leq n-k\}$.

F: Set of failure states, $F = \{(i, n-k+1)|0 \leq i \leq k-1\}$.

δ_s: $G \rightarrow N$ it indicates which unit to test if the last test has succeeded.

δ_f: $G \rightarrow N$ it indicates which unit to test if the last test has failed.

$\delta_s(i,j) = u_l$ and $\delta_f(i,j) = u_m$ mean that, in state (i,j), it u_l will be tested if last test succeeded and u_m will be tested if last test failed. They have assumed that the test before state (0,0) as succeeded. Also, they have proven that block-walking representation can be represented if the unit which has the smallest subscript (SS) from the intersection set is chosen.

3.2 Intersection Algorithm without Precedence Constraints

A k-out-of-n structure with the yield probability $p_1, p_2, , p_n$ and the testing cost $c_1, c_2, , c_n$. G,S and F can be constructed easily and δ_s and δ_f can be constructed by the following steps.

Algorithm 3.1: Intersection Algorithm

```
Step 1:
$tmp := U_k \bigcap V_n - k + 1;$
$\delta_s(0,0) := SS(tmp);$

Step 2:
for $i = 1$ to $k - 1$ do
    $tmp := tmp \bigcup \{u_{\pi(n-k+1+i)}\} - \{\delta_s(i-1,0)\};$
    $\delta_s(i,0) := SS(tmp);$
end for

Step 3:
for $i = 0$ to $k - 1$ do
    $tmp := V_{n-k+1+i} - \{\delta_s(j,0) | 0 \leq j \leq i\};$
    Sort tmp into ascending subscript order;
    for $j = 1$ to $n - k$ do
        $\delta_f(i,j)$:= the jth unit of tmp;
    end for
end for

Step 4:
for $i = 1$ to $n - k$ do
    for $j = 1$ to $k - 1$ do
        if $\delta_f(i,j) = \delta_f(i-1,j)$ then
            $\delta_s(i,j) = \delta_s(i,j-1);$
        else
            $\delta_s(i,j) = \delta_f(i,j);$
        end if
    end for
end for

End of Algorithm
```

The implementation steps of this algorithm is given by using a numerical example. In Table 3.1, success probability and testing cost of a 3-out-of-5 example is given below. Also, in Figure 3.1 and Figure 3.2, the binary decision tree and block-walking representation of this example are given.

i	u_1	u_2	u_3	u_4	u_5
p_i	0.95	0.9	0.7	0.82	0.6
c_i	2	2.5	2	4	3

Table 3.1: Example of 3-out-of-5 System

By using the information which is given on the Table 1, U and V sets are defined as $U = \{1, 2, 3, 4, 5\}$ and $V = \{3, 5, 4, 2, 1\}$. The implementation steps of intersection algorithm is given below:

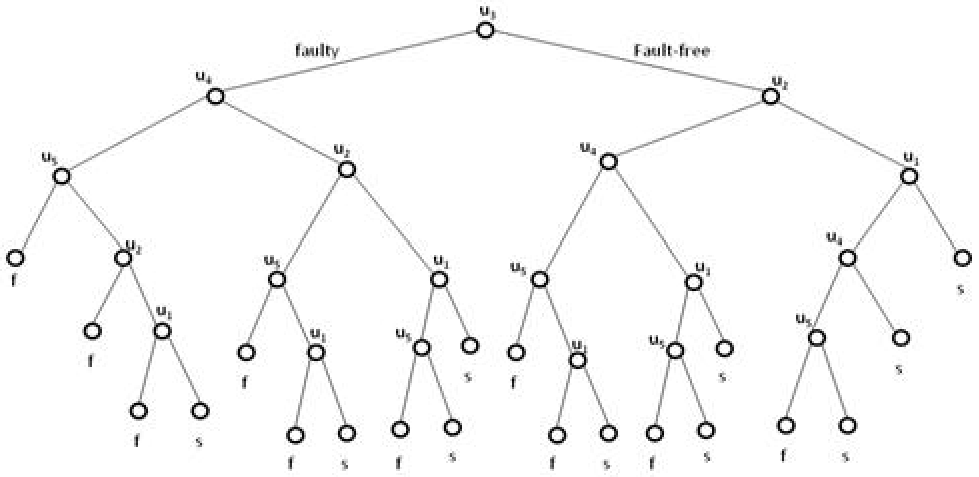


Figure 3.1: Example diagnosis procedure for 3-out-of-5 System

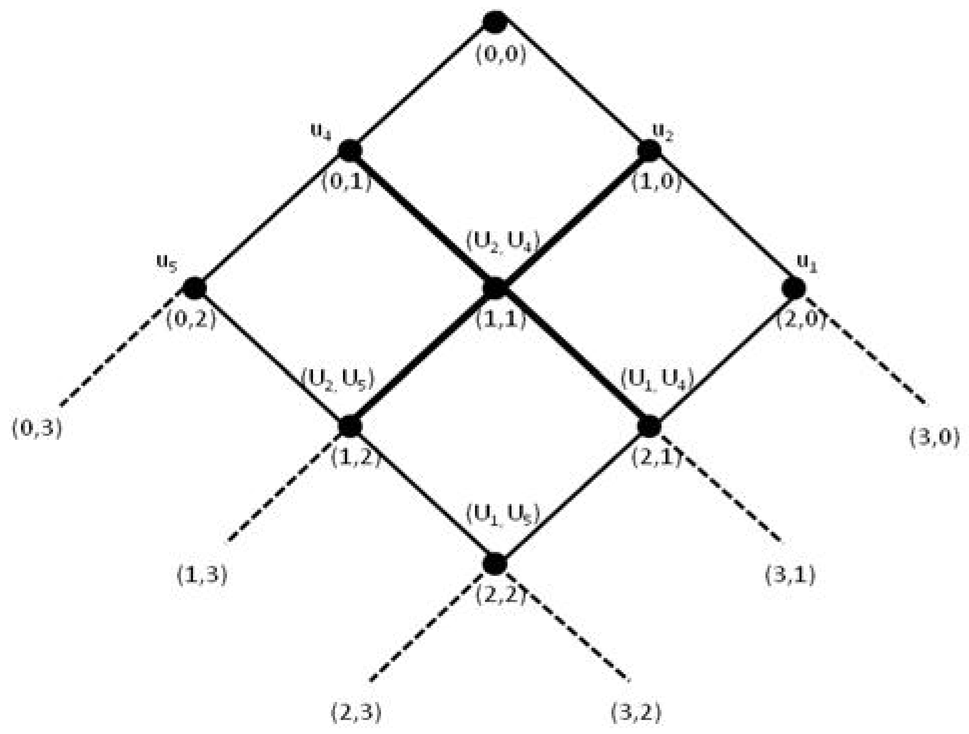


Figure 3.2: A diagnosis procedure defined by the block-walking representation

The total expected cost can be calculated by using this block-walking representation. Calculating the whole expected cost, all of the grid points are considered. A matrix representation by using this block-walking representation is given below.

Algorithm 3.2: Intersection Algorithm - Example

```
Step 1:
$tmp := U_3 = \{1,2,3\} \bigcap V_{5-3+1} = \{3,5,4\} = \{3\}$ so $\delta_s(0,0) := 3$;
Step 2:
for $i = 1$ to $3-1$ do
  $i = 1 \rightarrow tmp := \{3\} \bigcup \{2\} - \{\delta_s(0,0)\} = \{2\}$;
  $\delta_s(1,0) := 2$;
  $i = 2 \rightarrow tmp := \{2\} \bigcup \{1\} - \{\delta_s(1,0)\} = \{1\}$;
  $\delta_s(2,0) := 1$;
end for
Step 3:
for $i = 0$ to $3-1$ do
  $i = 0 \rightarrow tmp := \{3,5,4\} - \{\delta_s(0,0)\} = \{5,4\}$;
  Sort tmp into ascending subscript order, so $tmp := \{4,5\}$;
  for $j = 1$ to $5-3$ do
    $j = 1 \rightarrow \delta_f(0,1) = 4$;
    $j = 2 \rightarrow \delta_f(0,2) = 5$;
  end for
  $i = 1 \rightarrow tmp := \{3,5,4,2\} - \{\delta_s(0,0) and \delta_s(1,0)\} = \{5,4\}$;
  Sort tmp into ascending subscript order, so $tmp := \{4,5\}$;
  for $j = 1$ to $5-3$ do
    $j = 1 \rightarrow \delta_f(1,1) = 4$;
    $j = 2 \rightarrow \delta_f(1,2) = 5$;
  end for
  $i = 2 \rightarrow tmp := \{3,5,4,2,1\} - \{\delta_s(0,0), \delta_s(1,0) and \delta_s(2,0)\} = \{4,5\}$;
  Sort tmp into ascending subscript order, so $tmp := \{4,5\}$;
  for $j = 1$ to $5-3$ do
    $j = 1 \rightarrow \delta_f(2,1) = 4$;
    $j = 2 \rightarrow \delta_f(2,2) = 5$;
  end for
end for
Step 4:
for $i = 1$ to $5-3$ do
  $i = 1 \rightarrow$
  for $j = 1$ to $3-1$ do
    $j = 1 \rightarrow (\delta_f(1,1) = \delta_f(0,1))$
    $\delta_s(1,1) := \delta_s(1,0) = 2$;
    $j = 2 \rightarrow (\delta_f(1,2) = \delta_f(0,2))$
    $\delta_s(1,2) := \delta_s(1,1) = 2$;
  end for
  $i = 2 \rightarrow$
  for $j = 1$ to $3-1$ do
    $j = 1 \rightarrow (\delta_f(2,1) = \delta_f(1,1))$
    $\delta_s(2,1) := \delta_s(2,0) = 1$;
    $j = 2 \rightarrow (\delta_f(2,2) = \delta_f(1,2))$
    $\delta_s(2,2) := \delta_s(2,1) = 1$;
  end for
end for
End of Algorithm
```

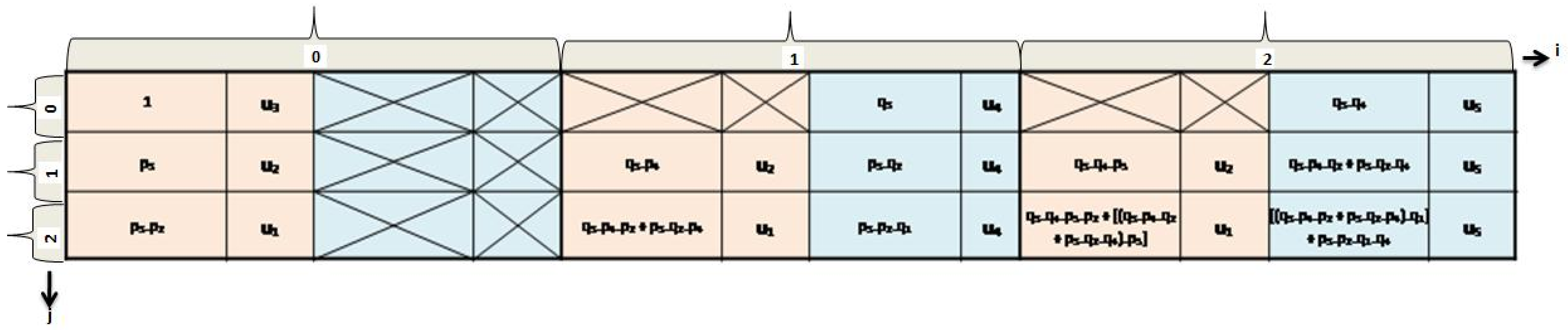

0
1
2
i
j
1
u_3
p_3
u_2
$p_3.p_2$
u_1
$q_3.p_4$
$q_3.p_4.p_2 + p_3.q_2.p_4$
q_3
u_4
$p_3.q_2$
$p_3.p_2.q_1$
$q_3.q_4.p_3$
$q_3.q_4.p_3.p_2 + [(q_3.p_4.q_2 + p_3.q_2.q_4).p_3]$
$q_3.q_4$
u_5
$q_3.p_4.q_2 + p_3.q_2.q_4$
$[(q_3.p_4.p_2 + p_3.q_2.p_4).q_1] + p_3.p_2.q_1.q_4$

Figure 3.3: An example diagnosis procedure defined by matrix representation

Representing the matrix form, for all states (i,j) there are four columns:

- For state (i,j), first column represents that the probability when you reached this state after you test a node and it was fault-free in state (i-1,j).
- Second column represents that δ_s(i,j).
- For state (i,j), third column represents that the probability when you reached this state after you test a node and it was faulty in state (i,j-1).
- Fourth column represents that δ_f(i,j).

By using this information, it is easy to calculate the root probability for each state. For example, for state (2,2), it is possible to reach this state after testing a node in state (1,2) and test result is success or after testing a node in state (2,1) and test result is fail. So, by using the root from (1,2), we can reach the column 1 (because the result is success), and the root probability is $q_3.q_4.p_5.p_2 + [(q_3.p_4.q_2 + p_3.q_2.q_4).p_5]$. Because we test u_2 and u_5 in (1,2) state and we know the result is success, so the success probability of those nodes multiplied the probability which is found from state (0,0) to the current state. By using the root from (2,1) $[(q_3.p_4.p_2 + p_3.q_2.p_4).q_1] + p_3.p_2.q_1.q_4$. Because in state (2,1), we test u_1 and u_4 and the result is failure, and we use the failure probability of those nodes. After completing the whole representation, we obtain the expected cost by multiplying the probabilities by testing units.

The expected cost of above example:

TC = $1.c_3 + q_3.c_4 + q_3.q_4.c_5 + p_3.c_2 + q_3.p_4.c_2 + p_3.q_2.c_4 + q_3.q_4.p_5.c_2 + [(q_3.p_4.q_2 + p_3.q_2.q_4).c_5] + p_3.p_2.c_1 + [(q_3.p_4.p_2 + p_3.q_2.p_4).c_1] + p_3.p_2.q_1.c_4 + [(q_3.q_4.p_5.p_2 + (q_3.p_4.q_2 + p_3.q_2.q_4).p_5)].c_1 + [((q_3.p_4.p_2 + p_3.q_2.p_4).q_1) + p_3.p_2.q_1.q_4].c_5$

For this example total cost is calculated 8.30499.

3.3 Intersection-Precedence Algorithm

Intersection algorithm is applied for k-out-of-n structure without precedence constraints in the previous section. In this section, to find the better solutions for this problem, Intersection-Precedence Algorithm (INT-PREC) is developed by using some rules of Intersection Algorithm. Intersection Algorithm gives an optimal strategy if there is no precedence relationship.

Under precedence constraints, it is more difficult to select which following node is tested. Every success and failure states, it is needed to control which nodes can be tested because of previous success and failure states and precedence relationship.

Notations and definitions for Intersection Algorithm are used for this algorithm. Also, there are some other notations and definitions which are given below:

tmp: Temporary set defined by the algorithm.

ind: Item has a highest number of successors of a set.

AI: Available item set according to precedence constraints in this stage.

$tmp - AI$: Intersection set of tmp and AI.

t: Tested units before reaching that state.

SS: Smallest subscript.

A k-out-of-n structure with precedence constraint the yield probability $p_1, p_2, , p_n$ and the testing cost $c_1, c_2, , c_n$. G,S and F can be constructed easily and δ_s and δ_f can be constructed by the following steps.

This algorithm gives always feasible results, because in all of the steps, it controls whether there are available items. If a testing strategy is shown by a binary decision tree, we check the available items by following the previous failure and success states. It is time consuming to decide which nodes are available to test in the present state. If a strategy is obtained by Intersection Algorithm, after some iterations, all of the available items are used and any feasible solution is not found. So, in this algorithm, to select following node, the number of successors of the item is another criteria. Also, this algorithm gives us a chance to select following component according to existing state of the system. So, this algorithm gives not only a testing sequence but also a binary tree of the testing components in all of the states. We will comparatively analyze the results of this algorithm and applied heuristics based on objective function value and run time. When we solve a small size of problem, we use the same data which is given in Table 3.1. So, $U = \{1, 2, 3, 4, 5\}$ and $V = \{3, 5, 4, 2, 1\}$. But we add new precedence relationship between {(2,4), (3,4), (4,5), (4,1)}. (i,j) means j can be tested if and only if i is tested before. For this particular example, we generate all of the possible policies in Excel, then check if they are feasible, calculate their objective function values and find the minimum one. The value is 9.704 is same as the value which we find by this algorithm.

Algorithm 3.3: Intersection-Precedence Algorithm

```
Step 1:
$tmp := U_k \bigcap V_n - k + 1$;
$AI$:= Find the available items according to precedence constraint;
$tmp - AI := tmp \bigcap AI$;
if tmp-AI is not an empty set and $ind(tmp - AI) \neq \emptyset$ then
  $\delta_s(0,0) := r$;
else
  Sort AI into order of $ind$ values, $\delta_s(0,0) := SS(AI)$;
end if
Step 2:
for $i = 1$ to $k - 1$ do
  $tmp := tmp \bigcup \{u_{\pi(n-k+1+i)}\}$;
  $AI := N - \{\delta_s(i-1,0)\}$
  $tmp - AI := tmp \bigcap AI$;
  if tmp-AI is not an empty set and $ind(tmp - AI) \neq \emptyset$ then
    $\delta_s(i,0) := r$;
  else
    Sort AI into order of $ind$ values, $\delta_s(i,0) := SS(AI)$;
  end if
end for
Step 3:
for $i = 0$ to $k - 1$ do
  for $j = 1$ to $n - k$ do
    $AI := N - \{\delta_s(m,0) | 0 \leq m \leq i\} - \{\delta_f(m,t) | 0 \leq t < jand0 < m < i\}$;
    if $i > 1$ and $\delta_f(i-1,j) \epsilon AI$ then
      $\delta_f(i,j) := \delta_f(i-1,j)$
    else
      $tmp := tmp \bigcup V_{n-k+1+i}$;
      $tmp - AI := tmp \bigcap AI$;
      if tmp-AI is not an empty set and $ind(tmp - AI) \neq \emptyset$ then
        $\delta_f(i,j) := r$
      else
        Sort AI into order of $ind$ values, then $\delta_f(i,j) := SS(AI)$;
      end if
    end if
  end for
end for
Step 4:
for $i = 1$ to $n - k$ do
  for $j = 1$ to $k - 1$ do
    if $\delta_f(i,j) = \delta_f(i-1,j)$ then
      $\delta_s(i,j) = \delta_s(i,j-1)$;
    else
      $\delta_s(i,j) = \delta_f(i,j)$;
    end if
  end for
end for
End of Algorithm
```

CHAPTER 4

HEURISTICS

In this chapter, we propose a tabu search (TS) and simulated annealing (SA) algorithm to find good permutation strategies for the sequential testing problem for k-out-of$n$ systems under general precedence constraints. As mentioned before, we focus on the permutation strategies since we can compute their expected costs efficiently. Both TS and SA require an initial solution to start with. First, we describe how we find a good initial solution, that we will use as a starting solution for both algorithms. Then we describe the TS and SA algorithms in detail, in terms of implementation details and parameters.

4.1 Initial Solution

We are interested in finding permutation strategies that respect the precedence constraints. We order the nodes as a sequence such that for every arc $(i, j) \in precedencelist$, order(i)¡ order(j). We construct feasible permutations by using different merit values and compare the results in order to find out which merit values perform the best. We implement the generic algorithm performed on 4.1. At each step, the available nodes are those nodes with indegree 0 or the nodes that can be tested immediately, i.e. the nodes for which all predecessors are already tested.

Selection Merit Value

Random Selection: While there are available nodes to use for order, the next node is chosen randomly among available nodes.

Increasing Order of Cost/Priori Success Probability: At each step we choose the next component as the component that has the minimum cost/probability of functioning. Let us

Algorithm 4.1: Initial Solution Algorithm

```
for all $i \in N$ do
    in degree(i):=0;
end for
for all $(i,j) \in A$ do
    in degree(j):=indegree(j)+1;
end for
$List := \varnothing$;
$next := 0$;
for all $i \in N$ do
    if $indegree(i) = 0$ then
        $List := List \bigcup \{i\}$;
    end if
end for
while $List \neq \varnothing$ do
    rank order nodes in List according to chosen merit value;
    select first node $i$ of List and delete it;
    $next := next + 1$;
    $order(i) := next$;
    for all $(i,j) \in A(i)$ do
        $indegree(j) := indegree(j) - 1$;
        if in degree(j)=0 then
            $List := List \bigcup \{j\}$;
        end if
    end for
end while
if next¡n then
    the precedence constraints give a directed cycle;
else
    the network is acyclic and the array order gives a suitable order of nodes;
end if
End of Algorithm
```

note that this sequence is optimal for 1-out-of-n (parallel) systems without any precedence constraints.

Increasing Order of Cost/Priori Failure Probability: At each step we choose the next component as the component that has the minimum cost/probability of failing. Let us note that this sequence is optimal for n-out-of-n (parallel) systems without any precedence constraints.

Increasing Order of Cost/[(Priori Failure Probability)(Priori Success Probability)]: At each step we choose the next component as the component that has the minimum cost/(probability of failing*probability of functioning). One may consider this as a trade-off between the two latter merit values

We try all of these criteria to find out which one gives better initial solution. Different selection methodologies are used for different values of k. These results are given in Chapter 5.

4.2 Tabu Search Algorithm

Tabu search is a meta-heuristic method developed by Glover [14, 15] and has since been widely used to solve combinatorial optimization problems in the field of scheduling, routing, facility design, and so on. We refer the interested reader to the book by Glover and Laguna [16] and the references therein.

The main motivation of TS heuristic is to enable the search process to escape the trap of the local optimal solutions. In order to achieve this, it allows climbing moves when no neighboring solution improves the previous best solution. Besides, unlike other search techniques, TS avoids examining previously explored regions recurrently by keeping a tabu list. Tabu list includes the solutions that have been considered in the short run. This list forbids some moves to avoid returning to the previous solution unless they satisfy some aspiration criterion.

The general flow of a TS heuristic can be described as follows: The algorithm starts with an initial solution. At each iteration, we evaluate neighbor solutions and select the best solution in the neighborhood of the current solution until a termination criterion is met. Note that if this best solution is obtained as a result of a tabu move, we check whether or not the aspiration condition is satisfied. The aspiration condition describes a favorable circumstance under which even a tabu move is allowed. After selecting the new solution, we set the selected solution as the current solution and update the tabu list. If the selected solution improves the best solution so far we also update the best solution.

We have already discussed the process of finding an initial feasible solution. Next we describe how we implement other components of the TS algorithm, including neighborhood strategy, solution evaluation, tabu list and aspiration condition and termination criteria.

4.2.1 Neighborhood Strategy

We use a neighborhood move that is widely used in the literature, described as follows:

- Swap(i,j): Change the orders that node i and j in the permutation strategy

The swap function is used to evaluate all neighbor solutions obtained by the swap moves and select the best neighbor solution. In our TS implementation, at each iteration we apply the swap function. Then, we calculate the objection function value of that neighbor solution. We also check the number of successive iterations without any improvement in the overall best solution. If this number exceeds 50, we terminate the algorithm.

4.2.2 Solution Evaluation

To force the search, we allow infeasible moves with respect to "precedence constraints". If the precedence constraints are violated, the objective function is modified. To make this modification, we add a penalty function $\beta P(\mathbf{x})$. Here $P(\mathbf{x})$ is the total number of violated precedence relations calculated according to the ordered sequence for a decision vector $\mathbf{x}$ and β is the penalty coefficient with an initial value of 1. If the assignment is feasible for the sequence then $P(\mathbf{x})$ is equal to zero. Every 5 iterations the penalty coefficient β is divided by 2 if all 5 previous solutions were feasible or multiplied by 2 if all were infeasible.

4.2.3 Tabu List and Aspiration Condition

TS algorithm determines a tabu list to have a short term memory. The tabu list includes some tabu moves that means these moves are forbidden to apply. We cannot consider these moves unless they satisfy some aspiration condition.

It is also an important decision to determine the tabu list size. As the list size increases we may not identify the local optimal solutions, also if it is smaller, it may cause to reach to the previously discovered local optimal solutions. In our implementation, we define the tabu list size based on the number of k values.

If the selected move is in the tabu list, then in order to accept this move we should check whether the aspiration condition is satisfied. If this tabu move leads to a solution that has an objective function value strictly better than the best solution so far, then the aspiration condition is satisfied and this tabu move is accepted.

4.2.4 Termination Criteria

We terminate the search algorithm if the maximum computation time criteria is met. In our implementation, the maximum computation time ($timelim$) is defined as 500 seconds. We select a time limit as larger as our computed time for all data set. For larger n values, the algorithm is typically terminated because of time limit.

We conduct a computational study to test the efficiency and effectiveness of the proposed tabu search algorithms and present the corresponding numerical results in the related chapter.

4.3 Simulated Annealing Algorithm Heuristic

Simulated Annealing is a widely used meta-heuristic methodology that compose a search process to escape from a local optimum [20]. The approach used in this methodology is to focus on searching the global optimum. This can be found in anywhere in the feasible region, the early emphasis is to take steps in random directions.

The general flow of SA heuristic, at each iteration search process moves from the current trial solution to an immediate neighbor in the local neighborhood of this solution. To define how an immediate neighbor is selected to be the next trial solution is different from TS. Selection rule and notations are given below:

- Z_c= objective function value for the current trial solution,
- Z_n= objective function value for the current candidate to be the next trial solution,
- T= a parameter that measures the tendency to accept the current candidate to be the next trial solution if this candidate is not an improvement on the current trial solution.

Since our problem is a minimization problem, selecting the next trial solution from among all candidate alternatives is performed according to move selection rule for minimization problems of the simulated annealing algorithm . This rule is given below:

- If $Z_n \leq Z_c \Rightarrow$ This candidate is always accepted.
- If $Z_n > Z_c \Rightarrow$ This candidate is accepted with probability: $Probability\{acceptance\} = e^x$ where $x = \frac{Z_c - Z_n}{T}$

  Basic simulated annealing algorithm outline can be described as follows:

- **Initialization:** It is started with an initial feasible solution.
- **Iteration:** Next trial solution is selected and if there is no suitable next trial solution, the algorithm is terminated.
- **Temperature Schedule Check:** Simulated Annealing Algorithm is started with an initial temperature (T) value. When the desired number of iterations have been performed at the current temperature value, temperature is decreased to any other temperature value by using the schedule. Then solution methodology is resumed performing iterations by using this new temperature value.
- **Stopping Rule:** When the desired number of iterations have been performed according to every determined T values in schedule or when none of the current trial solution improves the best trial solution.

4.3.1 Neighborhood Strategy

We used the move selection rule to select the next trial solution. Select two nodes from feasible trial order and swap them by considering the precedence constraints (Enumeration).

4.3.2 Checking the Temperature Schedule

When the possible number of iterations have been reached at the initial value of T (0.2 times objective function value of initial feasible solution), decrease T (0.2 times previous temperature value) to the next value in the temperature schedule and perform iterations by using this new value. The computational results of different T values is given in Chapter 5.

4.3.3 Termination Criteria

When possible number of iterations has been performed at the smallest value of T (T[0]) in the temperature schedule, stop. However, if reasonable numbers (all iterations for any of the T values) of last iterations have the same objective function values (no improvement), we terminate the algorithm.

We conduct a computational study to test the efficiency and effectiveness of the proposed simulated annealing algorithms and present the corresponding numerical results in the related chapter.

CHAPTER 5

COMPUTATIONAL STUDY

We conduct an extensive computational study to test the performance of the proposed algorithms. In this chapter, we first explain the problem instance generation procedure in detail. Then we discuss the implementation details of the algorithm and the performance of the algorithms.

5.1 Problem Instances Generation

In order to test the computational performance of our solution methods, we generated random problem instances with different properties, in terms of problem size, precedence graph structure, the success probabilities and testing costs.

- Problem size: We let the number of components n to assume 4 different values namely, 20, 50,100, 200.
- Testing cost and priori testing success probability: We generated the testing costs and success probabilities by using uniform distribution. The testing costs are generated with parameters 1.0 and 10.0. The success probabilities ware generated with different parameters such as (0.0-1.0), (0.5-1.0) and (0.75-1.0).
- Precedence relationship: We generated precedence graphs by inserting 1%, 5%, 10%, 20%, 40%, 50%, 75% of all possible arcs in the precedence graph.

Then we named our problem instances according to generating strategy of success probability. If we use the parameters (0.0-1.0) for success probability, we name this group of data set as A. Likely, when we use parameters (0.5-1.0) and (0.75-1.0), we name data set B and C, respectively. Then, we separate the data into eight subgroups according to strictness of precedence relationship. When we use 1%, 5%, 10%, 20%, 40%, 50%, 75% precedence relationship for data set A, these subgroups were named as $AA, AB, .., AH$, respectively. Data set B and C are named in this same strategy. We generated 10 data for all subgroup of data set. So, for different n values, we solved the problem for 240 different data. In total we have 960 problem instances and we solved these instances for 3 times with parameters k=n/2, k=n/3 and k=n/4.

5.2 Computational Results

5.2.1 Initial Solution

In Chapter 4, we discuss various methods to find an initial solution methodology. We try to select the next node according to a merit value which is obtained by increasing value of testing cost/priori success probability, testing cost/priori failure probability, testing cost/[(priori success probability)(priori failure probability)], random selection.

We perform all these criteria using a subset of our data set generated before, according to different k values. Our main goal is, to decide which selection criteria gives us a better initial solution. If we start with a better initial solution, we expect to have better solutions at the end.. When $k = n/2$, $k = n/3$ and $k = n/4$ the tables below give a comparative analysis among all mentioned selection criteria. For the parameter k, we just choose values less than or equal to $n/2$. This is due to the fact that the state of a k-out-of-n systems depends only on the number of functioning components. For instance, let us consider a 3-out-of-7 system. This system functions if at least 3 components function and fails if at least 5 components fail. On the other hand, if we consider a 5-out-of-7 system, the system functions if at least 5 components function and fails if at least 3 components fail. If we just interchange the labels of states, i.e. working state becomes failing state and vice versa, we essentially get the same system. In fact, these system functions are dual of each other in Boolean function context. Essentially, it suffices to investigate only k values that are less than or equal to $n/2$. Let us recall that a 1-out-of-n system is a parallel system and there are some optimality results for these systems under special conditions. When k is small, in some way, one can argue that the system behaves somehow similar to the parallel system. Thinking in this way, one can also argue that the most difficult instances are when $k = n/2$, since in this case it is difficult to prove that the system is functioning (we need $n/2$ functioning components) and it is also difficult to prove that the system is failing since we need $n/2 + 1$ failing components.

As it is seen in tables 5.1, 5.2 and 5.3 to select the available items changes according to different k values. For $k = n/2$ to use c/p is the best choice to start a good initial feasible solution, whereas for $k = n/3$ and $k = n/4$ to use c/q is the best choice to find the best initial feasible solution. So, c/q is the best choice to obtain a good initial feasible solution.

k=n/2	**c/p**	**c/pq**	**c/q**	**random**	**Best Selection Methodology**
n=20	79.47	82.76	78.09	84.81	**c/q**
n=50	189.80	204.19	202.66	207.43	**c/p**
n=100	381.54	399.85	393.57	401.63	**c/p**
n=200	793.65	821.63	820.25	825.08	**c/p**

Table 5.1: Objective function values for initial solution when k=n/2

k=n/3	**c/p**	**c/pq**	**c/q**	**random**	**Best Selection Methodology**
n=20	81.56	81.56	75.58	85.60	**c/q**
n=50	213.20	213.20	206.28	218.45	**c/q**
n=100	415.93	415.93	402.17	422.38	**c/q**
n=200	860.10	860.10	842.89	862.91	**c/q**

Table 5.2: Objective function values for initial solution when k=n/3

k=n/4	**c/p**	**c/pq**	**c/q**	**random**	**Best Selection Methodology**
n=20	20.00	20.00	20.00	20.00	**c/q**
n=50	50.00	50.00	50.00	50.00	**c/q**
n=100	100.00	100.00	100.00	100.00	**c/q**
n=200	200.00	200.00	200.00	200.00	**c/q**

Table 5.3: Objective function values for initial solution when k=n/4

5.2.2 Tabu Search Algorithm

In this section, the computational results for different n, k values and the parameters of Tabu Search Algorithm are presented. Our algorithm is developed under precedence constraints. If the precedence constraints are violated, the objective function is modified as explained in Chapter 4. To make this modification, we add a penalty function $\beta P(\mathbf{x})$. Every 5 iterations the penalty coefficient β is divided by 2 if all 5 previous solutions were feasible or multiplied by 2 if all were infeasible. Also, our algorithm is terminated if the maximum computation time is met. We define this value as 500 seconds in Chapter 4.

Average improvement values between the initial solution and the solutions stated by the Tabu Search Algorithm are given below $k = n/2$, $k = n/3$, $k = n/4$. Firstly, in tables 5.4, 5.5, 5.6 the improvement values between initial solution and Tabu search Algorithm of $k = n/2$ for data type A, B and C, respectively. As it is seen from tables, the improvement values of the algorithm for data type A is the least. A is generated with a success probability with the parameters (0.0-1.0). For data type B and C parameters are (0.5-1.0) and (0.75-1.0) respectively. So, B and C it is easier to select nodes according to their success probabilities. Also, the average improvement values are lower when the precedence relationship is strict. We have divided to data set into eight subgroups for A, B and C. However average improvement values for data set AA is 1.57%, it gets lower and it becomes 0.62% for AH. Because from A to H, instances are generated with more strict precedence constraints. If an instance is relaxed, than we can find so many strategy to try and hit a better one to minimize the total expected cost.

Secondly, in tables 5.7, 5.8, 5.9 the improvement values of $k = n/3$ for data type A, B and C, respectively. As it is seen from tables, the improvement values of the algorithm for data type A is still the least. Also, the average improvement values are lower when the precedence relationship is strict. We have divided to data set into eight subgroups for A, B and C. However average improvement values for data set AA is 6.70%, it gets lower and it becomes 2.39% for AH. Because from A to H, instances are generated with more strict precedence constraints. If an instance is relax, than we can find so many strategy to try and hit a better one to minimize the total expected cost.

Lastly, in tables 5.10, 5.11, 5.12 improvement values of $k = n/4$ for data type A, B and C, respectively. As seen from the tables, the improvement values of the algorithm for data type A is the least as same as for $k = n/2$ and $k = n/3$. Also, the average improvement values are still lower when the precedence relationship is strict.

5.2.3 Simulated Annealing Algorithm

In this section, the computational results and parameters of Simulated Annealing Algorithm are given. Before giving the computational results, we provide improvement values for Temperature parameter of Simulated Annealing algorithm. $T = 2\%$, $T = 5\%$, $T = 10\%$, $T = 20\%$ are tried to select the best value. Below table, for the same subset which is used for selection criteria for initial solution, improvement of objective function values are given for all different T values. And $T = 20\%$ gives more efficient results if we take average of the whole data that is used for this selection.

Below, we analyze the improvement of Simulated Annealing Algorithm based on defined initial solution. Best numerical results are studied by trying different temperature values. Generally the initial temperature value gives the best results, nearly 44% of the number of best solutions of data set. When possible number of iterations has been performed at the smallest value of T in the temperature schedule, the algorithm is stopped. However, for any of the T values if all possible iterations are performed and there is no improvement in the objective function value, then our algorithm is stopped.

Average improvement values between initial solution and Tabu Search Algorithm results are given below $k = n/2$, $k = n/3$, $k = n/4$. Firstly, in tables 5.14, 5.15, 5.16 improvement values of $k = n/2$ for data type A, B and C, respectively. Likewise in Tabu Search Algorithm results, improvement values of the algorithm for data type A is the least. A is generated with a success probability with the parameters (0.0-1.0). For data type B and C parameters are (0.5-1.0) and (0.75-1.0) respectively. Also, average improvement values are lower when the precedence relationship is strict. As it is given for Tabu Search results, improvement values get lower from A to H because of strictness of the precedence relationship.

Secondly, in tables 5.17, 5.18, 5.19 improvement values of $k = n/3$ for data type A, B and C, respectively. As it is seen from tables, improvement values of the algorithm for data type A is still the least. Also, the average improvement values are lower when the precedence relationship is strict.

Lastly, in tables 5.20, 5.21, 5.22 improvement values of $k = n/4$ for data type A, B and C, respectively. As it is seen from tables, improvement values of the algorithm for data type A is the least as same as for $k = n/2$ and $k = n/3$. Also, average improvement values are still lower when the precedence relationship is strict.

5.3 Comparison of Algorithms

In this section, Intersection-Precedence Algorithm and our heuristics algorithms are compared and analyzed. In general, we observe that Intersection-Precedence Algorithm generally improves the objective function value. For some instances, determining the next component to inspect according to the current testing state can lead to worse results. As it is seen below, in some cases our proposed algorithm does not improve our objective function value. It runs faster than the heuristics for small n values, the comparative analysis for the running time will be given.

Firstly, comparison of algorithms is given for $k = n/2$. As it is seen from the tables 5.23, 5.24, Simulated Annealing and Tabu Search heuristics give good results from initial solution. They give a testing sequence and the next component to test is not depended on the states of already tested. Our Intersection-Precedence Algorithm gives results for strategies described by a binary tree, and also, it gives generally better results than the heuristics. Generally, for smaller values of n average improvement values of Intersection-Precedence Algorithm than heuristics are higher with respect to higher values of n. When there are a few nodes to order, to decide which node to test according to present state can make a sharp decrease in the objective function value that obtained by using a permutation strategy. For example, when there are 100 nodes to test, to make a change in the order according to previous states cannot make a reasonable numerical effect in the objective function value. If the problem has a strict precedence relationship, there are a few possible orders and heuristics can not improve the objective function so much. But Intersection-Precedence Algorithm can choose the available items by looking at every state and it can improve the objective function value higher than the heuristics. As it is seen in tables 5.25, 5.26, for $n = 100$ and $n = 200$ heuristics give more faster results because available items are searched at every states of INTER-PREC. Also, average improvement values are getting higher from data set A to C. For example, when $k = n/2, n = 50$ average improvement value of Intersection-Precedence Algorithm with respect to Simulated Annealing is 0.74% for A, 7.24% for B, 7.48% for C. The prior success probabilities are higher and the failure probabilities are lower for C. As it is stated before, Intersection-Precedence Algorithm decides which item to test according to present state. Our next item to test can be changed according to result of our previous state is success or failure. When the items have a high success probability, it is easy to decide the next node by using the information of previous states. As a summary, for $k = n/2$ our developed algorithm give better results from heuristics according to objective function value.

After making a comparison between Intersection-Precedence Algorithm and Heuristics Algorithms, in Table 5.27 a comparison of Simulated Annealing Algorithm and Intersection Algorithm when there are no precedence constraints is given. To get benchmark results we used some well-known heuristics. To know that our algorithm really performs better, it is needed to to know performance of our benchmark results. Simulated Annealing and Tabu Search Algorithms gave similar results, so we made a comparison by using one of them, Simulated Annealing. We know that when there are no precedence constraints, Intersection Algorithm gives optimal results. So, we used same data without precedence constraints and solved by using Intersection Algorithm. We found optimal results for this data set. Then, we used Simulated Annealing for same data set and made a comparison. For $k = n/2$ Simulated Annealing Algorithm has 4.7% bigger objective function value than Intersection Algorithm. This result shows us our heuristics are good benchmark results. This analysis is not taken for also $k = n/3$ and $k = n/4$.

Likewise $k = n/2$ it is given below computational results for $k = n/3$. As it is seen from the tables 5.28, 5.29, our Intersection-Precedence Algorithm gives more efficient results from heuristics. INTER-PREC improves the objective function value 30% higher than the SA and TS. Generally, for smaller values of n average improvement values of Intersection-Precedence Algorithm than heuristics are higher with respect to higher values of n. Also, average improvement values are getting higher from data set A to C. Likewise for $k = n/2$, as it is seen in tables 5.30, for $n = 100$ and $n = 200$

Simulated Annealing Algorithm gives more faster results because available items are searches at every states of INTER-PREC. Also, Tabu Search Algorithm gives more time consuming results for only $n = 20$ than INTER-PREC that is shown in Table 5.31.

Lastly, comparison of algorithms are given for $k = n/4$. Again, as it is seen from the tables 5.32, 5.33, Intersection-Precedence Algorithm gives more efficient results from heuristics. INTER-PREC improves the objective function value 30% higher than the SA and TS. Generally, for smaller values of n average improvement values of Intersection-Precedence Algorithm than heuristics are higher with respect to higher values of n. Likewise for $k = n/3$, as it is seen in Table 5.30, for $n = 100$ and $n = 200$ Simulated Annealing Algorithm gives more faster results because INTER-PREC searches for available items at every states. Also, Tabu Search Algorithm gives more time consuming results for only $n = 20$ that is shown in Table 5.31.

Tabu Search Results - Data Set A	AA	AB	AC	AD	AE	AF	AG	AH	Improvement of Different n Values
n=20	2.97%	1.21%	1.85%	1.32%	0.25%	0.49%	0.09%	0.12%	**1.04%**
n=50	1.33%	1.24%	1.06%	1.05%	0.28%	0.45%	0.02%	0.02%	**0.68%**
n=100	1.01%	1.48%	0.57%	0.36%	0.21%	0.06%	0.07%	0.04%	**0.48%**
n=200	0.96%	0.34%	0.50%	0.29%	0.07%	0.04%	0.03%	0.01%	**0.28%**
Improvement of all Data Set Subgroup	**1.57%**	**1.07%**	**0.99%**	**0.75%**	**0.20%**	**0.26%**	**0.05%**	**0.05%**	**0.62%**

Table 5.4: Average improvement values of Tabu Search Heuristics for Data Set A for k=n/2

Tabu Search Results - Data Set B	BA	BB	BC	BD	BE	BF	BG	BH	Improvement of Different n Values
n=20	10.67%	9.97%	9.47%	3.92%	1.92%	0.90%	1.01%	0.11%	**4.75%**
n=50	7.39%	6.87%	3.85%	1.97%	1.07%	0.73%	0.48%	0.24%	**2.82%**
n=100	3.22%	3.00%	1.82%	1.26%	0.43%	0.28%	0.25%	0.09%	**1.29%**
n=200	2.47%	1.66%	1.10%	0.47%	0.18%	0.13%	0.12%	0.07%	**0.78%**
Improvement of all Data Set Subgroup	**5.94%**	**5.38%**	**4.06%**	**1.90%**	**0.90%**	**0.51%**	**0.47%**	**0.13%**	**2.41%**

Table 5.5: Average improvement values of Tabu Search Heuristics for Data Set B for k=n/2

Tabu Search Results - Data Set C	CA	CB	CC	CD	CE	CF	CG	CH	Improvement of Different n Values
n=20	12.12%	9.63%	7.43%	3.79%	1.48%	0.60%	0.77%	0.69%	**4.56%**
n=50	7.91%	6.76%	3.36%	2.25%	1.25%	0.65%	0.16%	0.13%	**2.81%**
n=100	4.46%	3.04%	1.68%	1.10%	0.21%	0.14%	0.29%	0.10%	**1.38%**
n=200	2.30%	1.46%	0.92%	0.54%	0.15%	0.16%	0.10%	0.83%	**0.81%**
Improvement of all Data Set Subgroup	**6.70%**	**5.22%**	**3.35%**	**1.92%**	**0.77%**	**0.39%**	**0.33%**	**0.44%**	**2.39%**

Table 5.6: Average improvement values of Tabu Search Heuristics for Data Set C for k=n/2

Tabu Search Results - Data Set A	AA	AB	AC	AD	AE	AF	AG	AH	Improvement of Different n Values
n=20	1.36%	1.94%	1.93%	1.88%	0.58%	0.42%	0.02%	0.11%	**1.03%**
n=50	0.61%	1.45%	0.93%	0.39%	0.41%	0.21%	0.01%	0.02%	**0.50%**
n=100	0.53%	2.28%	0.60%	0.18%	0.10%	0.06%	0.01%	0.02%	**0.47%**
n=200	1.17%	0.76%	0.21%	0.09%	0.04%	0.03%	0.01%	0.00%	**0.29%**
Improvement of all Data Set Subgroup	**0.92%**	**1.61%**	**0.92%**	**0.64%**	**0.28%**	**0.18%**	**0.01%**	**0.04%**	**0.57%**

Table 5.7: Average improvement values of Tabu Search Heuristics for Data Set A for k=n/3

Tabu Search Results - Data Set B	BA	BB	BC	BD	BE	BF	BG	BH	Improvement of Different n Values
n=20	3.65%	4.20%	3.63%	1.54%	0.93%	0.36%	0.11%	0.14%	**1.82%**
n=50	3.78%	3.22%	2.25%	0.84%	0.31%	0.13%	0.18%	0.06%	**1.35%**
n=100	2.78%	1.47%	1.48%	0.66%	0.33%	0.13%	0.15%	0.05%	**0.88%**
n=200	1.41%	1.30%	0.65%	0.38%	0.17%	0.12%	0.06%	0.02%	**0.51%**
Improvement of all Data Set Subgroup	**2.91%**	**2.55%**	**2.00%**	**0.85%**	**0.43%**	**0.18%**	**0.12%**	**0.07%**	**1.14%**

Table 5.8: Average improvement values of Tabu Search Heuristics for Data Set B for k=n/3

Tabu Search Results - Data Set C	CA	CB	CC	CD	CE	CF	CG	CH	Improvement of Different n Values
n=20	6.50%	6.14%	4.57%	1.98%	0.56%	0.86%	0.24%	0.41%	**2.66%**
n=50	5.42%	5.06%	2.76%	1.04%	0.30%	0.20%	0.13%	0.31%	**1.90%**
n=100	3.47%	2.64%	1.74%	0.84%	0.35%	0.20%	0.15%	0.12%	**1.19%**
n=200	2.00%	1.27%	0.69%	0.40%	0.15%	0.10%	0.07%	0.65%	**0.66%**
Improvement of all Data Set Subgroup	**4.35%**	**3.78%**	**2.44%**	**1.06%**	**0.34%**	**0.34%**	**0.15%**	**0.37%**	**1.60%**

Table 5.9: Average improvement values of Tabu Search Heuristics for Data Set C for k=n/3

Tabu Search Results - Data Set A	AA	AB	AC	AD	AE	AF	AG	AH	Improvement of Different n Values
n=20	1.93%	4.11%	3.35%	3.05%	1.70%	0.45%	0.01%	0.11%	**1.84%**
n=50	0.60%	2.31%	1.74%	1.12%	0.19%	0.10%	0.00%	0.01%	**0.76%**
n=100	0.62%	1.64%	0.78%	0.22%	0.20%	0.09%	0.03%	0.00%	**0.45%**
n=200	1.01%	0.43%	0.42%	0.14%	0.09%	0.03%	0.02%	0.00%	**0.27%**
Improvement of all Data Set Subgroup	**1.04%**	**2.12%**	**1.57%**	**1.13%**	**0.55%**	**0.17%**	**0.01%**	**0.03%**	**0.83%**

Table 5.10: Average improvement values of Tabu Search Heuristics for Data Set A for k=n/4

Tabu Search Results - Data Set B	BA	BB	BC	BD	BE	BF	BG	BH	Improvement of Different n Values
n=20	1.47%	0.77%	1.12%	0.66%	0.24%	0.37%	0.01%	0.06%	**0.59%**
n=50	1.01%	1.53%	0.69%	0.31%	0.05%	0.05%	0.05%	0.04%	**0.47%**
n=100	0.87%	0.48%	0.32%	0.19%	0.09%	0.07%	0.06%	0.01%	**0.26%**
n=200	0.94%	0.41%	0.25%	0.12%	0.02%	0.03%	0.01%	0.01%	**0.23%**
Improvement of all Data Set Subgroup	**1.08%**	**0.80%**	**0.60%**	**0.32%**	**0.10%**	**0.13%**	**0.03%**	**0.03%**	**0.39%**

Table 5.11: Average improvement values of Tabu Search Heuristics for Data Set B for k=n/4

Tabu Search Results - Data Set C	CA	CB	CC	CD	CE	CF	CG	CH	Improvement of Different n Values
n=20	4.36%	3.56%	1.26%	0.81%	0.52%	0.70%	0.36%	0.33%	**1.49%**
n=50	4.09%	3.42%	2.01%	1.34%	0.58%	0.10%	0.14%	0.03%	**1.46%**
n=100	2.42%	1.65%	1.49%	0.89%	0.38%	0.22%	0.16%	0.05%	**0.91%**
n=200	1.80%	1.08%	0.71%	0.29%	0.14%	0.08%	0.04%	0.41%	**0.57%**
Improvement of all Data Set Subgroup	**3.17%**	**2.43%**	**1.37%**	**0.83%**	**0.41%**	**0.27%**	**0.17%**	**0.20%**	**1.11%**

Table 5.12: Average improvement values of Tabu Search Heuristics for Data Set C for k=n/4

Comparative results for different T values	n=20	n=50	n=100	n=200
T=2%	3.24%	1.35%	0.92%	0.54%
T=5%	3.05%	1.20%	0.86%	0.88%
T=10%	3.21%	1.23%	0.82%	0.81%
T=20%	3.06%	1.17%	0.94%	0.94%
Selected Methodology	T=2%	T=2%	T=20%	T=20%

Table 5.13: Computational results of different Temperature results

Simulated Annealing Results - Data Set A	AA	AB	AC	AD	AE	AF	AG	AH	Improvement of Different n Values
n=20	2.04%	1.61%	1.90%	2.40%	0.81%	0.55%	0.13%	0.18%	**1.20%**
n=50	1.07%	1.14%	0.98%	0.93%	0.45%	0.37%	0.07%	0.06%	**0.63%**
n=100	0.63%	0.53%	0.49%	0.36%	0.24%	0.09%	0.04%	0.06%	**0.30%**
n=200	0.40%	0.25%	0.41%	0.29%	0.05%	0.03%	0.02%	0.00%	**0.18%**
Improvement of all Data Set Subgroup	**1.03%**	**0.88%**	**0.94%**	**1.00%**	**0.39%**	**0.26%**	**0.06%**	**0.07%**	**0.58%**

Table 5.14: Average improvement values of Simulated Annealing Heuristics for Data Set A for k=n/2

Simulated Annealing Results - Data Set B	BA	BB	BC	BD	BE	BF	BG	BH	Improvement of Different n Values
n=20	6.62%	6.27%	5.73%	3.36%	1.68%	1.23%	1.01%	0.16%	**3.26%**
n=50	4.60%	3.14%	2.81%	1.96%	1.05%	0.48%	0.48%	0.19%	**1.84%**
n=100	1.22%	1.85%	0.96%	0.98%	0.35%	0.26%	0.20%	0.08%	**0.74%**
n=200	0.64%	0.88%	0.65%	0.29%	0.13%	0.05%	0.08%	0.04%	**0.34%**
Improvement of all Data Set Subgroup	**3.27%**	**3.03%**	**2.54%**	**1.65%**	**0.80%**	**0.51%**	**0.44%**	**0.12%**	**1.54%**

Table 5.15: Average improvement values of Simulated Annealing Heuristics for Data Set B for k=n/2

Simulated Annealing Results - Data Set C	CA	CB	CC	CD	CE	CF	CD	CH	Improvement of Different n Values
n=20	5.28%	4.35%	2.39%	2.43%	1.02%	0.68%	0.83%	0.73%	**2.22%**
n=50	3.40%	2.79%	1.65%	1.60%	0.93%	0.70%	0.17%	0.17%	**1.43%**
n=100	2.43%	1.20%	1.46%	0.95%	0.21%	0.18%	0.31%	0.10%	**0.85%**
n=200	0.85%	0.61%	0.64%	0.49%	0.12%	0.14%	0.08%	0.47%	**0.42%**
Improvement of all Data Set Subgroup	**2.99%**	**2.24%**	**1.53%**	**1.37%**	**0.57%**	**0.43%**	**0.35%**	**0.37%**	**1.23%**

Table 5.16: Average improvement values of Simulated Annealing Heuristics for Data Set C for k=n/2

Simulated Annealing Results - Data Set A	AA	AB	AC	AD	AE	AF	AG	AH	Improvement of Different n Values
n=20	1.39%	2.09%	2.60%	3.16%	1.07%	0.74%	0.22%	0.12%	**1.42%**
n=50	0.57%	1.84%	1.79%	1.62%	0.80%	0.23%	0.11%	0.02%	**0.87%**
n=100	0.14%	1.31%	0.66%	0.46%	0.14%	0.11%	0.02%	0.03%	**0.36%**
n=200	0.65%	0.76%	0.45%	0.24%	0.05%	0.04%	0.02%	0.00%	**0.27%**
Improvement of all Data Set Subgroup	**0.69%**	**1.50%**	**1.37%**	**1.37%**	**0.51%**	**0.28%**	**0.09%**	**0.05%**	**0.73%**

Table 5.17: Average improvement values of Simulated Annealing Heuristics for Data Set A for k=n/3

Simulated Annealing Results - Data Set B	BA	BB	BC	BD	BE	BF	BG	BH	Improvement of Different n Values
n=20	3.59%	2.38%	3.05%	2.02%	1.16%	0.40%	0.23%	0.20%	**1.63%**
n=50	1.96%	1.58%	1.29%	0.54%	0.35%	0.16%	0.22%	0.08%	**0.77%**
n=100	0.84%	0.88%	1.14%	0.63%	0.32%	0.11%	0.18%	0.04%	**0.52%**
n=200	0.60%	0.72%	0.52%	0.33%	0.16%	0.05%	0.02%	0.01%	**0.30%**
Improvement of all Data Set Subgroup	**1.75%**	**1.39%**	**1.50%**	**0.88%**	**0.50%**	**0.18%**	**0.16%**	**0.08%**	**0.81%**

Table 5.18: Average improvement values of Simulated Annealing Heuristics for Data Set B for k=n/3

Simulated Annealing Results - Data Set C	CA	CB	CC	CD	CE	CF	CD	CH	Improvement of Different n Values
n=20	5.03%	3.46%	2.56%	2.19%	1.35%	0.97%	0.56%	0.55%	**2.08%**
n=50	2.79%	2.24%	1.50%	0.78%	0.31%	0.25%	0.13%	0.21%	**1.03%**
n=100	1.54%	0.95%	1.21%	0.65%	0.32%	0.27%	0.14%	0.09%	**0.65%**
n=200	0.60%	0.79%	0.44%	0.36%	0.13%	0.09%	0.07%	0.39%	**0.36%**
Improvement of all Data Set Subgroup	**2.49%**	**1.86%**	**1.43%**	**0.99%**	**0.53%**	**0.40%**	**0.23%**	**0.31%**	**1.03%**

Table 5.19: Average improvement values of Simulated Annealing Heuristics for Data Set C for k=n/3

Simulated Annealing Results - Data Set A	AA	AB	AC	AD	AE	AF	AG	AH	Improvement of Different n Values
n=20	1.53%	4.45%	4.15%	2.90%	2.36%	1.38%	1.11%	0.12%	**2.25%**
n=50	0.58%	2.53%	2.21%	2.15%	0.39%	0.24%	0.08%	0.03%	**1.03%**
n=100	0.54%	1.93%	1.41%	0.50%	0.28%	0.18%	0.06%	0.02%	**0.61%**
n=200	0.39%	0.76%	0.62%	0.19%	0.14%	0.03%	0.04%	0.00%	**0.27%**
Improvement of all Data Set Subgroup	**0.76%**	**2.42%**	**2.10%**	**1.44%**	**0.79%**	**0.46%**	**0.32%**	**0.04%**	**1.04%**

Table 5.20: Average improvement values of Simulated Annealing Heuristics for Data Set A for k=n/4

Simulated Annealing Results - Data Set B	BA	BB	BC	BD	BE	BF	BG	BH	Improvement of Different n Values
n=20	1.68%	1.80%	1.91%	1.21%	0.69%	0.60%	0.12%	0.13%	**1.02%**
n=50	1.12%	1.40%	0.91%	0.62%	0.14%	0.08%	0.10%	0.05%	**0.55%**
n=100	0.52%	0.37%	0.43%	0.18%	0.15%	0.05%	0.05%	0.00%	**0.22%**
n=200	0.29%	0.32%	0.24%	0.16%	0.03%	0.03%	0.01%	0.00%	**0.14%**
Improvement of all Data Set Subgroup	**0.90%**	**0.97%**	**0.87%**	**0.54%**	**0.25%**	**0.19%**	**0.07%**	**0.05%**	**0.48%**

Table 5.21: Average improvement values of Simulated Annealing Heuristics for Data Set B for k=n/4

Simulated Annealing Results - Data Set C	CA	CB	CC	CD	CE	CF	CD	CH	Improvement of Different n Values
n=20	3.42%	3.46%	1.93%	1.28%	1.01%	1.22%	0.71%	0.41%	**1.68%**
n=50	1.83%	1.34%	1.24%	0.86%	0.66%	0.15%	0.16%	0.10%	**0.79%**
n=100	1.19%	0.97%	0.88%	0.70%	0.34%	0.23%	0.15%	0.07%	**0.57%**
n=200	0.43%	0.63%	0.57%	0.28%	0.09%	0.08%	0.03%	0.28%	**0.30%**
Improvement of all Data Set Subgroup	**1.72%**	**1.60%**	**1.15%**	**0.78%**	**0.53%**	**0.42%**	**0.26%**	**0.22%**	**0.84%**

Table 5.22: Average improvement values of Simulated Annealing Heuristics for Data Set C for k=n/4

SA- INTER-PREC	A	B	C	Improvement of Different n Values
n=20	-3.34%	10.72%	11.92%	6.43%
n=50	-0.74%	7.24%	7.48%	4.66%
n=100	-0.19%	4.88%	4.69%	3.13%
n=200	0.03%	3.25%	2.80%	2.03%
Improvement of all Data Set Group	-1.06%	6.52%	6.72%	**4.1%**

Table 5.23: Average improvement values of Intersection-Precedence Algorithm from Simulated Annealing Algorithm for k=n/2

TS - INTER-PREC	A	B	C	Improvement of Different n Values
n=20	-3.17%	9.31%	9.72%	5.29%
n=50	-0.79%	6.29%	6.21%	3.90%
n=100	-0.36%	4.36%	4.18%	2.73%
n=200	-0.06%	2.84%	2.48%	1.75%
Improvement of all Data Set Group	-1.10%	5.70%	5.65%	**3.42%**

Table 5.24: Average improvement values of Intersection-Precedence Algorithm from Tabu Search Algorithm for k=n/2

Computational Time Values (seconds)	A (SA)	A (INTER-PREC)	B (SA)	B (INTER-PREC)	C (SA)	C (INTER-PREC)	Minimum Time Consuming Methodology
n=20	0.05	0.01	0.05	0.01	0.05	0.01	INTER-PREC
n=50	0.51	0.49	0.51	0.49	0.51	0.49	INTER-PREC
n=100	5.70	14.48	5.81	14.11	6.09	14.06	SA
n=200	86.84	466.82	86.02	459.27	90.84	438.71	SA

Table 5.25: Average time values of Intersection-Precedence Algorithm and Simulated Annealing Algorithm for k=n/2

Computational Time Values (seconds)	A (TS)	A (INTER-PREC)	B (TS)	B (INTER-PREC)	C (TS)	C (INTER-PREC)	Minimum Time Consuming Methodology
n=20	0.05	0.01	0.05	0.01	0.05	0.01	INTER-PREC
n=50	0.42	0.49	0.60	0.49	0.50	0.49	INTER-PREC
n=100	4.22	14.48	4.90	14.11	4.28	14.06	TS
n=200	57.85	466.82	56.50	459.27	54.42	438.71	TS

Table 5.26: Average time values of Intersection-Precedence Algorithm and Tabu Search Algorithm for k=n/2

SA-Intersection	A	B	C	Comparison of Different n Values
n=20	4.79%	6.00%	7.45%	6.08%
n=50	3.28%	4.10%	5.09%	4.16%
n=100	4.30%	5.04%	5.36%	4.90%
n=200	2.75%	4.00%	4.13%	3.62%
Comparison of all Data Set Group	3.78%	4.78%	5.51%	**4.7%**

Table 5.27: Comparison of Intersection Algorithm and Simulated Annealing Algorithm for k=n/2

SA- INTER-PREC	A	B	C	Improvement of Different n Values
n=20	-23.68%	49.67%	52.98%	26.32%
n=50	-11.80%	52.29%	53.79%	31.42%
n=100	-11.61%	53.20%	52.93%	31.51%
n=200	-6.36%	51.74%	51.35%	32.24%
Improvement of all Data Set Group	-13.37%	51.73%	52.76%	**30.4%**

Table 5.28: Average improvement values of Intersection-Precedence Algorithm from Simulated Annealing Algorithm for k=n/3

TS- INTER-PREC	A	B	C	Improvement of Different n Values
n=20	-23.07%	49.67%	52.74%	26.45%
n=50	-11.42%	52.03%	53.40%	31.34%
n=100	-11.78%	53.04%	52.68%	31.31%
n=200	-6.40%	51.65%	51.23%	32.16%
Improvement of all Data Set Group	-13.17%	51.60%	52.51%	**30.31%**

Table 5.29: Average improvement values of Intersection-Precedence Algorithm from Tabu Search Algorithm for k=n/3

Computational Time Values (seconds)	A (SA)	A (INTER-PREC)	B (SA)	B (INTER-PREC)	C (SA)	C (INTER-PREC)	Minimum Time Consuming Methodology
n=20	0.05	0.02	0.05	0.01	0.04	0.01	INTER-PREC
n=50	0.53	0.45	0.50	0.51	0.49	0.45	INTER-PREC
n=100	5.65	12.84	6.08	12.63	6.09	12.70	SA
n=200	84.29	201.17	84.39	109.09	86.60	99.75	SA

Table 5.30: Average time values of Intersection-Precedence Algorithm and Simulated Annealing Algorithm for k=n/3

Computational Time Values (seconds)	A (TS)	A (INTER-PREC)	B (TS)	B (INTER-PREC)	C (TS)	C (INTER-PREC)	Minimum Time Consuming Methodology
n=20	0.03	0.02	0.04	0.01	0.03	0.01	INTER-PREC
n=50	0.30	0.45	0.42	0.51	0.44	0.45	TS
n=100	3.63	12.84	4.23	12.63	4.36	12.70	TS
n=200	58.54	201.17	60.57	109.09	58.21	99.75	TS

Table 5.31: Average time values of Intersection-Precedence Algorithm and Tabu Search Algorithm for k=n/3

SA- INTER-PREC	A	B	C	Improvement of Different n Values
n=20	-39.07%	60.72%	70.46%	30.70%
n=50	-22.01%	64.63%	71.56%	38.06%
n=100	-15.46%	64.75%	68.26%	39.19%
n=200	-8.40%	65.44%	67.85%	41.63%
Improvement of all Data Set Group	-21.23%	63.89%	69.53%	**37.4%**

Table 5.32: Average improvement values of Intersection-Precedence Algorithm from Simulated Annealing Algorithm for k=n/4

TS- INTER-PREC	A	B	C	Improvement of Different n Values
n=20	-38.66%	60.96%	70.59%	30.96%
n=50	-21.72%	64.68%	71.39%	38.12%
n=100	-15.29%	64.74%	68.15%	39.20%
n=200	-8.43%	65.41%	67.78%	41.58%
Improvement of all Data Set Group	-21.02%	63.95%	69.47%	**37.47%**

Table 5.33: Average improvement values of Intersection-Precedence Algorithm from Tabu Search Algorithm for k=n/4

Computational Time Values (seconds)	A (SA)	A (INTER-PREC)	B (SA)	B (INTER-PREC)	C (SA)	C (INTER-PREC)	Minimum Time Consuming Methodology
n=20	0.05	0.01	0.04	0.01	0.04	0.01	INTER-PREC
n=50	0.56	0.35	0.50	0.35	0.46	0.36	INTER-PREC
n=100	6.27	10.86	5.49	10.86	5.97	10.88	SA
n=200	85.64	333.20	82.08	397.43	84.61	336.24	SA

Table 5.34: Average time values of Intersection-Precedence Algorithm and Simulated Annealing Algorithm for k=n/4

Computational Time Values (seconds)	A (TS)	A (INTER-PREC)	B (TS)	B (INTER-PREC)	C (TS)	C (INTER-PREC)	Minimum Time Consuming Methodology
n=20	0.03	0.01	0.03	0.01	0.03	0.01	INTER-PREC
n=50	0.30	0.35	0.32	0.35	0.41	0.36	TS
n=100	3.42	10.86	3.66	10.86	4.22	10.88	TS
n=200	59.92	333.20	57.83	397.43	59.28	336.24	TS

Table 5.35: Average time values of Intersection-Precedence Algorithm and Tabu Search Algorithm for k=n/4

CHAPTER 6

CONCLUSION AND FUTURE RESEARCH

In this study, the sequential testing problem for k-out-of-n systems under general type precedence constraints is investigated. We use the results from the literature for the sequential testing problem for k-out-of-n systems without any precedence constraints, to come up with an algorithm, ”Intersection-Precedence” that finds feasible strategies that can be efficiently represented by a Block-Walking Diagram. We also consider permutation strategies that can also be represented in an efficient manner and try to find good permutation strategies by implementing a Tabu Search and Simulated Annealing algorithm. We compare the performances of all these algorithms and we observe that on the average, Intersection-Precedence outperforms the others.

For the experimental analysis, one major difficulty that we deal with, is the representation of feasible solutions. The natural way to represent feasible solutions is to use Binary Decision Trees. On the other hand, even for moderate values of k and n, the size of the binary decision tree becomes exponential and it is not possible to work with these binary decision trees. That is why, in this work, we concentrate on strategies that can be efficiently represented. From a computational point of view we have to do it, on the other hand, in this way we only consider a subset of the feasible solutions.

As of now, there is no formal proof that the problem is NP-complete even for the series(parallel) systems. We are aware of some efforts towards this end. For the series(parallel) systems the problem resembles very much to the scheduling problem where one tries to minimize the total completion times on a single machine with general precedence constraints. This problem is NP-complete but there is no direct way to use this result to prove the NP-completeness of the sequential testing problem.

We mention some results from the literature where they claim to obtain an optimal algorithm under special conditions related to the data. One could also look for some special classes of precedence graphs where the optimal solution can be found. This has been done for the series(parallel) systems. Again, these do not extend easily to k-out-of-n systems.

Bibliography

[1] Bao Z.W., Cui L.R., Gao W., Optimal Sequential Inspections of Complex Systems Subject to Block-Chain Precedence Constraints, Fourth International Conference on Natural Computation, 2008.

[2] Ben-Dov Y., Optimal Testing Procedures for Special Structures of Coherent Systems, Management Science, 27, 12, 1981.

[3] Bert D. R., Roel L., R&D Project Scheduling when Activities May Fail, IIE Transactions, 40(4):367-384, April 2008.

[4] Biassizo A., Zuzek A., and Novak F., Sequential Diagnosis with Asymmetrical Tests, Comput. J., 41(3):163170, April 1998.

[5] Catay B., Ozluk O., Unluyurt T., TestAnt: An Ant Colony System Approach to Sequential Testing under Precedence Constraints, Expert Systems with Applications, 38, 1494514951, 2011.

[6] Chang M.F., Shi W., Fuchs W. K., Optimal Diagnosis Procedures for k-out-of-n Structures, IEEE Transactions on Computers, 39, 4, 1990.

[7] Chiu S.Y., Cox Jr. L.A., Sun X., Optimal Sequential Inspections of Reliability Systems subject to Parallel-chain Precedence Constraints, Discrete Applied Mathematics, 96-97, 326-327, 1999.

[8] Cox Jr. L.A., Qiu Y., Kuehner W., Heuristic Least-cost Computation of Discrete Classification Functions with Uncertain Argument Values, Ann. Oper. Res. 21, 1-21, 1989.

[9] Dubois D., Wellman M. P., D'Ambrosio B., Smets P.(Eds.), Guess-and-verify Heuristics for Reducing Uncertainties in Expert Classification Systems, Uncer-

tainty in Artificial Intelligence, Proceedings of the Eighth Conference, Morgan Kaufman, Los Altos,CA, 1992.

[10] Flynn J., Chung C-S., A Heuristic Algorithm for Determining Replacement Policies in Consecutive k-out-of-n Systems, Computers and Operations Research, 31, 1335-1348, 2004.

[11] Fraughnaugh K., Ryan J., Zullo H., and Cox Jr. L. A., Heuristics for Efficient Classification, Annals of Operations Research, 78:189200, 1998.

[12] Garey M., Optimal Binary Identification Procedures, SIAM J. Appl. Math., 23(2):173186, July 1972.

[13] Garey M., Optimal Task Sequencing with Precedence Constraints, Discrete Mathematics, vol. 4, pp. 3756, 1973.

[14] Glover F., Tabu Search part I, ORSA Journal on Computing 1, 3, 190-206, 1989.

[15] Glover F., Tabu Search part II, ORSA Journal on Computing 2, 1, 4-32, 1990.

[16] Glover F., Laguna M., Tabu Search, Kluwer Academic Publishers, Dordrecht, 1997.

[17] Greiner R., Finding Optimal Derivation Strategies in Redundant Knowledge Bases, Artificial Intelligence 50, 95-115, 1990.

[18] Halpern J.Y., Evaluating Boolean Function with Random Variables, Internat. J. Systems Sci. 5, 545-553, 1974.

[19] Joyce W.B., Organizations of Unsuccessful R&D Projects, IEEE Trans. Engrg. Manage. EM-18 2, 57-65, 1971.

[20] Kirkpatrick S., Gelatt C.D., and Vecchi, Optimization by Simulated Annealing, Science, Vol. 220, pp. 671-680, 1983.

[21] Pattipati K. R. and Alexandridis M. G., Application of Heuristic Search and Information Theory to Sequential Fault Diagnosis. IEEE Trans. Syst., Man, Cybern., 20:872887, July 1990.

[22] Pattipati K. R. and Dontamsetty M., On a Generalized Test Sequencing Problem, IEEE Trans. Syst., Man, Cybern., 22(2):392396, March-April 1992.

[23] Tanrıverdi A., Testing Strategies for k-out-of-n Systems under Forest Type Precedence Constraints, Sabanci University, Spring 2008.

[24] Ünlüyurt T., Sequential Testing of Complex Systems: A Review, Discrete Applied Mathematics, 142, 189 205, 2004.