Optimal Software Reliability and Optimal Multiprocessor Scheduling Problems

A Thesis

Submitted in fulfilment of the requirement

for the award of the degree of

DOCTOR OF PHILOSOPHY

IN

MATHEMATICS

SUBMITTED BY

Poonam Panwar

(Registration No.: 951011002)

SCHOOL OF MATHEMATICS

THAPAR UNIVERSITY, PATIALA-147004

PUNJAB (INDIA)

November, 2016

Optimal Software Reliability and Optimal Multiprocessor Scheduling Problems

A Thesis

Submitted in fulfillment of the requirement
for the award of the Degree of

DOCTOR OF PHILOSOPHY
IN
MATHEMATICS

SUBMITTED BY

Poonam Panwar
(Registration No.: 951011002)

SCHOOL OF MATHEMATICS
THAPAR UNIVERSITY, PATIALA-147004
PUNJAB (INDIA)

November, 2016

CERTIFICATE

This is to certify that the thesis entitled, "Optimal Software Reliability and Optimal Multiprocessor Scheduling Problems", submitted by Ms. Poonam Panwar in the fulfillment of the requirement for the award of the degree of Doctor of Philosophy in the School of Mathematics, Thapar University, Patiala, is a record of candidate's own work carried by her under our supervision and guidance.

The matter embodied in this thesis has not been submitted in part or full for award of any degree in other University or Institute.

Attestation by supervisors

(Dr. A. K. Lal)
Associate Professor,
School of Mathematics,
Thapar University, Patiala,
Punjab, India

(Dr. C. Mohan)
Professor Emeritus,
Department of Computer of Science and Engineering,
Ambala College of Engineering and Applied Research,
Ambala
Haryana, India

DECLARATION

It is certified that the research work presented in the thesis entitled "Optimal Software Reliability and Optimal Multiprocessor Scheduling Problems", submitted for the award of the degree of Doctor of Philosophy, submitted in School of Mathematics of Thapar University, Patiala, is an authentic record of my own research work carried out under the supervision of of Dr. A.K. Lal and Dr. C. Mohan and references cited herein have been duly acknowledged.

The matter embodied in thesis has not been previously submitted in part of full to any other University or Institute for the award of degree in India or abroad.

(Poonam Panwar)

Attestation by supervisors

(Dr. A. K. Lal)
Associate Professor,
School of Mathematics,
Thapar University, Patiala,
Punjab, India

(Dr. C. Mohan)
Professor Emeritus,
Department of Computer of Science and Engineering,
Ambala College of Engineering and Applied
Research, Ambala
Haryana, India

ACKNOWLEDGEMENTS

First of all I thank the Almighty **Lord** for the wisdom and perseverance He has bestowed upon me throughout my life and in particular during the present Ph.D. study period.

I acknowledge with a sense of deep and sincere gratitude, the meticulous guidance, ingenious discussions and steady encouragement rendered by **Dr. A. K. Lal**, Associate Professor and Head, School of Mathematics, Thapar University, Patiala and **Dr. C. Mohan**, Professor Emeritus, Department of Computer Science and Engineering, Ambala College of Engineering and Applied Research, Ambala. I owe the greatest debt of gratitude to them for suggesting the research problem followed by ever willing help extended to me throughout the period of my research and the freedom with which I could contact them whenever some need arose. I greatly appreciate their support and guidance not only in my research but also in my day to day life.

I take this opportunity to thank **Dr. Prakash Gopalan**, Director and **Dr. R. S. Kaler**, Deputy Director-I and **Dr. Susheel Mittal**, Deputy Director-II, Thapar University, Patiala for providing me university's resources and facilities necessary for carrying out this research work. I am also deeply thankful to **Dr. O.P Pandey**, Dean, Research and Sponsored Projects and **Dr. S.S Bhatia**, Dean Academic Affairs, Thapar University, Patiala, for their support and needful help during the various stages of my research work.I also place on record my indebtedness to **Dr. M. K. Sharma**, **Dr. Ajay Kumar** and **Dr. Harish Garg** for their cooperation and valuable support whenever needed. I also take this opportunity to express my sincere thanks to all faculty members and the staff of School of Mathematics, Thapar University, Patiala for providing me constant encouragement.

My sincere thanks are also due to all my colleagues and friends especially **Mr. Tarun Sachdeva** and **Ms. Tarunpreet Bhatia** and all research scholars of Thapar University who have helped whenever a need arose.

I am profoundly obliged towards **Management and Administration of Ambala College of Engineering and Applied Research, Ambala** in particular to **Dr. Jai Dev Gupta Ji** for granting me permission, motivation and unconditional support to carry out my research work.

I am thankful from core of my heart to **Smt. Tripta Kumari** wife of Prof. C. Mohan for her moral and spiritual support and motivation during present study. My sincere thanks also go to **Smt. Padma Lochana** wife of Dr. A. K. Lal for making me feel homely during my stay at Thapar University.

My heartfelt thanks to my dear husband, **Dr. Satish Kumar Rana** for always standing by me and supporting my decisions and doing everything possible for my happiness. He is the greatest blessing in my life, without his support I would not have reached thus far. My little son **Lakshya Pratap Rana** who makes me feel so happy with his innocent charming smile has been a pillar of support and comfort to me. I would also like to express my thanks to my father, **Sh. Satya Narayan** and mother, **Smt. Shauntala Devi** for their prayers, unconditional love and support. This thesis would not have been possible without their love and continuous support. I also owe my sincere appreciation to **my brothers** and **their families** for their endless love, confidence, and support. I am also grateful to my **parents-in-law** and my **sisters-in-law** and **brothers-in-law** for their love, patience, sincere support and encouragement.

I thank all those who genuinely offered best wishes and help. I express my gratitude and appreciation for everyone whom I might have thanklessly missed to remember and who have contributed towards completion of the work.

Dated: 07.11.16

(Poonam Panwar)

ABSTRACT

The scheduling and mapping of the precedence-constrained task graph to processors is considered to be the most crucial NP-complete problem in parallel and distributed computing systems. Several algorithms including genetic algorithms have been developed to solve this problem [18], [47], [124], [136]. A common feature in most of these has been the use of chromosomal representation for a schedule. However, these algorithms are monolithic, as they attempt to scan the entire solution space without considering how to reduce the complexity of the optimization process. In the case of multiprocessor scheduling problems there is still no optimum scheduling algorithm available in literature that can be applied to every type of problems that can be represented using Directed-Acyclic-Graphs of job pool. There is still a need for an efficient algorithm that can results in minimum execution time and at the same time making maximum utilization of the resources. Keeping this in view, in the present thesis we have focused on developing a genetic based approach for minimizing the schedule length (makespan) of tasks as well as maximizing the utilization of the resources.

Estimating the reliability of a software under development can help managers to make release decisions during the testing stage itself. Several methods have been proposed in literature to estimate the defect content using a vast variety of software reliability growth models (SRGMs) [3], [53], [60], [107], [113]. SRGMs have certain underlying assumptions which are usually not met fully in practice. However, empirical evidence has shown that many SRGMs are quite robust despite these assumption violations. The problem is that, because of assumption violations in practice, it is often difficult to decide in a given situation which model to apply in practice. Keeping this in mind we propose in the present thesis a method for selecting an appropriate SRGM to make release decisions. The proposed method provides guidelines on how to select an appropriate SRGM which may be used to decide on the best model to be used in practice for estimating the likely release date during testing stage itself and estimate the cost of the software under development.

Thesis consists of ten chapters. Chapter 1 is introductory in nature. Chapters 2 to 4 deal with multiprocessor scheduling problems, chapters 5 to 9 deal with optimal software reliability problems.

Chapter 1 underlines the main objectives and motivations behind carrying out the research reported in this thesis. A brief review of the relevant literature available on the subject has also been given here. The chapter closes with a brief summary of the work presented in the thesis.

In chapter 2, a genetic algorithm based approach has been proposed for multiprocessor task scheduling to minimize the makespan of tasks represented using Directed acyclic graphs. The algorithm uses suitably designed encoding and crossover approaches in assigning priorities dynamically to the tasks of the task graph. The proposed encoding and priority assignment techniques are simple and easy to implement. The proposed algorithm for minimizing the total processing time i.e. makespan makes use of the heterogeneous earliest finish time heuristic algorithms. This heuristic based task to processor mapping technique is used to search for a solution in order to minimize makespan without violating precedence constraints and accelerate convergence speed of the proposed algorithm. The proposed algorithm has five components which are executed in a sequential order for a given number of iterations to achieve the optimal makespan. We have applied the developed algorithm to a total of 20 problems listed in appendix A. Out of these 7 are of homogeneous type and 5 of heterogeneous type. All these problems have been taken from literature. The remaining eight (which are comparatively of larger size) are randomly self-generated. The number of tasks in these problems varies from 9 to 120 and the number of processors varies from 2 to 6. Our simulation results show that the proposed algorithm yields results which are generally better than those given in literature like DCP (Dynamic critical Path), PETS (Performance effective task scheduling), HEFT (Heterogeneous Earliest Finish Time), PMC (Priority based multi chromosome) and BGA (Basic Genetic Algorithm). Moreover in most of the cases the obtained results are quite close to results obtained using exhaustive search.

In chapter 3, we have further modified our algorithm developed in chapter 2 by adding one more fitness function to further improve it's performance so as to achieve load balancing as well. The two fitness functions have been applied one after the other. The first fitness function is concerned with minimizing the total execution time (schedule length), and the second with the maximizing the load balance on each processor. The developed algorithm is tested on a total of fifteen problems taken from literature and given in appendix A. The performance of the proposed algorithm is also compared with the results of traditional heuristic scheduling techniques like HEFT, BGA and DCP. Our results show that the proposed algorithm outperforms the algorithms which are commonly used for this purpose.

In Chapter 4, we focus on the problem of determining the optimal number of processors which are needed in a multiprocessor (homogeneous as well as heterogeneous) system so as to minimize the overall system cost. Results obtained in the case of homogeneous multiprocessor systems show that for a problem of fixed size with the increase in the number of processors, the execution time first decreases upto a certain stage and after that the speedup becomes zero. On the basis of this study it has been observed that 2 to 3 processors are in general sufficient for small size scheduling problems of upto 9 tasks on and 4 processors are sufficient for problems of size from 10 to 40 tasks. In fact not more than 8 processors are needed for problems of size up to 120 tasks. In the case of heterogeneous multiprocessor systems it is observed that one hardly needs more than three processors for executing medium size problems of upto 18 tasks and not more than 4 processors for executing problems of size up to 40 tasks. Infact in general it is observed that not more than 6 processors are required for problems up to 120 tasks.

In chapter 5, a method for selecting the most appropriate reliability growth model that best fits the available detected fault data midway during its testing stage itself is proposed. In order to select an appropriate software reliability growth model, various comparison criteria have been proposed in literature to compare models quantitatively. Most of these take into account more than ten parameters.

Our study has shown that the following relatively simple criteria can be used to rank competing software reliability models.

$$Rank\ Index_j = \frac{1}{2}\left[\frac{RSq_j}{max_j^n(RSq_j)} + \frac{min_j^n(RMSE_j)}{RMSE_j}\right].$$

where symbols have their usual meanings explained in chapter.

Competing models be then ranked in descending order of this rank index value, rank 1 being assigned to the model with the highest rank index value. In case there is a tie (two models getting close rank index values) then both be assigned same rank. For testing the effectiveness of the proposed criteria, it has been applied on ten datasets taken from literature and listed in appendix B. A comparison of the results obtained with corresponding results available in literature shows that the proposed approach is in general simpler and quite effective and faster in selecting the appropriate SRGM to depict the behavior of the actual observed failure data.

Whereas in chapter 5 we confined ourselves to sixteen software reliability growth models, in **chapter 6**, the method proposed in chapter 5 is applied to select an appropriate model from a set of generalized software reliability growth models considering perfect and imperfect processes during the testing and debugging stages. We have used the proposed approach on eleven real software failure datasets taken from literature and given in appendix B to compare its relative performance on some of the currently used perfect and imperfect debugging models.

In **chapter 7,** a genetic algorithm based technique has been proposed to estimate the unknown parameters of non-homogeneous Poisson process (NHPP) software reliability growth models (SRGMs). The objective function used in this case is: $min\ J = \sqrt{\sum_{t=0}^{n}[m(t) - \mu(t)]^2}$ where $m(t)$ is the actual number of observed failure, $\mu(t)$ is the estimated number of failures. The technique is used to estimate the unknown parameters of ten NHPP SRGMs and use it to rank ten data sets listed in appendix B. The performance of proposed technique is compared with results obtained using least squared estimation technique. The results show that it works well for estimating the unknown parameters of SRGMs, and it selects an appropriate SRGM faster with precision.

In chapter 8, a method for predicting the release date of software during its testing stage is proposed. The method is based on selecting the most appropriate reliability growth model that best fits the available detected fault data midway during its testing stage and then using it to predict the likely release date of the software under development. The selected model can then be used to estimate the total number of expected errors in the software. By specifying the reliability level to be achieved before release one can also estimate as to how much additional testing time will be needed in order to achieve the specified value of reliability. The proposed method is tested on ten real datasets given in appendix B. The results show that in most of the cases prediction of release time midway during the testing itself is quite close to actual release date. A comparison of results with other approaches currently in use shows that proposed approach in general is more effective and can be used to estimate the likely release date much earlier.

In chapter 9, an effort has been made to estimate the cost of software under development for achieving desired level of reliability. The main objective being to minimize the cost being incurred to achieve desired reliability level. The proposed method has been tested on a set of eleven real datasets collected from literature and given in appendix B. The comparison of with earlier methods show that the method is able to reasonably estimate the cost of software under development.

Conclusions based on the present study and scope for possible future work in this field are finally drawn in the concluding **chapter 10**.

TABLE OF CONTENTS

CHAPTER 1

CHAPTER 1

INTRODUCTION

This chapter is introductory in nature. Whereas in the section 1.1 we first consider the multiprocessor scheduling problems and in section 1.2 we present the overview of software reliability. A brief survey of the literature available on the subject is presented in section 1.3. A summary of the work presented in the succeeding chapters of this thesis is finally presented in section 1.4.

1.1 MULTIPROCESSOR SCHEDULING PROBLEM

Advances in hardware and software technologies have led to increased interest in the use of parallel and distributed systems for database management, defense and large-scale commercial applications. The operating system and management of the parallel tasks constitute integral parts of the parallel and distributed environments. One of the biggest issues in such systems is the development of effective techniques for the distribution of the tasks of parallel programs on multiple processors [28].

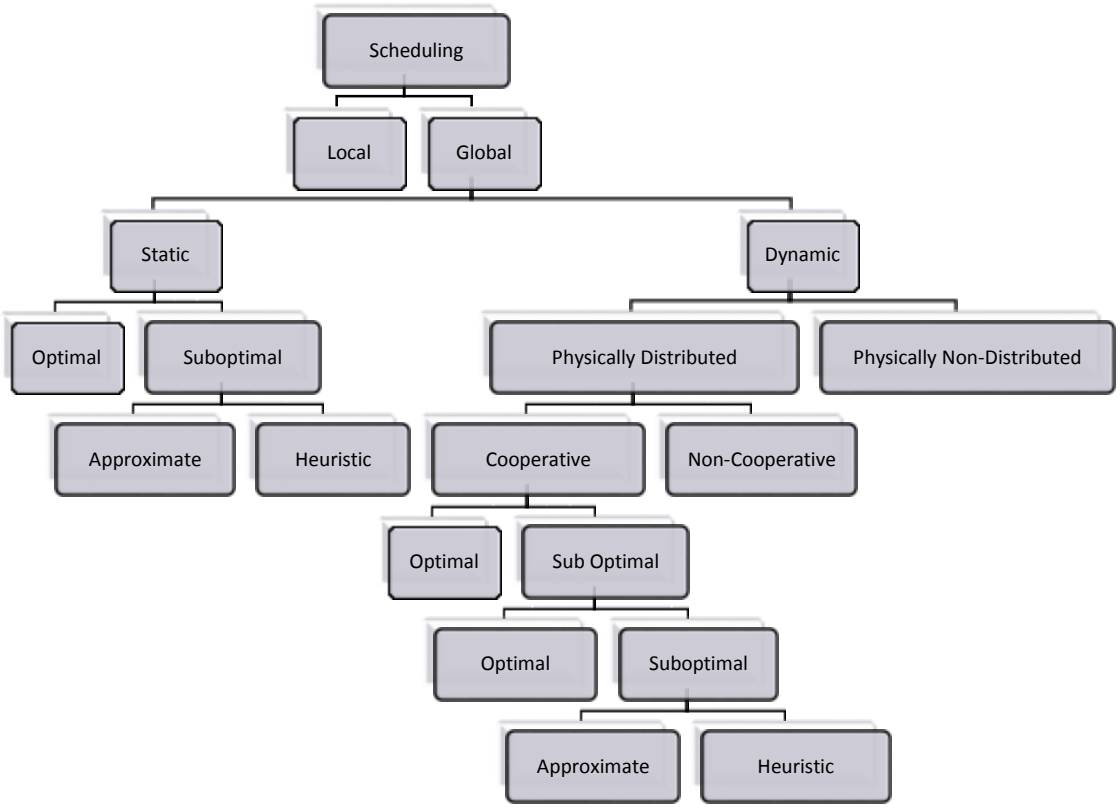

Fig 1.1: Classification of scheduling methods [29]

The problem is as to how to schedule the tasks among processors to achieve performance goals, such as minimizing execution time and/or maximizing resource utilization. From a system's point of view, this distribution choice becomes a resource management problem and is an important factor during the design phases of multiprocessor systems. Task scheduling methods are typically classified into several subcategories as depicted in fig 1.1.

Local scheduling performed by the operating system of a system consists of the assignment of tasks to the available processor. Global scheduling on the other hand, is the process of deciding where to execute a task in a multiprocessor system. Global scheduling may be carried out by a single central authority, or it may be distributed among the processors. The global scheduling methods, are classified into two major groups: static scheduling and dynamic scheduling also referred as dynamic load balancing.

1.1.1 Static Scheduling

A task is always executed on the processor to which it is assigned. In other words static scheduling methods are nonpreemptive in nature. Typically, the goal of static scheduling is to minimize the overall execution time (makespan) of a parallel program with multiple tasks. With this goal in view static scheduling attempts to:

- estimate the task sequence, execution times, and communication delays;
- allocate tasks to processors.

Major advantage of static scheduling methods is that all the overhead of the scheduling process is incurred at compile time, resulting in a more efficient execution time environment compared to dynamic scheduling methods. However, static scheduling suffers from many disadvantages. Static scheduling methods can be classified into optimal and suboptimal. Perhaps one of the most critical shortcomings of static scheduling is that, in general, generating optimal schedules is an NP-complete problem. NP-completeness of optimal static scheduling, with or without communication cost considerations, has been proven in the literature [29].

Since the goal of a scheduling method is to minimize the makespan of a set of tasks, the scheduler must decide which task should be assigned to which processor so that the overall completion time is minimized. In this case, the optimum schedule will ensure that the processing loads are assigned to all processors with no unnecessary delay periods. However, this problem is NP-complete as it can be easily mapped to the well-known, NP-complete "set-partitioning" problem. Since reaching optimal static schedules is an NP-complete problem, most of the research in this area has been focused on determining suboptimal solutions. The methods used are classified into approximate and heuristic approaches. In case of approximate suboptimal static scheduling methods, the solution space is searched in either a depth-first or a breadth-first manner. However, instead of searching the entire solution space for an optimal solution, the algorithm stops when an acceptable solution is reached.

Heuristic methods are based on rules-of-thumb to guide the scheduling process in the right direction to reach a "near" optimal solution. For example, the length of a critical path for a task is defined as the length of one of several possible longest paths from that task, through several intermediate and dependent tasks, to the end of the program. No concurrent program can complete its execution in a time period less than the length of its critical path. A heuristic scheduling method may take advantage of this fact by giving a higher priority in the scheduling of tasks with longer critical path lengths. The idea is that by scheduling the tasks on the critical path first, we have an opportunity to schedule other tasks around them, thus avoiding the lengthening of the critical path. It may be noted that there is no universally accepted standard for defining a "good" solution or a degree of "nearness" to an optimal solution. The researchers often use a loose lower-bound on the execution time of a concurrent program (for example, the length of a critical path), and show that their method can always achieve schedules with execution times within a factor of this lower-bound. In addition to the NP-completeness of optimal general scheduling algorithms, static scheduling suffers from a wide range of problems, most notable of these are:

- The insufficiency of efficient and accurate methods for estimating task execution times and communication delays can cause unpredictable performance degradations. The compile time estimation of the execution time of a program's tasks is often difficult to determine due to conditional and loop constructs, whose condition values or iteration counts are unknown before execution.
- Estimating communication delays at compile time is not practical because of the run-time network contention delays. Existing task scheduling methods often ignore the data distribution issue. This omission causes performance degradations due to run-time communication delays for accessing data at remote sites.
- Finally, static scheduling schemes need be augmented with a tool to provide a performance profile of the predicted execution of the scheduled program on a given architecture. The user can then utilize this tool to improve the schedule or to experiment with a different architecture.

1.1.2 Dynamic Scheduling

Dynamic scheduling is based on the redistribution of tasks among the processors during execution time. This redistribution is performed by transferring tasks from the heavily loaded processors to the lightly loaded processors (called load balancing) with the aim of improving the performance of the application. A typical load balancing algorithm is defined by three inherent policies:

- information policy, which specifies the amount of load information made available to task placement decision-makers;
- transfer policy, which determines the conditions under which a job can be transferred, that is, the current load of the host and the size of the job under consideration (the transfer policy may or may not include task migration, that is, suspending an executing task and transferring it to another processor to resume its execution); and
- placement policy, which identifies the processing element to which a job is to be transferred.

The load balancing operations may be centralized in a single processor or distributed among all the processing elements that participate in the load balancing process. Many combined policies may also exist. For example, the information policy may be centralized but the transfer and placement policies may be distributed. In that case, all processors send their load information to a central processor and receive system load information from that processor. However, the decisions regarding when and where a job should be transferred are made locally by each processor. If a distributed information policy is employed, each processing element keeps its own local image of the system load. This cooperative policy is often achieved by a gradient distribution of load information among the processing elements. Each processor passes its current load information to its neighbors at preset time intervals, resulting in the dispersement of load information among all the processing elements in a short period of time.

A distributed information policy can also be non-cooperative. Random scheduling is an example of non-cooperative scheduling, in which a heavily loaded processor randomly chooses another processor to which to transfer a job. Random load balancing works rather well when the loads of all the processors are relatively high, that is, when it does not make much difference where a job is executed. The advantage of dynamic load balancing over static scheduling is that the system need not be aware of the run-time behavior of the applications before execution. The flexibility inherent in dynamic load balancing allows for adaptation to the unforeseen application requirements at run-time. Dynamic load balancing is particularly useful in a system consisting of a network of workstations in which the primary performance goal is maximizing utilization of the processing power instead of minimizing execution time of the applications. The major disadvantage of dynamic load balancing schemes is the run-time overhead due to:

- the load information transfer among processors,
- the decision-making process for the selection of tasks and processors for job transfers, and
- the communication delays due to task relocation itself.

1.1.3 Future Directions in Dynamic Load Balancing

So far the research and development in the dynamic load balancing area has been focused on the identification and evaluation of efficient policies for information distribution, job transfers, and placement decision making. A greater exchange of information among programmer, compiler, and operating system is still needed. For example the emphasis can be on the development of efficient policies for load information distribution and placement decision-making, because these two areas cause most of the overheads of dynamic load balancing. Although dynamic load balancing incurs overhead due to task transfer operations, a task transfer will not take place unless the benefits of the relocation outweigh its overhead. Thus, future research and development in this area should address itself to:

- hybrid dynamic/static scheduling;
- effective load index measures;
- hierarchical system organizations with local load information distribution and local load balancing policies; and
- incorporation of a set of primitive tools at the distributed operating system level, used to implement different load balancing policies depending on the system architecture and application requirements.

1.2 SOFTWARE RELIABILITY

According to American National Standard Institute (ANSI) Software reliability is defined as the probability of the failure free software operation for a specified period of time in a specified environment. Mathematically reliability is the probability that a system will be functional in the interval from time 0 to time t [89]:

$$R(t) = P(T > t) \qquad (1.1)$$

where T is a random variable denoting the time to failure or failure time and unreliability $F(t)$, of a system is a measure of its failure. It is defined as the probability that the system will fail by time t.

$$F(t) = P(T \leq t) \qquad (1.2)$$

In other words, $F(t)$ is the failure distribution function. If the time to failure of random variable T has a density function $f(t)$, then

$$R(t) = \int_t^{\infty} f(s)ds \qquad (1.3)$$

or, equivalently,

$$f(t) = -\frac{d}{dt}[R(t)] \qquad (1.4)$$

The density function can be described mathematically in terms of T as

$$\lim_{\Delta t \to 0} P(t < T \leq t + \Delta t) \qquad (1.5)$$

This can be interpreted as the probability that the failure time t will occur between the t and the next interval of operation, $t + \Delta t$.

1.2.1 Software Reliability Curve

Software errors have caused human fatalities. The cause of failures ranges from poorly designed user interface to direct programming errors. It does not change over time unless intentionally changed or upgraded. Unlike mechanical parts, software stays as such unless there are problems in its design or in the hardware. Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of software or other unforeseen problems [86]. Over time, hardware exhibits the failure characteristics as shown in fig 1.2 known as bathtub curve.

Software is not susceptible to the environmental maladies that cause hardware to wear out. Therefore, the failure rate curve for software should ideally take the form of the "idealized curve" as shown in fig 1.3. Undiscovered defects will cause high failure rates early in the life of a program. Once these are corrected (possibly without introducing other errors) the curve flattens. In the useful life phase, software will experience a drastic increase in failure rate each time an

upgrade is made. The failure rate levels off gradually, partly because of the defects found and fixed after the upgrade.

Considering the "actual curve" in fig 1.3, during the software's life, software will undergo feature upgrades. For feature upgrades the complexity of software is likely to increased, since functionality of software is enhanced, causing the failure rate curve to spike as shown in fig 1.3. Before the curve return to the original steady state failure rate, another upgrade is requested causing the curve to spike again. Slowly, the minimum failure rate level begins to rise as the software is deteriorating due to upgrade in features. Since the reliability of software keeps on decreasing with increase in software complexity a possible curve is shown in fig 1.4.

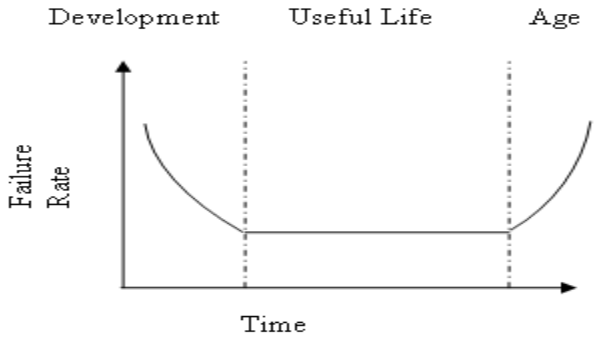

Fig 1.2: The Bathtub Curve [80]

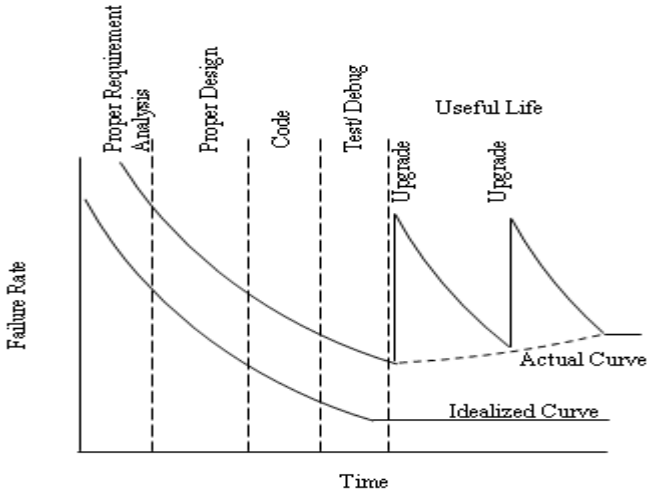

Fig 1.3: Software Reliability Curve [80]

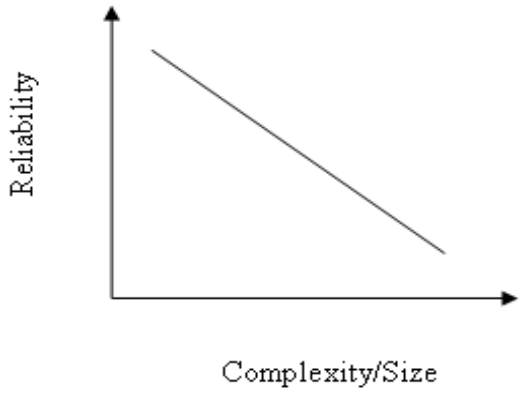

Fig 1.4: Idealized Software Reliability Curve [80]

1.2.2 Software Reliability Growth Models (SRGMs)

Software reliability growth of models measure and predict the improvement in reliability programs as the testing process progress. SRGMs represents the reliability or failure rate of a system as a function of time. The Non-Homogeneous Poisson Process (NHPP) group of SRGMs provide an analytical framework for describing the software failure phenomenon during testing. The main issue in the NHPP SRGMs is to estimate the mean value function of the cumulative number of failures experienced up to a certain point in time. Models commonly included in this group are

- Musa Okumoto [80]
- Goel Okumoto [29]
- S shaped Growth [107]
- Hyperexponential Growth [89]
- Discrete reliability Growth [89]
- Testing effort dependent reliability growth [89]
- Generalized NHPP [89]

With different assumptions, the models result with different functional forms of the mean value function. By definition, the mean value function of the cumulative number of failures ($m(t)$), can be expressed in terms of the failure intensity function of the software as

$$m(t) = \int_0^t \lambda(s)ds \qquad (1.6)$$

The reliability function of the software is

$$R(t) = e^{-m(t)} = e^{-\int_0^t \lambda(s)ds\lambda} \qquad (1.7)$$

Goel Okumoto's NHPP model belongs to this class. Other types of mean value functions are delayed s-shaped, inflection s-shaped, Yamada exponential, Yamada Imperfect, and Pham Zhang etc. An NHPP SRGM is a model that has been used for predicting software reliability. In our present study we have used various NHPP SRGMs.

Reliability is one of the most important quality attributes of software since it quantifies software failures during the development process. However, higher level reliability of software can definitely result in more development cost which, in turn, restricts the software distribution [82]. A strong relationship exists between cost and reliability. In practical engineering, software project managers are required to estimate the cost needed to complete the development of a software system taking into account the required level of reliability. Indeed, the cost should be estimated, as early and accurately as possible, so that the development team can choose the proper techniques in order to address constraints of cost and required reliability. Moreover, estimating the development cost for a software system before being developed is an important foundation for the feasibility study and is the precondition to software system management.

Therefore, after the design phase is completed, it is necessary to estimate the cost of implementation phase based on the architecture generated out of the design phase taking the required level of software reliability into consideration. Several software reliability models have been proposed in the past decades to help software managers and developers to analyze and design systems under such dual and often conflicting constraints of maximizing reliability and minimizing cost. In spite of there being many software reliability models, most of them focus on the cost

estimation during the testing phase of software development [45-46], [67], [91] and software release policies [25], [43], [91], [127], i.e., when to stop testing and deliver the system.

1.3 BRIEF SURVEY OF THE LITERATURE

Problem of efficient scheduling of tasks on multiprocessors has been engaging the interests of researchers and has been studied for quite some period of time. It is an NP complete problem. Approximate solutions to this NP complete problem typically combine traditional search techniques and heuristics. Traditional search techniques require a deterministic search of the solution space, which is computationally exhaustive, whereas a number of heuristic approaches have also been proposed in literature [4], [73], [95].

Classic solutions to this problem have been also provided that use a combination of search techniques and heuristics [20], [24], [132]. Holland [52] and Goldberg [34] provide robust, stochastic solutions for numerous optimization problems. Genetic algorithms are known to provide robust, stochastic solutions for numerous optimization problems. Hou, Ansari and Ren [40] proposed a genetic algorithm that solved the multiprocessor scheduling problem for a directed task graph using simple genetic search techniques in combination with a variety of random selection techniques.

Topcuoglu et al. [115] presented two scheduling algorithms for a bounded number of heterogeneous processors with an objective to simultaneously meet high performance and fast scheduling time which are called Heterogeneous Earliest-Finish-Time (HEFT) algorithm and the Critical-Path-on-a-Processor (CPOP) algorithm. Baskiyar S. and SaiRanga [7] evaluated the performance of a non-preemptive heuristic algorithm called Heterogeneous Critical Node First (HCNF) that statically schedules Directed Acycliced Graphs (DAGs) on heterogeneous multiprocessor systems to minimize the makespan. Using simulations on real applications and benchmark graphs it was shown that HCNF outperforms HEFT significantly in schedule length ratio, speedup and efficiency.

Ilavarasan et al. [48] introduced a Performance Effective Task Scheduling (PETS) algorithm for network of heterogeneous system, which provides optimal results for applications represented by DAGs. Montazeri et al. [77] used genetic algorithm as well as incorporated a local search method called a memetic within the genetic algorithm for a global search. Jelodar et al. [118] proposed a representation to solve the task scheduling problem using Genetic Algorithm (GA).

Hwang et al. [47] designed the encoding mechanism with a multi-functional chromosome that uses the priority representation of the so called Priority-based Multi-Chromosome (PMC). PMC can efficiently represent a task schedule and assign tasks to processors. They proposed priority based GA that shows effective performance in various parallel environments for scheduling problems. Yang et al. [128] presented an uncertain intelligent scheduling algorithm based on an expected value model and a genetic algorithm to solve the multiprocessor scheduling problem in which the computation time and the communication time are given by stochastic variables.

Bohler et al. [11] described the design and implementation of a genetic algorithm for minimizing the schedule length for a general task graph to be executed on a multiprocessor system. Mehrabi et al. [76] proposed a method based on genetic algorithms that incorporates a repair function to guarantee valid assignments during the process of the algorithm. Yellapu, G. and Penmetsa S. K. [129] modelled the problem of scheduling with expected availability of resources. Jain, D. and Jain, S.C. [49] proposed a load balancing real-time periodic task scheduling algorithm for multiprocessor environment.

Okmoto and Goel [17] presents a stochastic model for the software failure phenomenon based on a nonhomogeneous Poisson process. The failure process is analyzed to develop a suitable mean value function for the NHPP; expressions are given for several performance measures. Actual software failure data are analyzed and the results indicate that the model provides a good fit to the observed failure phenomenon.

Musa and Okumoto [80] proposed a logarithmic Poisson execution time model for software reliability measurement. The model incorporates both execution time and calendar time components, each of which is derived. The model is evaluated, using actual data, and compared with other models. The results show that the model predicts the expected failures and hence related reliability quantities as well or better than existing software reliability models, and is simpler than any of the models that approach it in predictive validity.

Goel [30] analyzed a number of analytical models proposed literature for assessing the reliability of a software system. The author presented an overview of the key modeling approaches, provides a critical analysis of the underlying assumptions, and assess the limitations and applicability of these models during the software development cycle. They also proposed a step-by-step procedure for fitting a model and illustrate it via an analysis of failure data from a medium sized real-time command and control software system.

Ohba and Chou [83] discussed the improvement in conventional software reliability growth models by elimination of the unreasonable assumption that errors or faults in a program can be perfectly removed when they are detected. The results show that exponential type software reliability growth models that deal with error counting data could be used even if the perfect debugging assumption were not held, in which case the interpretation of the model parameters should be changed. An analysis of real project data presented by them shows that the imperfect debugging is not a crucial factor in analyzing software reliability data. .

Xie et al. [123] proposed a method for the estimation of software reliability growth in the early stage of testing. Their simulation results shows that their approach is easy to use as the estimation does not require a numerical algorithm. The results are stable when the maximum likelihood estimates are reasonable. Ramani et al. [99] presented the high-level design of a Software Reliability Estimation and Prediction Tool (SREPT), that offers a unified framework containing techniques (including the architecture-based approach) to assist in the evaluation of software reliability at all phases of the software life-cycle.

Stringfellow and Andrews [113] presented an empirical method for selecting SRGMs to make release decisions. The method provides guidelines on how to select among the SRGMs to decide on the best model to use as failures are reported during the test phase. The method applies various SRGMs iteratively during system test. They are fitted to weekly cumulative failure data and used to estimate the expected remaining number of failures in software after release. If the SRGMs pass proposed criteria, they may then be used to make release decisions. The method is applied in a case study using defect reports from system testing of three releases of a large medical record system to determine how well it predicts the expected total number of failures.

Lyu et al. [75] proposed a unified scheme of some nonhomogeneous Poisson process models for software reliability estimation. They described that how several existing software reliability growth models based on Nonhomogeneous Poisson processes can be comprehensively derived by applying the concept of weighted arithmetic, weighted geometric, or weighted harmonic mean. Furthermore, based on these three weighted means, we thus propose a more general NHPP model from the quasi arithmetic viewpoint. In addition to the above three means, the authors formulated a more general transformation that includes a parametric family of power transformations. Under this general framework, they verified the existing NHPP models and derive several new NHPP models.

Huang and Lyu [43] studied the impact of software testing effort & efficiency on the modeling of software reliability, including the cost for optimal release time. This paper presented two important issues in software reliability modeling & software reliability economics: testing effort, and efficiency. First, a generalized logistic testing-effort function that enjoys the advantage of relating work profile more directly to the natural flow of software development is proposed. Secondly, the effects of new testing techniques or tools for increasing the efficiency of software testing is evaluated. The experimental results provides a comprehensive analysis of software based on cost & test efficiency. Moreover, the policy can also help project managers determine when to stop testing for market release at the right time.

Kapur [54] discussed an optimal release time problem for an imperfect fault debugging model considering effect of perfect and imperfect debugging separately on the total expected software cost. A SRGM incorporating the effect of imperfect fault debugging and error generation is also presented. The proposed model is validated on a data set cited from literature and a release time problem is formulated minimizing the expected cost subject to a minimum reliability level to be achieved by the release time using the proposed model. A numerical illustration is given for both type of release problem and finally a sensitivity analysis is performed.

Sharma et al. [107] developed a deterministic quantitative model based on a distance based approach (DBA) and applied it for evaluation, optimal selection, and ranking of SRGMs using sixteen NHPP SRGMs. DBA recognizes the need for relative importance of criteria for a given application, without which inter criterion comparison could not be accomplished. It requires a set of model selection criteria, along with a set of SRGMs, and their level of criteria for optimal selection; and it successfully presents the results in terms of a merit value which is used to rank the SRGMs. The authors used the approach on two distinct, real data sets for demonstration of the DBA method. The result of this study will be a ranking of SRGMs based on the Euclidean composite distance of each alternative to the designated optimal SRGM.

Kapur et al. [53] developed a unified framework for two cases, i.e. the case when failure observation or removal are considered as one testing process (GINHPP-1), and case when failure observation and fault removal processes are considered to be two different testing processes (GINHPP-2). This division of a process into different processes defines the complexity of faults present in software. The more the delay in removal of a fault on its observation, the more complex is the fault. Using this generalized approach, a wide range of incorporating the concept of imperfect debugging and error generation can be developed for different design environments. The results obtained from the models discussed in this paper are quite encouraging, as can be viewed through the numerical illustrations shown in the tables obtained after we performed the estimation on real data sets.

Kapur et al. [58] developed a multi-release software reliability model. This model uniquely takes into account the faults of the release which is under the testing phase. The model has been developed under the assumption that the fault removal process is governed not only by testing time but also by the testing resources expended. With this development structure, a relatively unexplored area of software reliability is investigated. The proposed multi-release two dimensional model is applied to a real data set of four releases. Then a release planning problem is formulated and solved using a GA which minimizes the expected software cost subject to removing a minimum desired proportion of faults from the new version that is to be brought into the market. A numerical illustration is also given for the developed optimal release planning problem.

Yang et al. [127] proposed an architecture based multi objective optimization approach to testing resource allocation. An architecture-based model, in particular a Discrete Time Markov Chain (DTMC) hierarchical state-based model, is used for system reliability assessment, which has the advantage to explicitly consider system architecture and is more computationally tractable, and more flexible to component changes and architecture revision than composite architectural models.

Zahedi and Ashrafi [131] adopted the analytic hierarchy process (AHP) method for modeling the software architecture with cost as the constraints and proposed a model relating to the system reliability maximization. It is fair to say that there has been limited success as yet. Henlander et al. [38] described the relationship between reliability and cost based on reliability allocation approach. They presented two approaches for reliability and cost planning named reliability-constrained cost-minimization (RCCM) and budget-constrained reliability-maximization (BCRM) and three functions for measuring cost-to-attain failure intensity. The modeling frameworks presented focus on cost distribution among software components with a quantified system reliability goal.

Wadekar and Gokhale [117] explored cost and reliability tradeoffs in architectural alternatives by using a genetic algorithm. Huang et al. [46] outlined a

method using logistic testing effort (LTE) function to optimize the reliability allocation and testing schedule for a software system taking into account the reliability growth of its components. Boehm et al. [10] presented a method for evaluating software development cost by using the constructive cost model (COCOMO). Williams [119] discussed a modified approach to calculate the delivery cost of a software product, when warranty is to be provided, with an imperfect debugging phenomenon.

Forbes and Long [25] proposed a reliability investment model that can be used to predict the investment in reliability required to achieve a given amount of reliability improvement by dividing the investment in reliability in terms of average production unit cost. The model proposed is intended to comprise four sub models; only two of them are completed thus far, and more research is required to mature the basic model.

There are many other different methods to address the relationship between software reliability and development cost [41], [45], [68], [98]. Guan et al. [36] proposed a new method to estimate the relationship between software reliability and software development cost taking into account the complexity for developing the software system and the size of software intended to develop during the implementation phase of the software development life cycle.

1.4 THE PRESENT WORK

In the case of multiprocessor scheduling problems there is still a need for optimum scheduling algorithm that can be effectively applied to different types of situations. As in the case of Directed-Acyclic-Graphs of job pool there is still a need for an efficient algorithm that can result in minimum execution time and at the same time maximum utilization of resources. In case of scheduling problems the cost increases with increase in the execution time. If optimal solutions are not provided then there will be less demand of multiprocessor systems in the market.

Similarly in the field of software reliability, softwares are now being used everywhere, so there is a need for developing more efficient and reliable softwares. Though significant work has been done in the field of reliability assessment and reliability modeling yet there is no method available in literature to select a model that can appropriately fits in a variety of situations to address the software reliability related problems. Most of the studies in literature deal with optimal release policies i.e. are used to determine time to deliver the software. But how to design a minimum cost and optimum level of reliability software is still a problem. In the proposed study we intend to address ourselves to both these types of problems.

The present study has been undertaken with a view to find optimal solutions for the multiprocessor scheduling problems as well as achieving maximum possible reliability of software being developed at minimum possible cost.

Chapter wise summary of the work presented in the subsequent chapters of this thesis is as follows:

Thesis consists of ten chapters. Chapter 1 is introductory in nature. Whereas chapters 2 to 4 deal with multiprocessor scheduling problems. Chapters 5 to 9 deal with optimal software reliability problems. **Chapter 1** underlines the main objectives and motivations behind carrying out the research work reported in this thesis. A brief review of the relevant literature available on the subjects has also been given here. The chapter closes with a brief summary of the work presented in thesis.

In chapter 2, we propose a genetic algorithm based approach for minimizing the makespan of multiprocessor task scheduling problems represented with directed acyclic graphs (DAGs). It makes use of suitably designed encoding and crossover approaches in assigning priorities dynamically to the tasks of the task graph. The proposed algorithm has five components which are executed in a sequential order for a specific number of iterations to achieve the optimal makespan. We have applied the developed algorithm to solve a total of 20 test problems listed in appendix A. Our simulation results show that the proposed algorithm yields results which are generally better than those reported in literature using DCP (Dynamic

critical Path), PETS (Performance effective task scheduling), HEFT (Heterogeneous Earliest Finish Time), PMC (Priority based multi chromosome) and BGA (Basic Genetic Algorithm). Moreover in most of the cases the obtained results are comparable with corresponding results obtained using exhaustive search.

In chapter 3, we have further modified our algorithm developed in chapter 2 by adding one more fitness function with a view to improve its perform in achieving load balancing as well. The two fitness functions have been applied one after the other. The first fitness function is concerned with minimizing the total execution time (schedule length), and the second is concerned with the maximizing the load balance on processors. The proposed algorithm is tested on the same set of problems listed in appendix A and three benchmark problems. The performance of the proposed algorithm has also been compared with results obtained using traditional heuristic scheduling techniques like HEFT, BGA and DCP. Our experimental study has shown that in most of the cases the proposed algorithm generally outperforms the conventional algorithms used for this purpose.

Chapter 4, is devoted on determining the optimal number of processors which need to be used in a multiprocessor system (homogeneous as well as heterogeneous) so as to minimize the overall system cost. Results obtained in the case of homogeneous multiprocessor systems show that for a given size problem with the increase in the number of processors the execution time of the problems keeps on decreasing upto a certain stage and after that the speedup becomes practically zero. This study has shown that that in homogeneous case 2 to 3 processors are in general sufficient for small size scheduling problems of upto 9 tasks, 4 processors are sufficient for problems of size from 10 to 40 tasks. In fact not more than 8 processors have been found necessary for problems of size upto 120 tasks. In the case of heterogeneous multiprocessor systems it is observed that one hardly needs more than three processors for executing medium size problems of tasks up to 18 and not more than 4 processors for executing problems of size up to 40 tasks. Infact more than 6 processors were not required for problems up to 120 tasks.

In chapter 5 a method for selecting the most appropriate reliability growth model that best fits the available detected fault data midway during its testing stage is proposed. In order to select an appropriate software reliability growth model, various comparison criteria have been proposed in literature to compare models quantitatively. Most of these take into account more than ten parameters. Our study has shown that the following simple criteria can be used to rank competing software reliability models, where symbols have their usual meanings.

$$Rank\ Index_j = \frac{1}{2}\left[\frac{RSq_j}{max_j^n(RSq_j)} + \frac{min_j^n(RMSE_j)}{RMSE_j}\right]$$

After obtaining the value of this rank index for competing models these are then ranked in descending order of this rank index value, rank 1 being assigned to the model with highest rank index value. In case there is a tie (two top models are getting very close rank index values) then both are assigned the same rank. The method has been applied on ten datasets taken from literature and given in appendix B. A comparison of the obtained results with results available in literature shows that the proposed approach is in general more effective and faster in selecting an appropriate SRGM for a failure dataset.

Whereas in chapter 5 we have restricted ourselves to a sixteen NHPP SRGMs. **In chapter 6**, the method proposed in chapter 5 is applied to select an appropriate models from a set of generalized software reliability growth models based on perfect and imperfect debugging processes during testing stage. We have used the proposed approach on the same set of ten software failure datasets listed in appendix B to compare the effectiveness of some of the currently used perfect and imperfect debugging models.

In chapter 7, a genetic algorithm based technique is proposed to select an appropriate SRGM. The objective function used in this case is: $min\ J = \sqrt{\sum_{t=0}^{n}[m(t) - \mu(t)]^2}$ where $m(t)$ is actual number of observed failure, $\mu(t)$ is the estimated number of failures at the time t during testing. The technique is compared

with least squared estimation technique using ten software reliability growth models both perfect and imperfect. The results show that this technique can be for estimating the unknown parameters of SRGMs and selecting an appropriate SRGM with relatively greater precision and faster convergence.

In chapter 8, a method for predicting the release date of a software during its testing stage is proposed. The method is based on selecting the most appropriate reliability growth model that best fits the available detected fault data midway during its testing stage and then uses it to predict the likely release date of the software under development. The selected model also can estimates the total number of expected errors in the software. By specifying the reliability level to be achieved before release one can also estimate as to how much additional testing is needed in order to achieve an acceptable value of reliability. The proposed method is tested on same ten datasets given in appendix B. The results show that in most of the cases the prediction of release time midway during the testing stage itself was quite close to actual release time. A comparison of the present results with results using approaches available in literature shows that the proposed approach in general is more effective and also faster in deciding the date of release.

In chapter 9, an effort has been made to estimate the cost of software under development for achieving desired level of reliability. First a generalized cost model is used to estimate the cost of all ten datasets listed in appendix B. In second part a genetic algorithm is proposed to analyse the software reliability and cost trade-off. The simulation results shows that the proposed method is able to reasonably estimate the cost and reliability of a software under development.

Conclusions based on the present study and scope for possible future work in this field are mentioned in the concluding **chapter 10**.

CHAPTER 2

CHAPTER 2

TASK SCHEDULING ON HOMOGENEOUS AND HETEROGENEOUS MULTIPROCESSOR SYSTEMS

In this chapter we propose a genetic algorithm based approach for minimizing the makespan of multiprocessor task scheduling problems represented by directed acyclic graphs (DAGs). It makes use of suitably designed encoding and crossover approaches in assigning priorities dynamically to the tasks of the task graph. The proposed algorithm has five components which are executed in a sequential order for a specific number of iterations to achieve the optimal makespan. We have applied the developed algorithm to solve a total of 20 test problems listed in appendix A. Our simulation results show that the proposed algorithm yields results which are generally better than those reported in literature. Moreover in most of the cases the obtained results are comparable with corresponding results obtained using exhaustive search.

2.1 MOTIVATION

Significant research has been done on the problem of allocating tasks to the processors of a distributed computing environment. In more realistic cases, a scheduling algorithm needs to address a number of issues. It should exploit the parallelism by identifying the task graph structure and take into consideration task granularity, computation, and communication costs.

An important problem in distributed computer systems is the task allocation problem. Many heuristic approaches have been proposed in a number of studies [1], [18], [22-23], [47], [26], [63], [69], [96], [100], [102], [105], [114-116], [118], [129], [138]. Most of these provide suboptimal solutions. However, in the case of practical problems, it is difficult to decide as to how appropriate is the obtained solution, because one does not know the exact solution. Execution of safety-critical tasks in real-time systems and parallel computers employ a number of processors that offer

high computing power. These are commercially available these days. This has fueled interest in developing more powerful algorithms for difficult optimization problems. Priority list scheduling method has been also investigated over a period of time for scheduling a set of precedence constrained tasks onto a finite number of identical processors with and without communication overhead to minimize the makespan [61], [74]. Static scheduling of a program can also be represented by a directed task graph on a multiprocessor system to minimize the program completion time. Some of the list scheduling heuristics consider communication costs also. A graph matching approach to optimal assignment of task modules with varying length and precedence relationship in a distributed computing system has been proposed by Wang et al. [118]. Several techniques have been also proposed in the recent past for designing algorithms for optimization of makespan of scheduling problems on multiprocessor systems [9], [63].

Traditional solutions to multiprocessor task scheduling require a deterministic search of the solution space that is computationally and temporally exhaustive [129]. Genetic algorithms [26], simulated annealing [118], particle swarm optimization [96] and dynamic programming [23] based metaheuristics are known to provide robust, stochastic solutions for a variety of optimization problems. These are commonly referred to as evolutionary algorithms [18].

Genetic algorithms are one of the best known heuristics based optimization algorithms. These mimic the process of natural selection and genetics. Genetic algorithms are now being widely used as effective tools in problem solving and optimization [6], [18], [23]. There have also been attempts to solve multiprocessor scheduling problem using genetic algorithms [35]. Genetic algorithms have certain weaknesses particularly with encoding. How to encode solution space of the problem into a chromosome is a key issue for the genetic algorithms. In the recent past various encoding approaches have been proposed for specific problems. These encoding techniques can be applied to only specific types of scheduling problems. Braun et al. [13]; Ritchie and Levine [101], treat the chromosome as a n-dimensional array denoting the n tasks to be allocated. The encoded variable in each element

represents the processor selected to execute the associated task. While such an encoding scheme is simple to implement, it does not consider the order in which the various tasks are processed. As a result evolved schedules may not satisfy the precedence constraints.

Wu et al. [121] considered a representation which encodes task processor pairs and the order in which the pairs appear in the chromosome to determine the order in which the tasks will be performed on each processor. Hwang et al. [47] in 2008 proposed a new encoding mechanism with a multi-functional chromosome that uses the priority representation of the so called priority based multi-chromosome (PMC). PMC can efficiently represent a task schedule and assign tasks to processors. Again the encoding method used for scheduling of jobs on a homogeneous multiprocessor system may not be suitable for a heterogeneous multiprocessor system [31], [66]. The assignment of priority to a task is also important.

On a critical study of the existing techniques available in literature, it was felt that there is still scope for designing improved algorithms for task scheduling. The remainder of this chapter is organized as follows. The formulation of model is described in section 2.2. In section 2.3, the proposed algorithm to minimize the makespan on multiprocessor computing systems is presented. In section 2.4, its working for task scheduling on homogeneous as well as heterogeneous multiprocessor systems is described. The proposed algorithm is next applied to a total of 18 test problems. Out of these 12 are taken from literature and the rest 8 are self-generated and comparatively of larger size. In section 2.5, the comparison of obtained software simulations results have been done with the corresponding results available in literature as well as with exhaustive search results. Conclusions based on the present study are finally drawn in Section 2.6.

2.2 FORMULATION OF MODEL

The general task scheduling problem includes the problem of efficiently assigning the tasks of an application to suitable processors as well as the problem of ordering task execution on each resource. In the general form of task scheduling problem an

application is represented by a Directed Acyclic Graph (DAG) as given in fig 2.1 and table 2.1; which is a generic model of a workflow application consisting of a set of tasks (nodes) among which precedence constraints exist. DAG is represented by $G = (T, E)$, where T is the set of n tasks (nodes) $T = \{T_1, T_2, T_3, \dots, T_n\}$ that can be executed on a set of m available processors $P = \{P_1, P_2, P_3, \dots, P_m\}$ with different or identical processing capabilities. E is the set of directed arcs or edges between the tasks that maintain a partial order amongst them. The partial order introduces partial constraints, i.e. if edge $e_{jk} \in E$, then task T_k cannot start its execution before T_j gets completed. Matrix C of size $n * m$ denotes the execution time, where C_{ji} is the execution time of task T_j on processor P_i.

The task graph is a weighted graph and the weight τ_{jk} of an edge stands for the communication costs between the tasks (the amount of data that must be communicated between them). Mathematical formulation of the problem with optimal task scheduling of a DAG, and with a parallel processing system using m processors is presented in Fig 2.2

The objective is to minimize makespan (f) which is the time of completion of all jobs i.e. $min(f) = \max\{AFT(T_{exit})\}$ subject to (2.1) - (2.3).

$$t_k - p_k - d_{jk} \geq t_j \qquad (2.1)$$

$$T_j \succ T_k \; \forall \, j, k \qquad (2.2)$$

$$t_j \geq 0 \;\; \forall j \qquad (2.3)$$

In the above following symbols have been used

f: makespan

n: number of tasks

m: number of processors

t_j : completion time of task T_j

i: processor index, $i = 1,2, \dots, m$

j: task index, $i = 1,2, \dots, n$

p_k: execution time of task T_k

τ_{jk}: communication cost between Tj and T_k

$\succ$: represents a precedence relation between tasks; $T_j \succ T_k$ means that task T_j precedes task T_k

$pre(j)$: the set of predecessors of task T_j

$$d_{jk} = \begin{cases} \tau_{jk}, if\ task\ T_j\ and\ T_k\ are\ assigned\ to\ different\ processors \\ 0, otherwise \end{cases}$$

2.3 PROPOSED ALGORITHM FOR MINIMIZING MAKESPAN

The proposed algorithm for minimizing the total processing time i.e. makespan makes use of the heterogeneous earliest finish time (HEFT) heuristic algorithm. This heuristic based task to processor mapping technique is used to search for a solution in order to minimize makespan without violating precedence constraints and accelerating convergence speed of proposed algorithm. The proposed algorithm has five components described in subsections 2.3.1 to 2.3.5 and are executed in a sequential order for a given number of iterations to achieve the optimal makespan. It differs from the earlier algorithms based on genetic algorithms approach as it uses a new chromosome representation scheme and new crossover and mutation operators that can be applied for both homogeneous and heterogeneous multiprocessor systems.

2.3.1 Chromosome Representation

One of the fundamental and important tasks in the design of a GA is devising an encoding mechanism for representing search nodes as chromosomes. It is desirable that any chromosome should determine a schedule uniquely. For this purpose we use a chromosome structure with n genes, which represents a possible solution of the task scheduling problem. The chromosome structure contains integers $1, 2, \ldots, n$ in a random order representing a priority queue of the n tasks, as shown in Fig 2.3.

Each gene of the chromosome as shown in fig 2.3 corresponds to one of these subtasks in a DAG application, and the order of genes in this queue represents their execution order. Moreover, the order of the subtasks in the chromosome should be a valid topological order where the entry node is placed at the beginning of the

molecule, and the exit node is placed at the end. In order to assure that a queue of integers should be feasible task schedule, all the subtasks in a DAG should have been scheduled and the schedule should satisfy the precedence relations. For generating an initial gene pool of k_c chromosomes the following procedure is used.

Input: set of tasks to be scheduled n, number of processors m and number of genes j (this is the number of tasks in given DAG).

Output: chromosome string $v(j)$.

begin

$v(1) = T_1$;

for $i = 1$ *to* k_c

for $j = 2$ *to* $n - 1$

$t = \ random\ [2, n-1]$;

$v(j) = \ T_t, if\ pre\ (T_t)\ already\ exists\ in\ chromosome$;

end for

end for

$v(n) = T_n$;

end

2.3.2 Mapping of tasks to Processors and Fitness Function

The fitness of a chromosome is an indicator of the quality of the solution it represents. In a given task graph, a task without any parent is called an entry task and a task without any child is called an exit task. In proposed algorithm it is assumed that the task executions of a given application are nonpreemptive. When the encoding of chromosomes is complete then the earliest start time (EST) and the earliest finish time (EFT) of task T_j on processor P_i is calculated. For the entry task T_1 the EST on each processor is considered 0. Moreover EFT value on each processor is the execution time of T_1 on that processor. For example considering DAG shown in fig 2.1 and its cost matrix as given in table 2.1; $EST(T_1, P_1) = 0$, $EFT(T_1, P_1) = 5$; $EST(T_1, P_2) = 0$, $EFT(T_1, P_2) = 3$; $EST(T_1, P_3) = 0$ and $EFT(T_1, P_3) = 4$. For other tasks the EST and EFT values are computed recursively using (2.4) and (2.5).

$$EST(T_i, P_j) = \max\{avail[P_j], \max(AFT(T_k + d_{ki}))\} \qquad (2.4)$$

$$where\ T_k \in pre(T_i)\ ,$$

$$EFT(T_i, P_j) = P_{ij} + EST(T_i, P_j) \qquad (2.5)$$

where $EST(T_i, P_j)$ is the earliest start time of T_i on processor P_j, $avail[P_j]$ is the earliest time at which processor P_j is ready for task execution. If T_k is the last assigned task on processor P_j, then $avail[P_j]$ is the time at which processor P_j completed the execution of the task T_k and is ready to execute another task. The inner max block in (2.4) returns the ready time, i.e., the time when all the data needed by T_i has arrived at processor P_j, $pre(T_i)$ is the set of immediate predecessor tasks of task T_i and d_{ki} is the communication cost (it is 0 if T_k and T_i are assigned to same processor otherwise it is the value present at edge e_{ki} in DAG. In order to compute the EFT of a task T_i, all immediate predecessor tasks of T_i must have been scheduled. EFT of task T_i is calculated by adding the value of its execution time on processor P_j $(i.e. P_{ij})$ to the value of Earliest Start Time of T_i on same processor $EST(T_i, P_j)$. After calculation of EST and EFT of task T_i on each of the available processor; the processor P_i with minimum EFT is selected to schedule the task and the earliest start time and the earliest finish time of T_k on processor P_i is the actual start time of T_i (AST(T_i)) and the actual finish time of T_i (AFT(T_i)). After all the tasks in a graph are scheduled, the schedule length of the full chromosome string (i.e. the overall completion time) is the actual finish time of the exit task T_{exit}. Finally the schedule length is defined as (2.6), and selected processors are assigned to the chromosome string.

$$makespan(f) = \max\{AFT(T_{exit})\} \qquad (2.6)$$

The evaluation function used for our algorithm is based on the makespan (f) of the schedule. The fitness function used is given in (2.7), which has to be maximized.

$$evaluate(V_k) = 1/f^k\ , \quad k = 1,2,3, \dots, k_c \qquad (2.7)$$

where f^k is the makespan of the k^{th} chromosome and k_c is the total number of chromosomes.

2.3.3 Selection

Selection process is used to select the parent chromosomes for crossover. In the proposed algorithm the binary tournament selection procedure [66] has been used. In the binary tournament selection, a pair of individuals is selected randomly from the archive. Thereafter, the selected pair of individuals will enter a tournament where the chromosome with the fitness value as given in (2.4) is selected for reproduction. This procedure is performed until the mating pool is filled to preserve the original population size.

2.3.4 Crossover

Crossover process produces new 'individuals' that have portions of genetic material of both parent's. However commonly used single/double point crossover is not appropriate for present type of scheduling problems because the duplication and omission of vertices can in some cases produce infeasible sequences in the offspring. Therefore in the proposed crossover procedure, after selecting two chromosomes as father and mother, we choose a crossover point (for example T_9 in Fig 2.4) from the set $\{T_1, T_2, \ldots, T_n\}$ and copy the left portion of father (including crossover point) in left segment of offspring. In the right segment of the offspring, the tasks included are those of the mother, which were not in the copied left portion of offspring (see Fig 2.4). This ensures that the new topological order of tasks does not violate the precedence constraints of DAG.

2.3.5 Mutation

Mutation is necessary for inserting new characteristics that are not present in the current chromosomes. In our proposed algorithm the swap mutation operator is used where two positions are selected at random and their contents are swapped. Random swapping is used as a mutation operation, which is swapping two randomly selected genes. However mutation is performed in a manner that ensures that the generated offspring does not violate the prescribed precedence order of DAG. The mutation process is shown in Fig 2.5.

The crossover rate and mutation rate can be specified by user and are used as the probabilities at which the parents can crossbred and offspring are mutated. In our case crossover is always performed so its probability is one. For mutation probability between 0.6 to 0.8 seemed more appropriate in our case. The algorithm is terminated if there is no change within the tolerance in the objective function value during 100 consecutive number of iterations or a total of 1000 iterations whichever is earlier. It uses parentwise priority (any node whose predecessor has been executed can be scheduled on any of available processor for execution). Therefore the assignment of priorities is not static and hence yields better results.

2.4 IMPLEMENTATION OF THE PROPOSED GENETIC ALGORITHM

We have tested the performance of proposed algorithm using both homogeneous as well as heterogeneous multiprocessor systems. Homogeneous multiprocessor systems have processors with same processing capability i.e. execution time of a task T_j is same for each processor. However in case of heterogeneous multiprocessor systems the processing capability of each processor is different. We have applied the proposed algorithm to solve a total of 20 problems. Out of these 7 of homogeneous type and 5 of heterogeneous type have been taken from literature (listed in appendix A) and the remaining (which are of comparatively of larger size) are randomly self-generated. The number of tasks in these problems varies from 9 to 120 and number of processors varies from 2 to 6. These are listed in Appendix A for ready reference.

2.4.1 Implementation on Homogeneous Multiprocessor Systems

We demonstrate the working of our proposed algorithm on a set of total seven homogeneous multiprocessor problems taken from literature. In order to justify the application and usefulness of the proposed algorithm on homogeneous multiprocessor systems we apply it to seven examples taken from literature.

Example 1: This is a homogeneous multiprocessor system with eighteen executable tasks $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}\}$ and P={P_1, P_2} given in example 1 of Appendix A.

The tasks are initially encoded in a chromosome string using the encoding procedure given in section 2.3.1. In this process starting node (node without predecessor) is assigned the first location in array. If it is single then it is statically assigned else it is randomly chosen from the set of all nodes that have no predecessor. In this example two processors are used. The first node is T_1 because it is the only node that has no predecessor. The EST for T_1 on both the processors $P_1$and P_2 is considered 0 because it have no predecessor and the EFT of task T_1 on each of the processor is equal to the execution time of T_1 on each processor; which is 80 for both the processors. Therefore

$$EFT(T_1, P_1) = 80 + 0 \qquad (2.8)$$

$$EFT(T_1, P_2) = 80 + 0 \qquad (2.9)$$

Now the task T_1 can be assigned randomly to any processor because both the processors have same EFT. Assume T_1 is assigned to processor P_1 then AFT of T_1 is 80. Next we select the second task randomly from chromosome string i.e. T_3, and calculate the EST and EFT for task T_3 on the processors $P_i, i = 1,2$; using (2.4) and (2.5). Task T_3 is assigned on processor P_1 as given in Table 2.2. In the similar way, we repeat the process to calculate the EST and EFT of remaining tasks on all the processors, and the tasks are assigned to processors using the procedure described in section 2.3.2. Next the total execution time of chromosome is calculated. This is the time it takes from the instant the first task begins execution on a processor to the instant of the completion of the last task.

The makespan for each of the chromosome in population is calculated in similar way. After this the chromosomes are evaluated using (2.7) and the two chromosomes having highest fitness value are selected for crossover. One point crossover is applied to the selected chromosomes to generate the new chromosomes (crossover point is selected randomly). Mutation operator is applied with specified probability to a randomly selected chromosome. The type of mutation used is a swap mutation as explained in section 2.3.5. The makespan of the new chromosome is computed and compared with makespan of the parent chromosomes. If it's better

than the earlier one then it participates in genetic process otherwise it is discarded and the process continues. The results obtained after 100 iterations is 460 and after 200 iterations is 440. Final result obtained for DAG given is summarized in Table 2.2.

Table 2.2 shows that $makespsn = \max\{ECT(T_{18})\} = 440$; i.e. the makespan of the tasks schedule can be defined as the time it takes from the instant the first task begins execution to the instant at which the task completes execution is 440 units of time. The algorithm has been applied to a total of seven homogeneous problems taken from literature and their corresponding results are presented in Table 2.2 to 2.8 respectively.

2.4.2 Implementation on Heterogeneous Multiprocessor Systems

Heterogeneous multiprocessor systems have processors with different processing capabilities i.e. execution time of a task T_j is different for each processor. In this section we explain the working of the proposed algorithm in heterogeneous environment on a total of 5 examples taken from literature. Starting chromosome initialization, mapping of tasks to processors and calculating fitness is performed as described in section 2.3. The results of various examples taken from literature are tabulated in Tables 2.9 to 2.13 for heterogeneous problems given in appendix A.

2.4.3 Application to Self-Generated Problems

The proposed algorithm has also been applied to eight problems which are randomly self-generated. Of these four are homogeneous type and other four are of heterogeneous type. The number of tasks in these problems varies from 25 to 120 and number of processors varies from 4 to 6. The best and worst makespan results obtained in these cases are tabulated in Table 2.14 for a total of 20 trails and 200 iterations each.

2.5 ANALYSIS OF RESULTS

A comparison of makespan obtained using proposed algorithm is done with makespan available in literature and is given in fig 2.6 to 2.7 and table 2.15.

In case of homogeneous problems the makespan obtained for problem 3, 4 and 7 are less than the results suggested in literature and in none case is greater than those suggested in literature. In case of heterogeneous systems the developed algorithm outperforms in all the considered cases. The optimal result obtained is 23 after 150 iterations for problem 1 of heterogeneous type. The problem was also earlier solved by algorithms such as DCP (Dynamic Critical Path), and the optimal result found till now was 28 [72]. In our present case it is 23 only which is less than the smallest reported in literature as per our information. Similarly for other problems the results obtained are minimum as listed in table 2.15 and are same as obtained using exhaustive search except for problem 3 of heterogeneous type. The exhaustive search is also performed to check the efficiency of the proposed algorithm for problems having upto 11 tasks because it requires large amount of time for other problems and is shown in fig 2.6 to 2.7 and table 2.15. In exhaustive search we have generated each possible sequence of tasks and calculated its makespan. The results shows that the proposed algorithm yields makespan equivalent to the exhaustive search in 8 problems out of 9 problems considered from literature.

2.6 CONCLUSIONS

In this chapter we have proposed an algorithm for finding optimal schedule for multiprocessor task scheduling problems that take into account the execution time of tasks on multiple processors as well as the communication cost between the tasks. Our proposed algorithm which is based on genetic algorithm approach is able to handle the problems of encoding, crossover and mutation efficiently witout violating the precedence costarints between tasks. In numerical experimentation we have used several task graphs for the multiprocessor scheduling problem, and compared the results obtained with the proposed algorithm with the corresponding results available in literature using alternative approaches. The proposed algorithm is suitable for finding the minimum makespan for a given task graph. However it is noticed that it requires to be improved further in terms of load balancing on each processor. This we have done in next chapter.

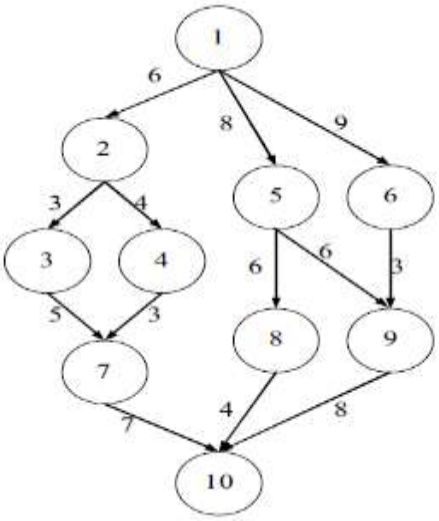

Fig 2.1: DAG with 10 Tasks to Be Scheduled On Heterogeneous Multiprocessor Systems

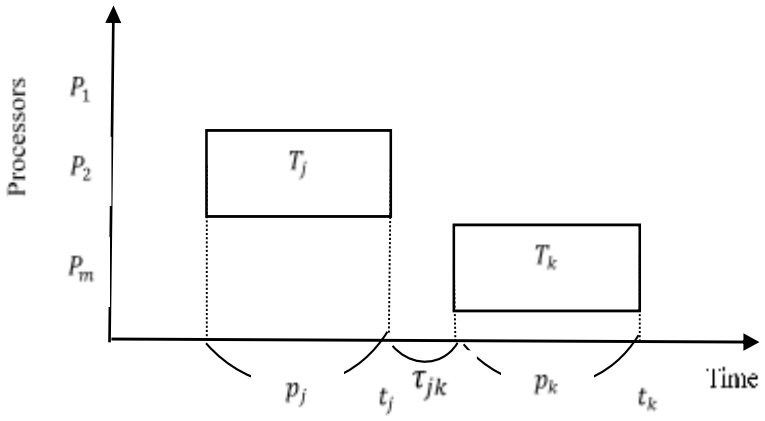

Fig 2.2: Time chart of DAG

T_1	T_2	T_5	T_6	T_9	T_3	T_4	T_7	T_8	T_{10}

Fig 2.3: A sample chromosome structure encoding using Proposed Algorithm for a DAG with ten tasks.

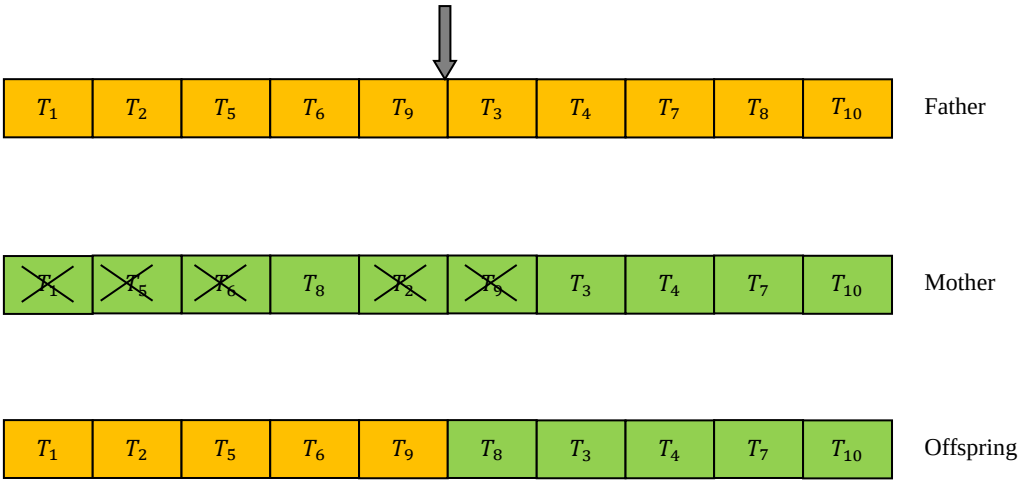

Fig 2.4: Single Point Crossover Operator used in Proposed Algorithm

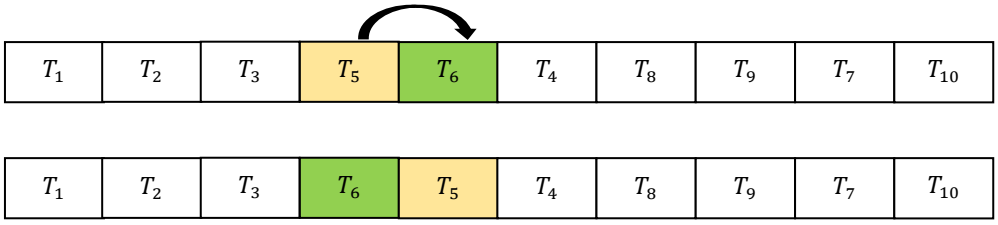

Fig 2.5: Mutation Operator used in Proposed Algorithm

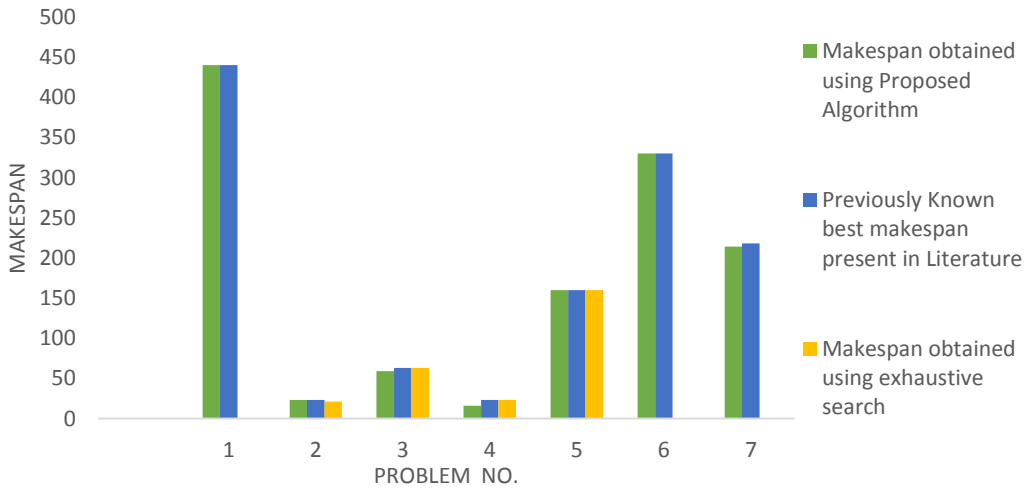

Fig 2.6: Comparison of results obtained using proposed algorithm with suggested results in literature for homogeneous Problems

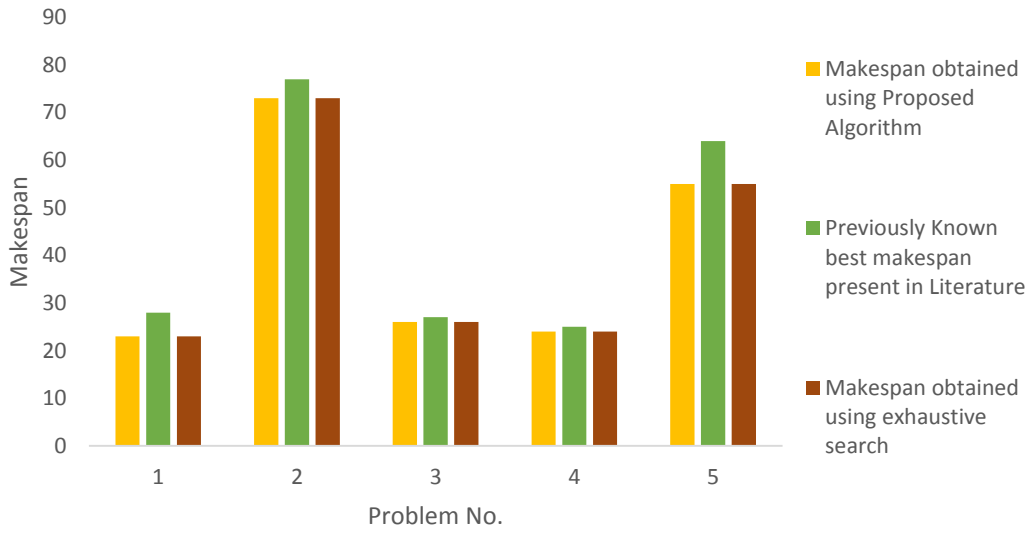

Fig 2.7: Comparison of results obtained using proposed algorithm with suggested results in literature for heterogeneous Problems

Table 2.1: Computation cost matrix of DAG given in Fig 2.1.

Task	P_1	P_2	P_3
1	5	3	4
2	3	2	7
3	3	4	8
4	7	5	3
5	5	2	5
6	3	4	8
7	4	2	3
8	2	3	4
9	4	5	3
10	6	4	5

Table 2.2: Representing the assignment of tasks to the processors P_1 and P_2 for example 1 (homogeneous)

Priority of Task	Entry Task	Processors				Predecessors	Processor Selected
		P_1		P_2			
		EST1	ECT1	EST2	ECT2		
1	T_1	0	80	0	80	Null	P_1
2	T_3	80	120	200	240	T_1	P_1
3	T_7	120	180	200	260	T_1, T_3	P_1
4	T_4	180	220	200	240	T_1	P_1
5	T_9	220	250	300	330	T_4, T_7	P_1
6	T_{12}	250	290	330	370	T_7, T_9	P_1
7	T_5	290	330	200	240	T_1	P_2
8	T_{13}	290	310	410	430	T_7, T_9	P_1
9	T_6	310	350	240	280	T_1	P_2
10	T_{11}	360	390	300	330	T_6, T_7	P_2
11	T_{10}	320	350	330	360	T_5, T_7	P_1
12	T_{14}	350	370	430	450	T_{10}, T_{12}	P_1
13	T_{16}	370	390	450	470	T_{12}, T_{14}	P_1
14	T_2	390	430	330	370	T_1	P_2
15	T_{17}	390	400	510	520	T_{14}, T_{16}	P_1
16	T_{15}	410	430	410	430	T_{11}, T_{12}	P_1
17	T_8	430	460	370	400	T_3, T_7	P_2
18	T_{18}	430	440	510	520	T_{15}, T_{16}	P_1

Table 2.3: Representing the assignment of tasks to the processors P_1 and P_2 for example 2 (homogeneous)

Priority of Task	Entry Task	Processors				Predecessors	Processor Selected
		P_1		P_2			
		EST1	ECT1	EST2	ECT2		
1	T_1	0	2	0	2	Null	P_1
2	T_5	2	7	3	8	T_1	P_1
3	T_2	7	10	6	9	T_1	P_2
4	T_3	7	10	9	12	T_1	P_1
5	T_4	10	14	9	13	T_1	P_2
6	T_6	10	14	13	17	T_2	P_1
7	T_8	T_1	18	17	21	T_2, T_3, T_4, T_5	P_1
8	T_7	18	22	22	26	T_1, T_2, T_3	P_1
9	T_9	22	23	32	33	T_6, T_7, T_8	P_1

Table 2.3 shows that, $makespsn = \max\{ECT(T_9)\} = 23$.

Table 2.4: Representing the assignment of tasks to the processors P_1 and P_2 for example 3 (homogeneous)

Priority of Task	Entry Task	Processors				Predecessors	Processor Selected
		P_1		P_2			
		EST1	ECT1	EST2	ECT2		
1	T_1	0	0	0	0	Null	P_1
2	T_3	0	8	0	8	T_1	P_2
3	T_2	0	20	8	28	T_1	P_1
4	T_4	20	28	8	16	T_1	P_2
5	T_5	20	27	16	23	T_1	P_2
6	T_{10}	20	35	23	38	T_2, T_4	P_1
7	T_6	35	38	23	26	T_3, T_4, T_5	P_2
8	T_8	35	48	26	39	T_2, T_5	P_2
9	T_{11}	39	58	39	58	T_2, T_3, T_8	P_2
10	T_7	35	48	58	71	T_3	P_1
11	T_9	48	60	58	70	T_2, T_3, T_7	P_1
12	T_{12}	60	60	60	60	T_9, T_{10}, T_{11}	P_1

Table 2.4 shows that, $makespsn = \max\{ECT(T_{12})\} = 60$.

Table 2.5: Representing the assignment of tasks to the processors P_1, P_2 and P_3 for example 4 (homogeneous)

Priority of Task	Task	Processors						Predecessors	Processor Selected
		P_1		P_2		P_3			
		EST1	ECT1	EST2	ECT2	EST3	ECT3		
1	n_1	0	2	0	2	0	2	Null	P_3
2	n_2	6	9	6	9	2	5	n_1	P_3
3	n_4	3	7	3	7	5	9	n_1	P_2
4	n_3	3	6	7	10	5	8	n_1	P_1
5	n_7	12	16	12	16	5	9	n_1, n_2	P_3
6	n_6	6	10	7	11	9	13	n_2	P_1
7	n_5	10	15	7	12	9	14	n_1	P_2
8	n_8	10	14	12	16	9	13	n_3, n_4	P_3
9	n_9	18	19	18	19	15	16	n_6, n_7, n_8	P_3

Table 2.5 shows that, $makespsn = \max\{ECT(T_9)\} = 16$.

Table 2.6: Representing the assignment of tasks to the processors P_1, P_2 and P_3 for example 5 (homogeneous)

Priority of Task	Task	Processors						Predecessors	Processor Selected
		P_1		P_2		P_3			
		EST1	ECT1	EST2	ECT2	EST3	ECT3		
1	n_1	0	20	0	20	0	20	Null	P_3
2	n_2	60	90	60	90	20	50	n_1	P_3
3	n_4	30	70	30	70	50	90	n_1	P_2
4	n_3	30	60	70	100	50	80	n_1	P_1
5	n_7	120	160	120	160	50	90	n_1, n_2	P_3
6	n_6	60	100	70	110	90	130	n_2	P_1
7	n_5	100	150	70	120	90	140	n_1	P_2
8	n_8	100	140	120	160	90	130	n_3, n_4	P_3
9	n_9	180	190	180	190	150	160	n_6, n_7, n_8	P_3

Table 2.6 shows that, $makespsn = \max\{ECT(n_9)\} = 160$.

Table 2.7: Representing the assignment of tasks to the processors P_1, P_2 and P_3 for example 6 (homogeneous)

Priority of Task	Task	Processors						Predecessors	Processor Selected
		P_1		P_2		P_3			
		EST1	ECT1	EST2	ECT2	EST3	ECT3		
1	T_1	0	80	0	80	0	80	Null	P_1
2	T_3	80	120	110	150	110	150	T_1	P_3
3	T_5	120	160	110	150	110	150	T_1	P_2
4	T_4	120	160	160	200	110	150	T_1	P_3
5	T_7	120	180	160	220	150	210	T_1, T_3	P_3
6	T_2	180	220	160	220	150	210	T_1	P_1
7	T_9	180	210	210	240	210	240	T_4, T_7	P_3
8	T_6	210	290	160	240	210	240	T_1	P_1
9	T_{12}	210	250	160	240	190	270	T_7, T_9	P_2
10	T_{13}	250	270	280	300	280	300	T_9, T_{12}	P_3
11	T_{10}	270	300	240	270	210	240	T_6, T_7	P_2
12	T_8	270	300	240	270	240	270	T_3, T_7	P_3
13	T_{14}	270	290	280	300	280	300	T_{10}, T_{12}	P_1
14	T_{16}	290	310	310	330	310	330	T_{12}, T_{14}	P_2
15	T_{11}	310	340	240	270	270	300	T_6, T_7	P_3
16	T_{15}	310	330	280	300	290	310	T_{11}, T_{12}	P_1
17	T_{17}	310	320	340	350	340	350	T_{14}, T_{16}	P_1
18	T_{18}	320	330	340	350	340	350	T_{15}, T_{16}	P_2

Table 2.7 shows that, $makespsn = \max\{ECT(T_{18})\} = 330$.

Table 2.8: Representing the assignment of tasks to the processors P_1, P_2, P_3 and P_4 for example 7 (homogeneous)

Priority of Task	Task	Processors								Predecessors	Processor Selected
		P_1		P_2		P_3		P_4			
		EST1	ECT1	EST2	ECT2	EST3	ECT3	EST4	ECT4		
1	T_{10}	0	12	0	12	0	12	0	12	Null	P_2
2	T_1	0	12	12	24	0	12	0	12	Null	P_4
3	T_6	0	9	12	21	0	9	12	21	Null	P_3
4	T_2	0	35	12	47	9	44	12	47	Null	P_1
5	T_5	35	53	12	30	9	27	12	30	Null	P_3
6	T_3	35	41	12	18	27	33	12	18	T_1	P_4
7	T_4	35	39	12	16	27	31	41	45	Null	P_2
8	T_{18}	35	56	16	37	27	48	41	62	Null	P_2
9	T_9	35	61	37	63	27	53	41	67	Null	P_3
10	T_8	35	47	37	49	53	65	41	53	T_4, T_6	P_1
11	T_{14}	47	70	37	60	53	76	41	64	Null	P_2
12	T_{20}	47	83	60	96	53	89	41	77	T_3	P_4
13	T_{11}	47	75	60	88	53	81	77	105	T_1	P_1
14	T_{22}	75	100	60	85	53	78	77	102	T_6, T_{18}	P_1
15	T_{12}	75	78	75	78	78	81	77	80	T_{11}	P_2
16	T_7	75	83	78	86	78	86	77	85	T_2	P_2
17	T_{15}	83	99	78	94	78	94	77	93	T_4, T_{14}	P_4
18	T_{17}	83	90	78	85	78	85	93	100	T_{12}	P_2
19	T_{21}	83	101	85	103	78	96	93	111	T_3, T_{18}	P_3
20	T_{13}	83	96	85	98	96	109	93	106	T_8	P_1
21	T_{29}	96	113	96	113	96	113	96	113	T_1, T_2, T_9	P_4
22	T_{19}	96	113	85	102	96	113	113	130	T_{11}, T_{13}, T_{22}	P_2
23	T_{24}	96	121	102	127	96	121	113	138	T_2, T_{20}	P_3
24	T_{34}	96	108	102	114	121	133	113	125	T_2, T_{10}, T_{12}	P_1
25	T_{25}	121	157	121	157	121	157	121	157	T_{19}, T_{24}	P_2
26	T_{23}	108	147	157	196	121	160	113	152	T_{15}, T_{17}, T_{19}	P_1
27	T_{31}	147	175	157	185	121	149	113	141	T_{10}, T_{29}	P_4
28	T_{16}	147	159	157	169	121	133	141	153	T_{10}, T_{15}	P_3
29	T_{32}	157	179	157	179	157	179	157	179	T_{25}	P_2
30	T_{27}	147	156	179	188	147	156	147	156	T_{27}	P_4
31	T_{26}	157	172	179	194	157	172	157	172	T_7, T_{25}	P_1
32	T_{35}	172	204	179	211	133	165	156	188	T_3, T_{10}, T_{29}	P_3
33	T_{30}	172	179	179	186	165	172	156	163	T_{27}	P_4
34	T_{33}	172	196	179	203	165	189	163	187	T_{30}, T_{31}	P_4
35	T_{28}	172	195	179	202	172	195	187	210	T_{21}, T_{26}	P_1
36	T_{36}	195	217	179	201	165	187	187	209	T_{27}, T_{34}	P_3
37	T_{38}	195	229	187	221	187	221	187	221	T_{32}, T_{33}, T_{35}	P_4
38	T_{40}	221	237	221	237	221	237	229	145	T_{38}	P_3
39	T_{37}	195	208	187	200	237	250	229	242	T_{25}, T_{36}	P_2
40	T_{39}	200	214	200	214	2337	251	229	243	T_{23}, T_{28}, T_{37}	P_1

Table 2.8 shows that, $makespsn = \max\{ECT(T_{40})\} = 214$.

Table 2.9: Representing the assignment of tasks to the processors P_1, P_2 and P_3 for example 1 (heterogeneous)

Priority of Task	Task	Processors						Predecessors	Processor Selected
		P_1		P_2		P_3			
		EST1	ECT1	EST2	ECT2	EST3	ECT3		
1	T_1	0	5	0	3	0	4	Null	P_2
2	T_2	9	12	3	5	9	16	T_1	P_2
3	T_5	11	16	5	7	11	16	T_1	P_2
4	T_6	12	15	7	11	12	20	T_1	P_2
5	T_9	14	18	11	16	14	17	T_1	P_2
6	T_3	8	11	16	28	8	16	T_5, T_6	P_1
7	T_4	11	18	16	21	9	12	T_2	P_3
8	T_7	15	19	16	18	16	19	T_2	P_2
9	T_8	13	15	18	19	13	17	T_3, T_4	P_1
10	T_{10}	25	31	19	23	25	30	T_7, T_8, T_9	P_2

Table 2.9 shows that, $makespsn = \max\{ECT(T_{10})\} = 23$.

Table 2.10: Representing the assignment of tasks to the processors P_1, P_2 and P_3 for example 2 (heterogeneous)

Priority of Task	Task	Processors						Predecessors	Processor Selected
		P_1		P_2		P_3			
		EST1	ECT1	EST2	ECT2	EST3	ECT3		
1	T_1	0	14	0	16	0	9	Null	P_3
2	T_2	27	40	27	46	9	27	T_1	P_3
3	T_3	21	32	21	34	27	46	T_1	P_1
4	T_6	32	45	23	39	27	36	T_1	P_3
5	T_5	32	44	20	33	36	46	T_1	P_2
6	T_4	32	45	33	41	36	53	T_1	P_2
7	T_9	64	82	43	55	64	84	T_2, T_4, T_5	P_2
8	T_8	68	73	55	66	68	82	T_2, T_4, T_6	P_2
9	T_7	32	39	66	81	55	66	T_3	P_1
10	T_{10}	77	98	66	73	77	93	T_7, T_7, T_7	P_2

Table 2.10 shows that, $makespsn = \max\{ECT(T_{10})\} = 73$.

Table 2.11. Representing the assignment of tasks to the processors P_1, P_2, P_3 and P_4 for example 3 (heterogeneous)

Priority of Task	Task	Processors								Predecessors	Processor Selected
		P_1		P_2		P_3		P_4			
		EST 1	ECT 1	EST 2	ECT 2	EST 3	ECT 3	EST 4	ECT 4		
1	T_1	0	4	0	4	0	4	0	4	Null	P_4
2	T_4	6	9	6	9	6	9	4	7	T_1	P_4
3	T_3	6	10	6	12	6	10	7	14	T_1	P_1
4	T_2	10	15	6	11	6	11	7	12	T_1	P_3
5	T_7	10	15	9	17	11	16	7	12	T_4	P_4
6	T_5	14	17	14	19	11	14	14	18	T_2	P_3
7	T_6	12	15	12	19	14	16	12	14	T_2, T_3	P_4
8	T_8	10	12	9	13	14	19	14	17	T_4	P_1
9	T_{10}	16	19	16	23	16	21	15	17	T_6, T_7, T_8	P_4
10	T_9	16	21	16	22	16	23	17	22	T_5, T_6	P_1
11	T_{11}	21	26	23	29	23	30	23	31	T_9, T_{10}	P_1

Table 2.13 shows that $makespsn = \max\{ECT(T_{11})\} = 26$.

Table 2.12. Representing the assignment of tasks to the processors P_1 and P_2 for example 4 (heterogeneous)

Priority of Task	Entry Task	Processors				Predecessors	Processor Selected
		P_1		P_2			
		EST1	ECT1	EST2	ECT2		
1	T_1	0	2	0	2	Null	P_1
2	T_2	2	3	3	8	T_1	P_1
3	T_5	3	10	9	14	T_2	P_1
4	T_3	10	12	4	6	T_1	P_2
5	T_7	10	15	6	18	T_3	P_1
6	T_6	15	24	6	11	T_3	P_2
7	T_9	15	18	13	16	T_5, T_6, T_7	P_2
8	T_4	35	41	17	20	T_2	P_2
9	T_8	19	22	26	28	T_4, T_5, T_6	P_1
10	T_{10}	22	24	23	25	T_8, T_9	P_1

Table 2.15 shows that, $makespsn = \max\{ECT(T_{11})\} = 24$.

Table 2.13. Representing the assignment of tasks to the processors P_1 and P_2 for example 5 (heterogeneous)

Priority of Task	Entry Task	Processors				Predecessors	Processor Selected
		P_1		P_2			
		EST1	ECT1	EST2	ECT2		
1	T_1	0	15	0	22.5	Null	P_1
2	T_0	15	19	0	6	Null	P_2
3	T_3	15	28	6	25.5	T_0	P_2
4	T_6	15	23	25.5	37.5	T_0	P_1
5	T_2	23	27	25.5	31.5	T_1	P_1
6	T_8	27	39	25.5	43.5	T_3	P_1
7	T_7	39	43	25.5	31.5	T_1	P_2
8	T_5	39	46	31.5	41.5	T_1	P_2
9	T_4	39	49	41.5	56.5	T_1	P_1
10	T_{10}	49	58	41.5	55	T_2, T_3, T_6, T_8	P_2
11	T_9	49	55	55	64	$T_2, T_4, T_5, T_6, T_7, T_8$	P_1

Table 2.17 shows that, $makespsn = \max\{ECT(T_9)\} = 55$.

Table 2.14. Results obtained using proposed algorithm of randomly self-generated problems.

Problem No.	No. of Tasks	No. of Processors	Makespan obtained using proposed algorithm	
			Best	Worst
Homogeneous Multiprocessor Scheduling Problems				
1	25	4	342	342
2	40	4	206	209
3	80	5	600	601
4	120	6	745	745
Heterogeneous Multiprocessor Scheduling Problems				
2	25	3	166	174
3	40	4	153	157
4	80	5	574	576
5	120	6	757	757

Table 2.15. Comparison of results obtained using proposed algorithm with suggested results in literature

Problem No.	No. of Tasks	No. of Processors	Makespan obtained using Proposed Algorithm	Previously Known best makespan present in Literature	Makespan obtained using exhaustive search
Homogeneous Multiprocessor Scheduling Problems					
1	18	2	440	440 [47]	-
2	9	2	23	23 [47]	21
3	12	2	59	63 [14]	63
4	9	3	16	23 [1]	23
5	9	4	160	160 [1]	160
6	18	3	330	330 [70]	-
7	40	4	214	218 [11]	-
Heterogeneous Multiprocessor Scheduling Problems					
1	10	3	23	28 [72]	23
2	10	3	73	77 [48]	73
3	11	4	26	27 [5]	26
4	10	2	24	25 [74]	24
5	11	2	55	64 [19]	55

*Note: Entries in the large braces [ ] in the last column is the reference from where the result has been cited.

CHAPTER 3

CHAPTER 3
LOAD BALANCING IN A MULTIPROCESSOR SYSTEM USING GENETIC ALGORITHM

In this chapter, the genetic algorithm proposed in chapter 2 has been suitably modified to improve its performance in terms of load balancing. To achieve this objective two fitness functions have been used and applied one after the other. Whereas the first fitness function is concerned with minimizing the total execution time (makespan), the second one is concerned with the maximizing the load balance on each processor. The developed algorithm is again tested on the same set of total 12 problems taken from literature as well as on three additional benchmark problems.

3.1 MOTIVATION

Advances in hardware and software technologies have led to increased interest in the use of large scale parallel and distributed systems for database, real time, defense, and large-scale commercial applications. The operating system and management of the concurrent processes constitute integral parts of the parallel and distributed environments. One of the biggest issues in such systems is the development of effective techniques for the distribution of the processes of a parallel program on multiple processors. The problem is how to distribute (or schedule) the processes among processing elements to achieve some performance goal(s), such as minimizing execution time, minimizing communication delays, and/or maximizing resource utilization [29-30]. From a system's point of view, this distribution choice becomes a load balancing problem and should be considered an important factor during the design phases of multiprocessor systems.

The load on the processors is a critical factor. Heavily loaded processors consume more power, which cause higher heat dissipation and can cause more occurrence of faults. A good load balancing algorithm finishes the tasks as early as possible without excessively burdening individual processor, provides good

resource utilization and minimizes task response time [31]. Good load balancing avoids excessive heat dissipation on the processor. Lihua et al. [32] and Zhi-qiang et al. [33] addressed Earliest Deadline First-First Fit (EDF-FF) and Earliest Deadline First- Best Fit (EDF-BF) respectively. EDF-FF and EDF-BF partition tasks using First Fit (FF) and Best Fit (BF) respectively and both use EDF to schedule tasks on individual processor of multiprocessor system after partitioning. FF algorithm starts from first processor and search for the processor which can execute the task without overloading. BF algorithm searches for processor with maximum current load that can accommodate the task without overloading. These algorithms are simple but do not consider load balancing amongst the processors.

Zhang et al. [34] presented a load balancing algorithm Local Border Search Algorithm (LBSA). Scheduling of real-time tasks in multiprocessor environment consists of task allocation stage and task scheduling stage. In task allocation stage, the tasks are sorted by the ascending order of their loads firstly. Allocation is done in phases. In each phase, available processors are sorted in descending order of their current loads and then each processor is assigned one task from sorted task set. In task scheduling stage, EDF (Earliest-Deadline First) algorithm is used to schedule tasks in single processor. This algorithm offers poor load balancing at heavy load. It is observed that EDF-FF and EDF_BF do not provide load balancing whereas LBSA give poor performance at high load conditions [31].

Keeping these facts in view in this chapter a genetic algorithm that while allocating jobs to processors to minimize makespan also tries to maintain load balance is proposed. The remainder of this chapter is organized as follows: Section 3.2 gives a description of the modified genetic algorithm proposed in chapter 2 with an additional load balancing feature. Implementation of the proposed algorithm is given in section 3.3 and its performance is analyzed Section 3.3. Conclusions based on the present study are finally drawn in Section 3.4.

3.2 PROPOSED ALGORITHM FOR LOAD BALANCING AND MINIMIZING MAKESPAN

The main objective of the task scheduling is to minimize the schedule length (makespan), it is found that several solutions can produce the same schedule length, but the load balance between processors might not be satisfied in some of them. The aim of load balance modification is that to obtain a minimum schedule length at the same time to satisfy the load balance requirement. Our proposed algorithm discussed in chapter 2 consists of five steps; starting from chromosome representation, mapping of tasks to processors, selection, crossover and mutation. The fitness function used there is a single evaluation function based on the makespan (f) of the schedule, which has to be maximized. However in the proposed algorithm the solutions have to satisfy two fitness functions; which are applied one after the other. The first fitness function deals with minimizing the total execution time, and the second fitness function is used to satisfy the requirements of load balance between processors. The fitness function for makespan is same as given in chapter 2 but the second function is proposed in [35]. It is calculated as the ratio of the maximum execution time (i.e. schedule length) to the average execution time over all processors [37]. If the execution time of processor P_j is denoted by $E\_time[P_j]$, then the average execution time over all processors is:

$$Avg = \sum_{j=1}^{m} \frac{E\_time[P_j]}{m} \qquad (3.1)$$

here m is the total number of available processors in system to schedule the tasks. So, the load balance is calculated as :

$$load\_balance = \frac{makespan}{Avg} \qquad (3.2)$$

For example consider a DAG given in Fig 3.1 consisting of nine tasks to be scheduled on a homogeneous multiprocessor system with three processors. Suppose that two task scheduling solutions as given in Fig.3.2 are obtained using some algorithm. The schedule length of both solutions is equal to 23.

$$Avg = (12 + 17 + 23)/3 \approx 17.33$$

$Load\_balance = 23/17.33 \approx 1.326$

$$Avg = (9 + 11 + 23)/3 \approx 14.33$$

$Load\_Balance = 23/14.33 \approx 1.604$

So, according to the balance fitness function, solution in Fig 3.2(a) is better than solution in Fig 3.2(b). Hence using this function we will obtain not only the minimum makespan but also greater load balance. Therefore we now propose to use two fitness functions given in (3.3) and (3.4) one after other, where f^k is the makespan of the k^{th} chromosome and k_c is the total number of chromosomes and whereas (3.3) has to be maximized and (3.4) has to be minimized.

$$evaluate(V_k) = 1/f^k, \quad k = 1,2,3,\dots,k_c \tag{3.3}$$

$$evaluate(V_k) = f^k/Avg, \quad k = 1,2,3,\dots,k_c \tag{3.4}$$

3.3 IMPLEMENTATION OF THE PROPOSED ALGORITHM

To evaluate the performance of our proposed algorithms, we have applied it to twelve task graph examples given in appendix A and three benchmark application programs which are taken from a Standard Task Graph (STG) archive [36] (and given in table 3.1). The first program of this STG set consists of a task graph of robot graph program with ninety tasks, the other two are random task graphs with fifty and hundred nodes respectively. The algorithm is implemented on an Intel processor (2.6 GHz) using C language. The results obtained are compared in fig 3.3 to 3.5.The population size is considered to be 20, and the maximum number of generations used is 500. The results obtained by proposed algorithm are tabulated in table 3.2 and 3.3. For comparison the corresponding results whereas available are also listed in these tables.

3.4 ANALYSIS OF THE RESULTS

From the results presented in tables 3.2 and 3.3 we observe that in the case of homogeneous multiprocessor problems without communication cost there is very

minor and practically no effect of load balancing on the makespan (examples 3 and 7 and benchmark problems). In the case of homogeneous examples with communication cost there is improvement in load balancing in almost each case. In the case of heterogeneous multiprocessor examples there is improvement in the load balance but it is at the cost of makespan. However in case of example 3 of heterogeneous type, which is an assignment eleven tasks on four processors. There is considerable improvement in load balance value (from 1.82 to 1.41) without any increase in the makespan. This is primarily because without load balancing tasks are assigned only to three processors. In general it has been noticed that load balancing is not needed in the case of problems without communication costs but is needed in case of heterogeneous multiprocessor systems with communication cost.

3.5 CONCLUSIONS

In this chapter we have proposed an algorithm for finding optimal schedule for multiprocessor task scheduling problems which tries to balance the load on different processors by using load balance function. This algorithm is able to find a optimal solution with minimum makespan and maximum available load on spread each processor. Our proposed algorithm has been applied to a set of total fifteen problems. It is noticed that although load balancing can be tried on both homogeneous and heterogenous multiprocessor systems it is more effective in the case of homogeneous systems as compared to heterogeneous systems.

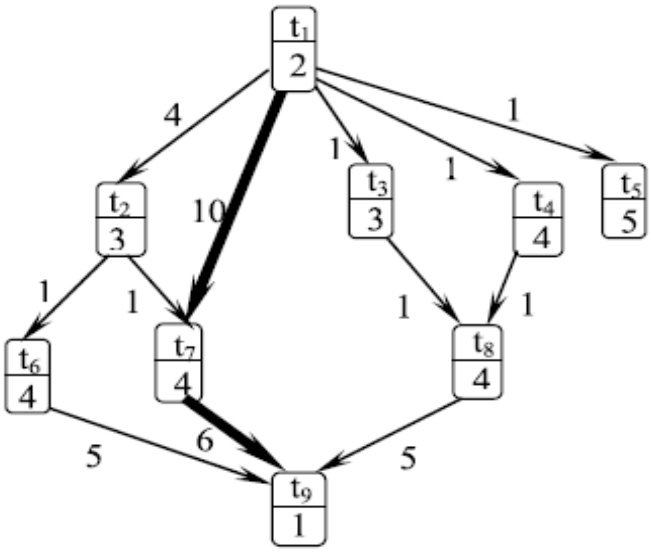

Fig 3.1: An Example DAG with nine Tasks

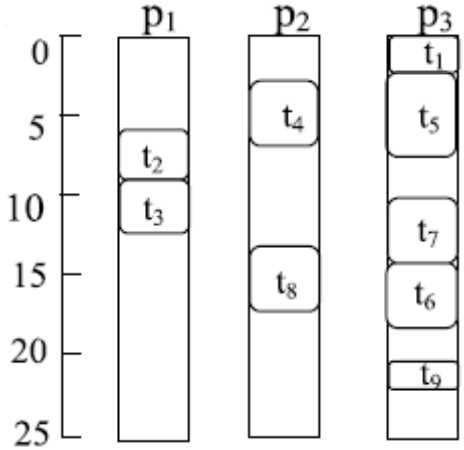

Fig.3.2(a): Schedule for the DAG given in Fig 3.1 (makespan 23 and load balance 1.326.

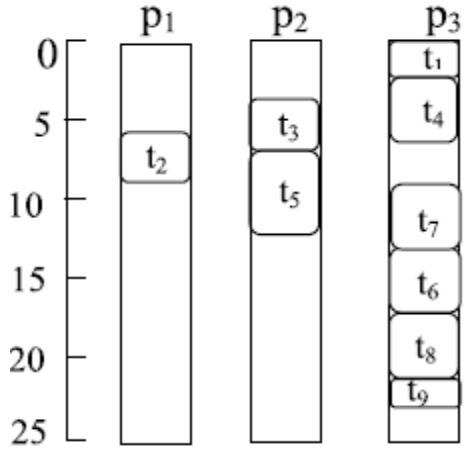

Fig.3.2(b): Schedule for the DAG given in Fig 3.1 (makespan 23 and load balance 1.604

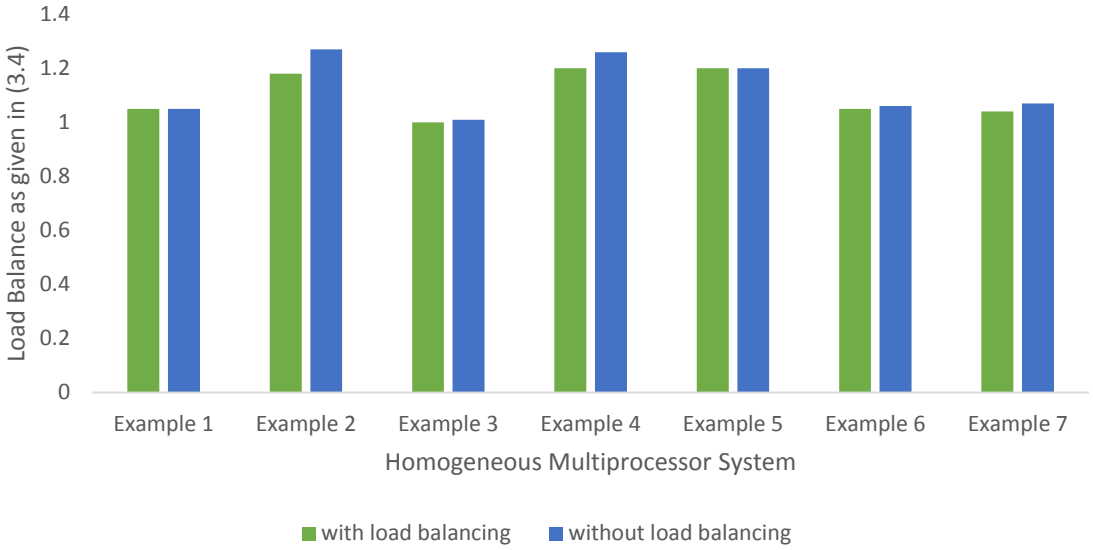

Fig 3.3: A comparison of load balance for homogeneous multiprocessor problems

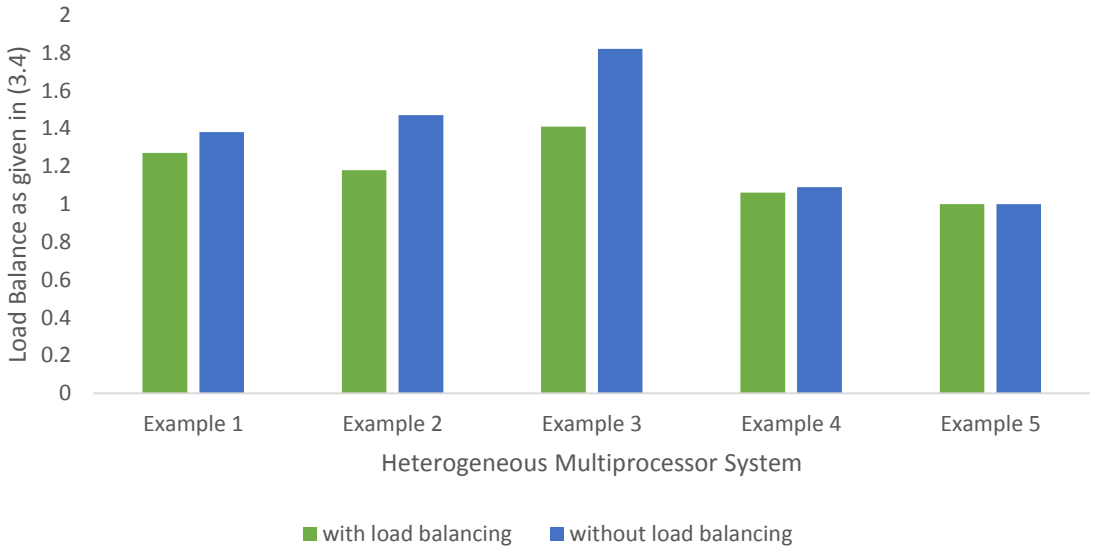

Fig 3.4: A comparison of load balance for heterogeneous multiprocessor problems

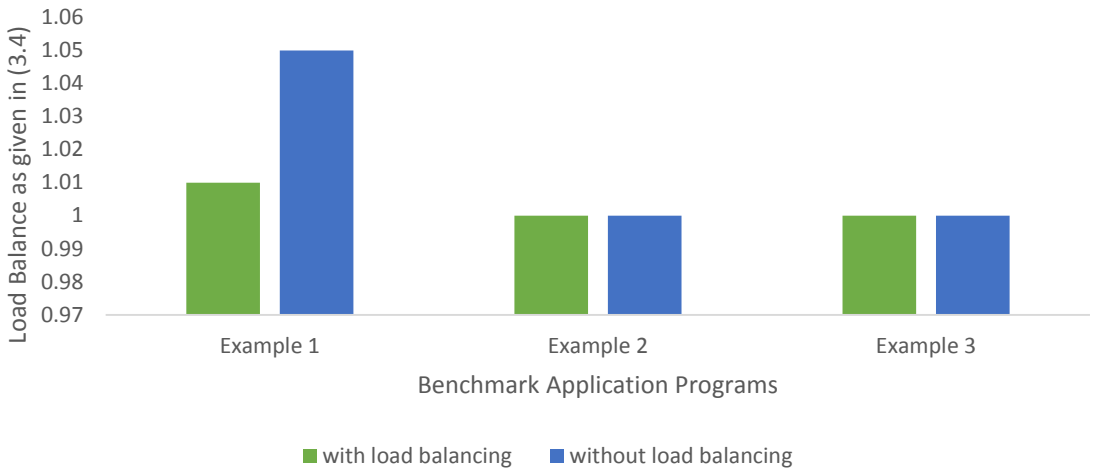

Fig 3.5: A comparison of load balance for benchmark problems

Table 3.1: Selected benchmark programs

Benchmark Programs	No. of Tasks	No. of Processors	Note
robot.stg	90	2	Robot control program
rand0000	50	2	Random Graph
rand0001	100	2	Sparse matrix solver

Table 3.2: Results obtained before applying load balance fitness function

Example No.	No. of Tasks	No. of Processors	Commun-ication Cost	Makespan	Load Distribution	Avg Load	Load Balance
Homogeneous Multiprocessor System							
1	18	2	Yes	440	440, 400	420	1.05
2	9	2	Yes	23	13, 23	18	1.27
3	12	2	No	60	60, 58	59	1.01
4	9	3	Yes	16	10, 12, 16	12.66	1.26
5	9	3	Yes	160	100, 160, 140	133.30	1.20
6	18	3	Yes	390	390, 360, 350	366.7	1.06
7	40	4	No	208	208, 192, 206,166	193	1.07
Heterogeneous Multiprocessor System							
1	10	3	Yes	23	15, 23, 12	16.66	1.38
2	10	3	Yes	73	39, 73, 36	49.33	1.47
3	11	4	Yes	26	26, 0, 14, 17	14.25	1.82
4	10	2	Yes	24	24, 20	22.00	1.09
5	11	2	Yes	56	56, 56	56	1.00
Benchmark Examples (Homogeneous Multiprocessor System)							
1	90	2	No	1313	1313, 1170	1241.5	1.05
2	50	2	No	131	131, 131	131	1.00
3	100	2	No	291	291, 291	291	1.00

Table 3.3: Results obtained after applying load balance fitness function

Example No.	No. of Tasks	No. of Processors	Commun-ication Cost	Makespan	Load Distribution	Avg Load	Load Balance
Homogeneous Multiprocessor System							
1	18	2	Yes	440	440, 400	420	1.05
2	9	2	Yes	23	16, 23	19.50	1.18
3	12	2	No	62	62, 62	62	1.00
4	9	3	Yes	16	16, 14, 10	13.33	1.20
5	9	3	Yes	160	140, 160, 100	133.33	1.20
6	18	3	Yes	390	390, 360, 360	370.00	1.05
7	40	4	No	217	216, 202, 217, 201	209	1.04
Heterogeneous Multiprocessor System							
1	10	3	Yes	28	19, 28, 19	22	1.27
2	10	3	Yes	90	72, 90, 66	76	1.18
3	11	4	Yes	26	26, 13, 16, 19	28.50	1.41
4	10	2	Yes	25	25, 22	23.50	1.06
5	11	2	Yes	56	56, 56	56	1.00
Benchmark Examples (Homogeneous Multiprocessor System)							
1	90	2	No	1313	1313, 1289	1301	1.01
2	50	2	No	131	131, 131	131	1.00
3	100	2	No	291	291, 291	291	1.00

CHAPTER 4

CHAPTER 4

OPTIMAL NUMBER OF PROCESSORS IN A MULTIPROCESSOR SYSTEM

In this chapter we have focused on determining the optimal number of processors that need to be used in a multiprocessor system with a view to minimize the system cost which is incurred by adding new processors to the system (By optimal number of processors we mean the number of processors by using more than which there is no significant improvement in the makespan).

4.1 MOTIVATION

It is possible to construct parallel computers employing a large number of processors. The availability of such systems has fueled interest in investigating their performance for various kinds of problems. When a problem is solved in parallel, it is reasonable to expect a reduction in the execution time. However it is commensurate with the additional cost incurred and amount of processing resources employed to solve the problem. The scalability of a parallel algorithm on parallel architecture is a measure of its capacity to effectively utilize increasing number of processors. A scalable parallel algorithm may be used to determine the optimal number of processors to be used and the maximum speed up that can be obtained. Kumar and Gupta [66] have shown that the scalability can also predict the impact of changing hardware technology on the performance and thus help design better parallel architectures for solving various problems at a low cost.

In the case of multiprocessor systems, it is found that if we increase the number of processors then that does not automatically decrease the execution time of a program. However installing a new processor always increases the hardware cost of the system. The availability of the efficient scheduling algorithms does not guarantee maximum utilization of processors. So, there is a need to find the optimal number of processors that will increase the throughput without adding unnecessarily to the cost of the overall system.

In chapter 2 we proposed a technique based on genetic algorithm approach for calculating the minimum execution time on a fixed number of processors. Its use for deciding the optimal number of processors is presented in the present chapter. The proposed algorithm in this chapter is used to determine optimal schedules using different number of processors for a set of 7 homogeneous multiprocessor systems and 5 heterogeneous multiprocessor scheduling problems taken from literature and given in appendix A.

The proposed method to decide optimal number of processors is given in section 4.2. Its implementation is done in section 4.3 on same set of problems listed in appendix A. The obtained results are next analyzed in section 4.4. Conclusions based on the present study are drawn in section 4.5.

4.2 DECIDING OPTIMAL NUMBER OF PROCESSORS IN A MULTIPROCESSOR SYSTEM

There are various scalability measures available in the literature, but no single measure exists that is uniformly better than others. Different measures are suitable for different situations. Keeping this in view we have developed a technique based on Genetic Algorithms (GAs) approach that can be used to determine optimal number of processors and maximum speedup possible for problems of multiprocessor scheduling. For this multiprocessor scheduling problem is set in the form of a Directed Acyclic Graph (DAG). The algorithm for fixed number of processors as given in chapter 2 can be applied to calculate minimum execution time of a problem on a specified set of processors.

The proposed algorithm for determining the optimal number of processors is summarized below.

Step 1. Input: Number of tasks "n", Number of processors "m", computation cost matrix "$cm[n][m]$", communication cost matrix "$cd[n][n]$".

Step 2. Decide priorities of tasks.

Step 3. Encode the chromosome string, size of initial population equal to 10.

Step 4. Calculate the fitness of chromosome string using evaluation function.

Step 5. Using tournament selection selects the fittest chromosomes.

Step 6. Crossover: the good chromosomes are copied unaltered in the next population.

Step 7. Mutation: the swap mutation operator is used that guarantee generation of legal offspring's.

Step 8. Repeat the steps 1 to 7 until optimal solution is achieved.

Step 9. Repeat the steps 1 to 8 for different choice of number of processors until Makespan stabilized or start increasing.

Step 10. Output: Makespan (Minimum execution Time with full schedule with processors allocated to all tasks) and optimal number of processors required.

4.3 IMPLEMENTATION OF PROPOSED METHOD

The proposed algorithm has been coded in C language and implemented on a laptop having Intel(R) Core(TM) 2 Duo 1.67 GHz. Following parameters have been used in our study to decide the optimal number of processors required:

1. Initial Population size: popSize=4 for problems of 9 to 12 tasks and 10 for problems 18 and 40 tasks.
2. Maximum number of iterations: maxGen=200 for problems of 9 to 10 tasks, 400 for problem of 11 to12 tasks, 800 for problem of 18 to 20 tasks and 1000 for problem of 40 tasks.
3. Crossover probability=0.8
4. Mutation probability=0.6
5. Terminating condition: 50 generations with same fitness for problems of size 9-18 tasks, 120 generations with same fitness for problem of 40 tasks.
6. Total number of trials= 10 for each problem.

The crossover and mutation probabilities are kept quite high to generate a new set chromosomes in each generation. To analyze the problem of deciding optimal number of processors we have conducted study on a set of 12 experimental

problems taken from literature. In this study we calculate the total minimum execution time of each DAG using different number of processors to find the optimal number of processors. We kept on increasing number of processors till Makespan value stabilized or started increasing. The optimal number of processors is the number of processors by using more than which there is no significant improvement in Makespan value. The number of processors varies from 1 to 8 depending on the data available in literature.

For illustration purpose we present in detail its implementation on a heterogeneous computing system of "$n = 10$" executable tasks $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}\}$ and consisting of a set of "$m = 3$" processors $P = \{P_1, P_2, P_3\}$ connected by an arbitrary network [72]. The DAG of problem example is given in fig 4.1 and its computation cost matrix is given in table 4.1 respectively. The proposed algorithm when simulated calculates the minimum execution time on specified set of processors which are then used to decide the optimal number of processors required. After calculating the execution time of all processors it suggests the best solution available.

The detailed result depicting the minimum execution time and optimal number of processors is presented in fig 4.9. This problem has been earlier solved using different algorithms for fixed number of processors. The optimal result found till now was 28 units of time when executed on three processors. But the proposed algorithm has given the schedule length equal to 23 units of time when executed on 3 processors namely (P_1, P_2, and P_3). After this when we increased the number of processors, we found no reduction in schedule length.

However in a heterogeneous system processors have different capabilities. In the case of heterogeneous multiprocessor systems whenever we had to add a new processor the communication cost of one of existing processors was duplicated on this processor. In other words if we have a set of 3 processors (P_1, P_2 and P_3) and we want to add an additional processor P_4 then communication cost of P_1 is duplicated on P_4 in one case, then of P_2 and P_3. The new sets of four processors created in this way are (P_1, P_2, P_3, and P_1), (P_1, P_2, P_3 and P_2) and (P_1, P_2, P_3 and P_3). The results

obtained for problems given in appendix A are depicted in fig 4.2 to fig 4.13 respectively.

4.4 ANALYSIS OF RESULTS AND CONCLUSIONS

Results obtained in the case of homogeneous multiprocessor systems show that if we have a fixed size problem and increase the number of processors then the execution time of the problems does not necessarily go on decreasing. In case of a homogeneous multiprocessor system in fact the speedup becomes zero if we apply a large number of processors to small size problems. For example in problem 2 of homogeneous system we noticed that the speedup actually peaked at 3 and when we increase the number of processors from 3 to 4, it does not decrease and instead starts increasing mainly because of transfer time. Moreover the overheads grow faster with introduction of the additional processors. On the basis of this study we notice that 2 to 3 processors are in general sufficient for small size scheduling problems of upto 9 tasks on a homogeneous multiprocessor system and 4 processors will be sufficient for medium size problems of up to 40 tasks. Infat not more than 8 processors are necessary for problems of size up to 120.

In the case of heterogeneous multiprocessor scheduling problems the processors present in the system have different processing capabilities. So, by using the proposed algorithm one not just only gets the optimum number of processors but is also able to select the best combination of processors. If we keep the number of processors same but change the processors with those having different capabilities then there is a change in execution time of task graph as shown to fog 4.10 to 4.14. For example in problem 4 of heterogeneous system we observed that if we keep the number of processors same but vary the combination of processors from P_1, P_1, P_2 to P_1, P_2, P_2 then there is a considerable decrease in the execution time. So the optimal number of processors in this case is 3 with choice (P_1, P_2, P_2). In this case also our analysis has shown that in general one hardly needs more than three processors for executing medium size problems of tasks up to 18, not more than 4 processors for executing problems of size up to 40 tasks and not more than 6 processors for 120 task problems.

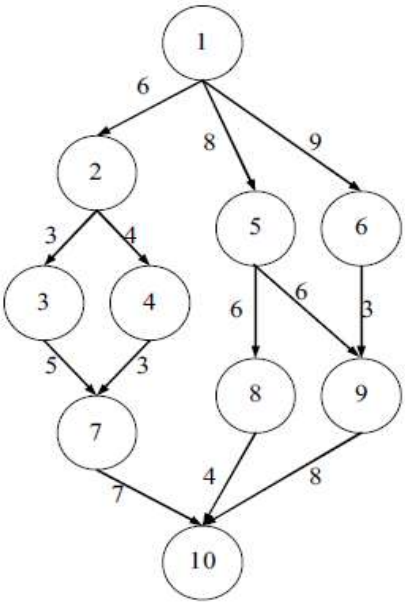

Fig 4.1: DAG of Experimental Problem

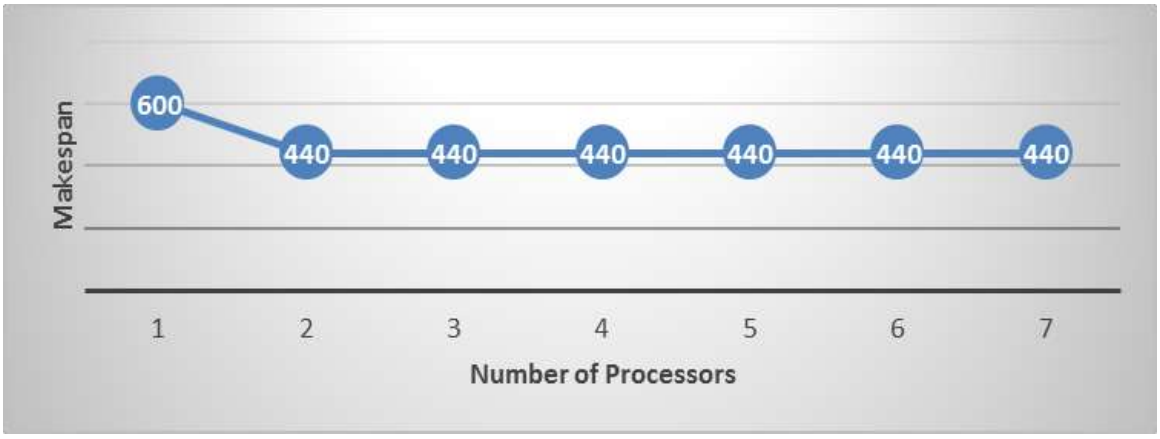

Fig 4.2: Makespan obtained on different number of processors for homogeneous scheduling example 1.

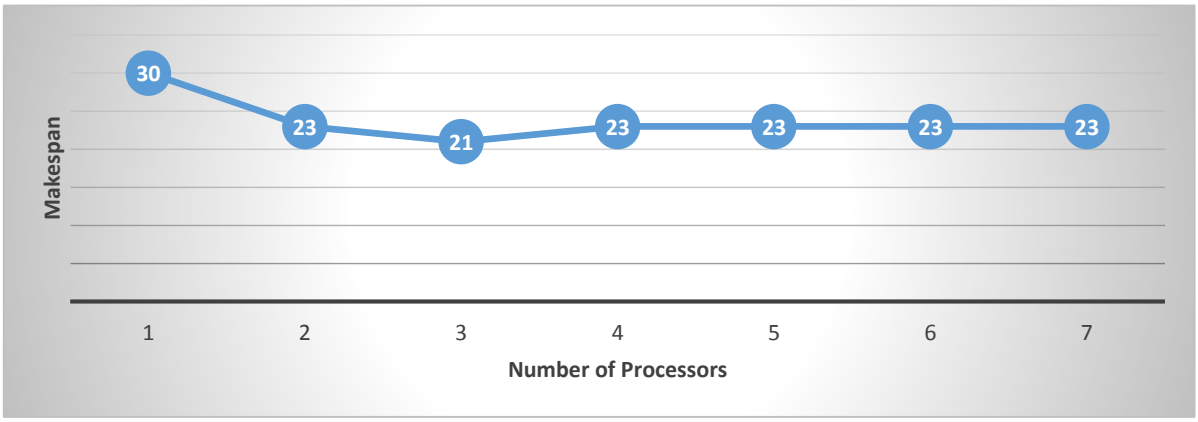

Fig 4.3: Makespan obtained on different number of processors for homogeneous scheduling example 2.

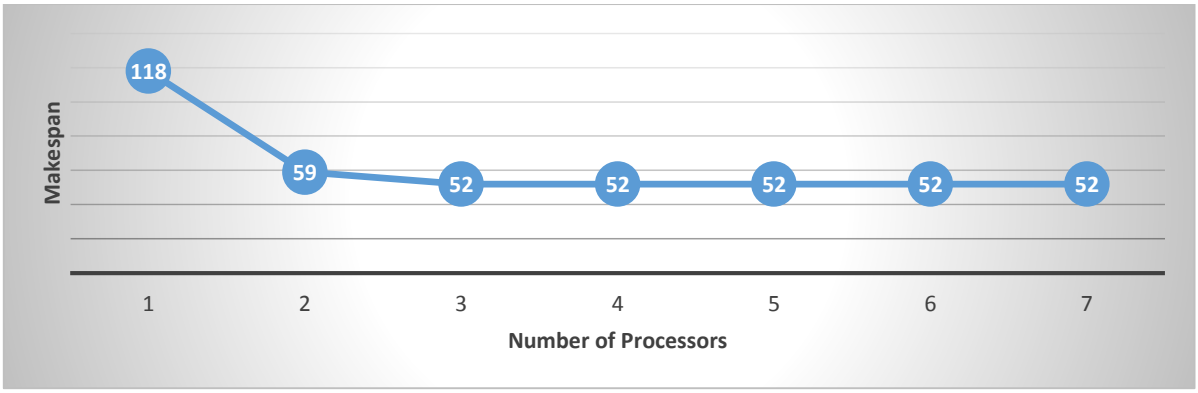

Fig 4.4. Makespan obtained on different number of processors for homogeneous scheduling example 3.

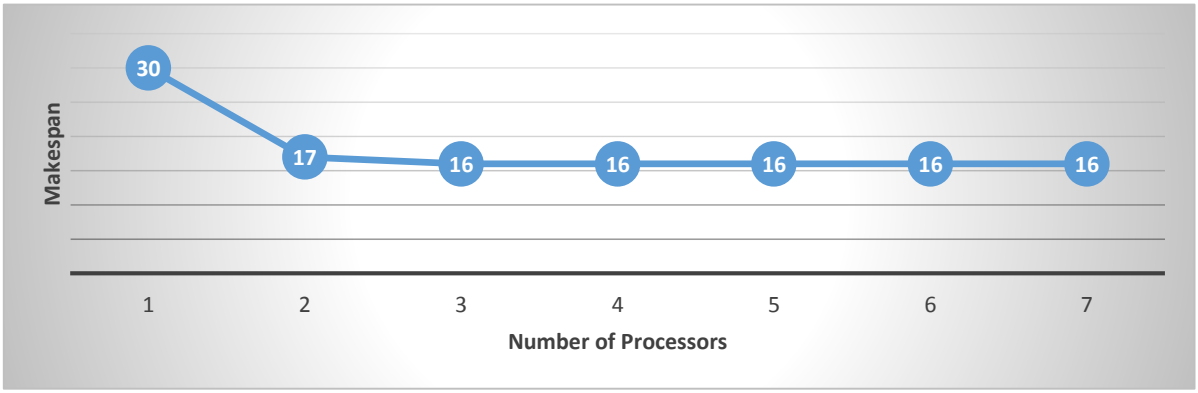

Fig 4.5: Makespan obtained on different number of processors for homogeneous scheduling example 4.

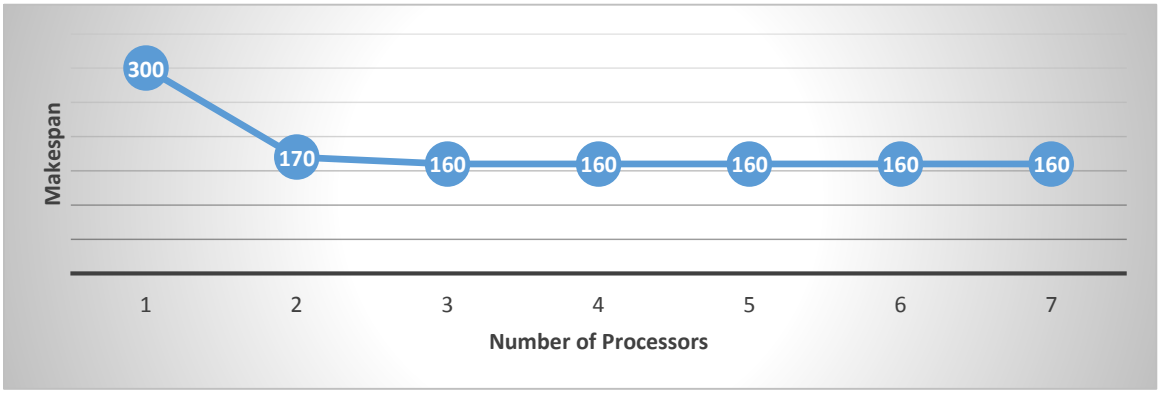

Fig 4.6: Makespan obtained on different number of processors for homogeneous scheduling example 5.

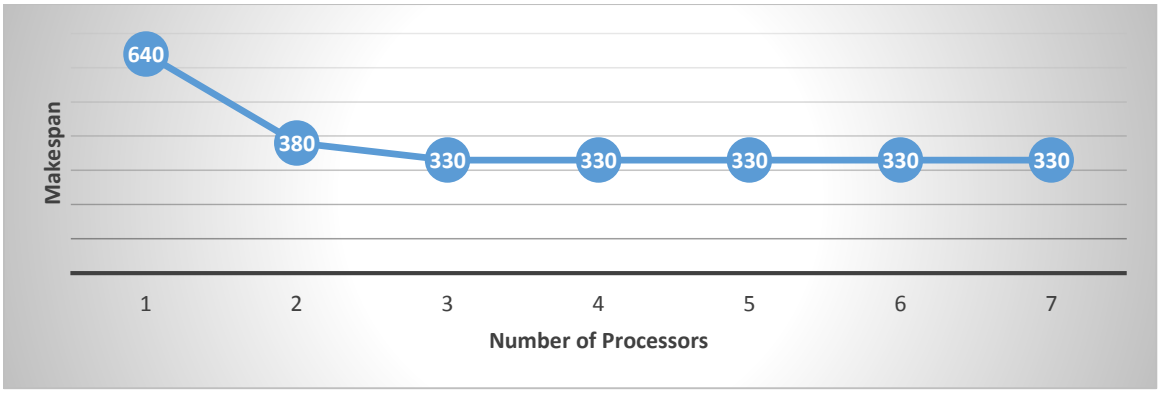

Fig 4.7: Makespan obtained on different number of processors for homogeneous scheduling example 6.

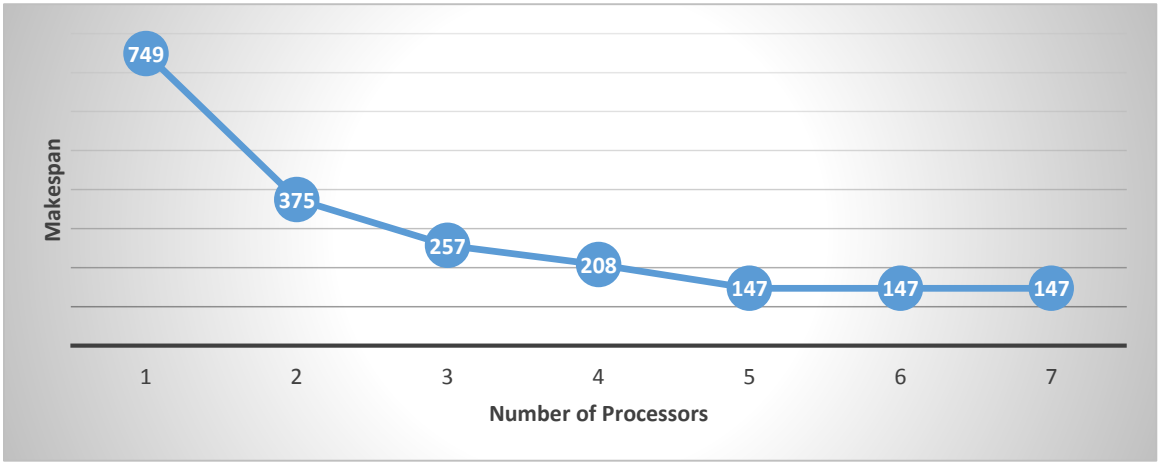

Fig 4.8: Makespan obtained on different number of processors for homogeneous scheduling example 7.

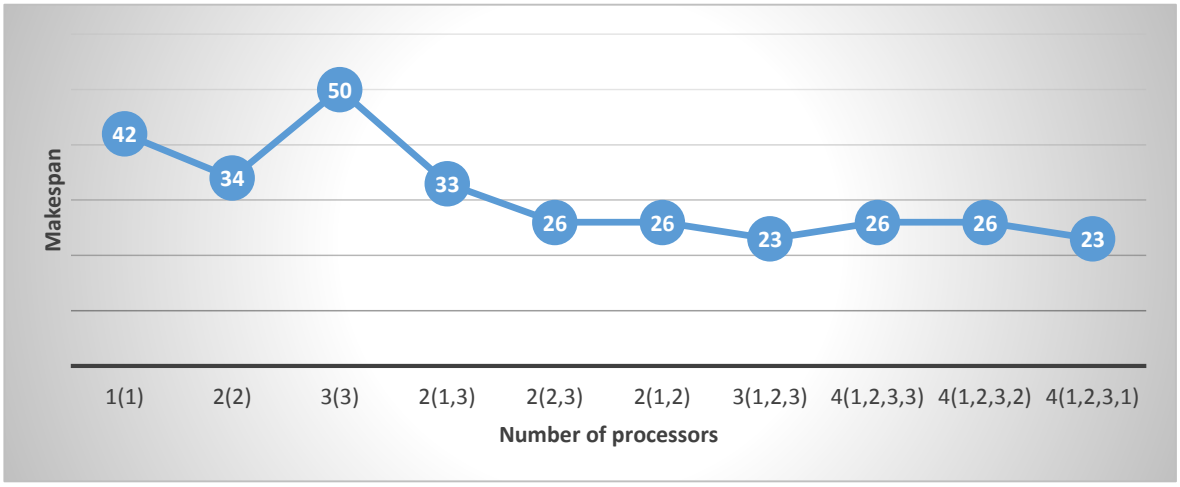

Fig 4.9: Makespan obtained on different number of processors for heterogeneous scheduling example 1, number in small parenthesis denote the name of processor i.e. 1 is used for P_1, 2 for P_2, 3 for P_3 and 4 for P_4.

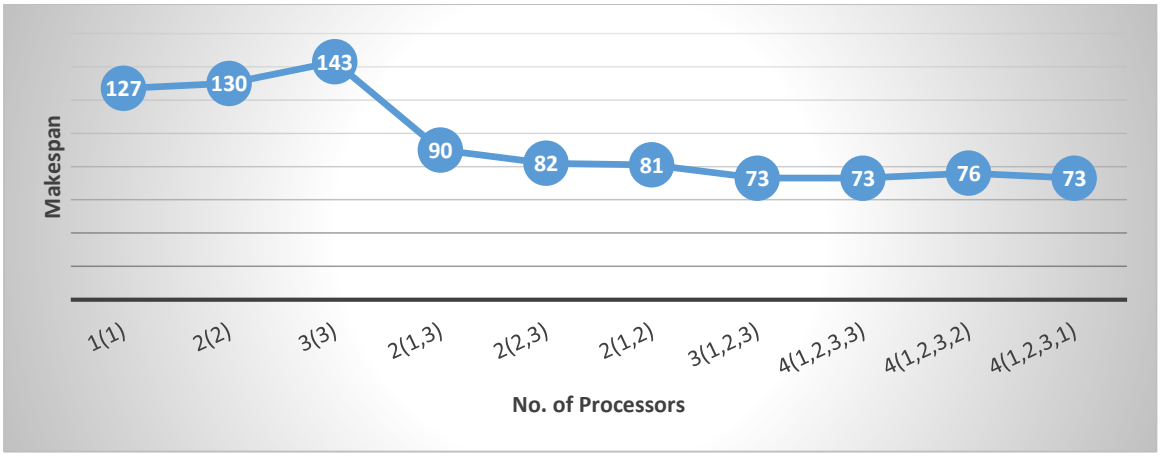

Fig 4.10: Makespan obtained on different number of processors for heterogeneous scheduling example 2, number in small parenthesis denote the name of processor i.e. 1 is used for P_1, 2 for P_2, 3 for P_3 and 4 for P_4

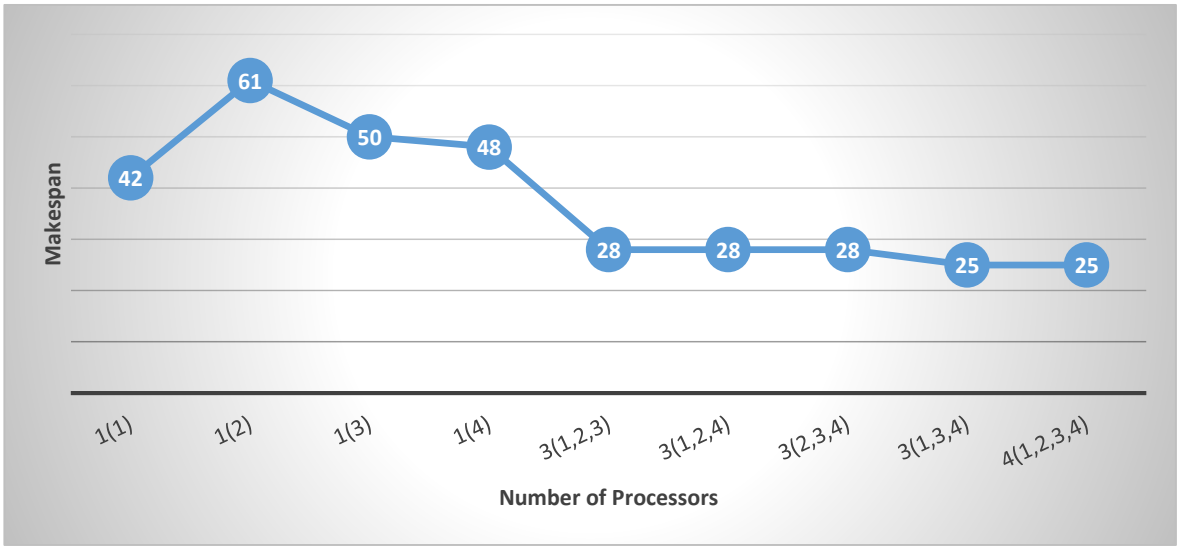

Fig 4.11: Makespan obtained on different number of processors for heterogeneous scheduling example 3, number in small parenthesis denote the name of processor i.e. 1 is used for P_1, 2 for P_2, 3 for P_3 and 4 for P_4

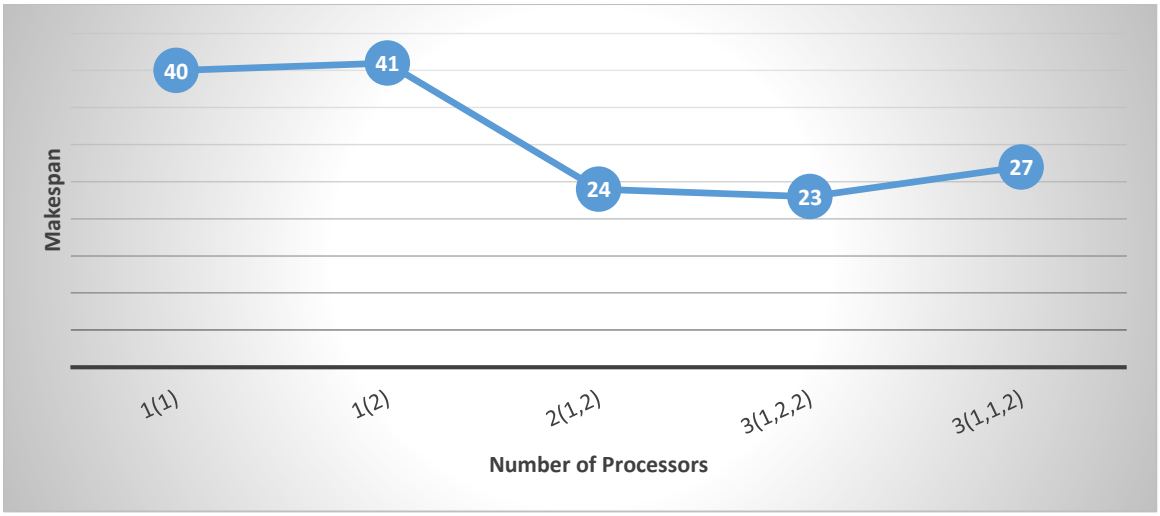

Fig 4.12: Makespan obtained on different number of processors for heterogeneous scheduling example 4, number in small parenthesis denote the name of processor i.e. 1 is used for P_1, 2 for P_2 and 3 for P_3.

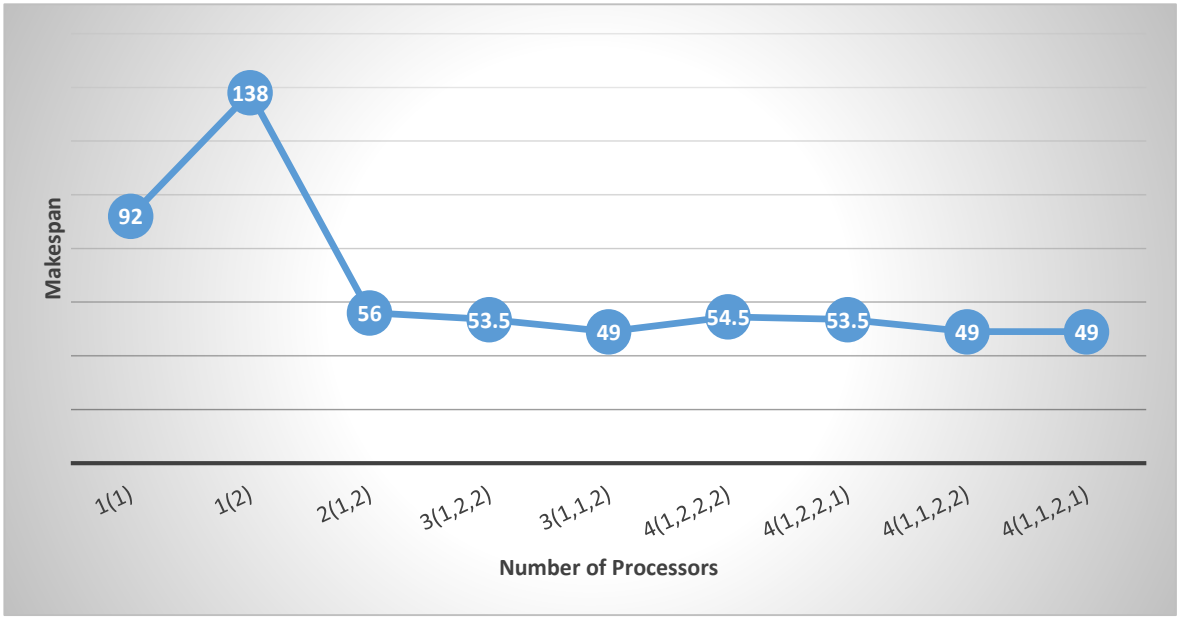

Fig 4.13: Makespan obtained on different number of processors for heterogeneous scheduling example 5, number in small parenthesis denote the name of processor i.e. 1 is used for P_1, 2 for P_2, 3 for P_3 and 4 for P_4.

Table 4.1: Communication Cost matrix of experimental Problem given in Fig 4.1

PEs / Task n_i	PE1	PE2	PE3
1	5	3	4
2	3	2	7
3	3	4	8
4	7	5	3
5	5	2	5
6	3	4	8
7	4	2	3
8	2	3	4
9	4	5	3
10	6	4	5

CHAPTER 5

CHAPTER 5

CHOICE OF AN APPROPRIATE SOFTWARE RELIABILITY GROWTH MODEL FOR OBSERVED FAILURE DATA

In this chapter a method is proposed for selecting an appropriate reliability growth model that best fits the available detected fault data midway during its testing stage. The proposed method is relatively simple and easy to implement. Its performance has been tested on ten real datasets taken from literature. The performance of the method has been also compared with some of the earlier methods proposed in literature for this purpose.

5.1 MOTIVATION

Computer software is now being used in practically every field of human activity. For this appropriate softwares have to be designed, developed and tested before release. Software development is a time consuming process involving extensive costs. When software has been developed it is subjected to testing before release to ensure that as far as possible it is bug free and reliable. Software reliability is in fact one of the most important features of a developed software system. ANSI defines [75], software reliability as the probability of failure-free operation of a software for a specified period of time under specified environment. In practice, it is very difficult for the project managers to measure the software reliability. Since early 1970s, a number of Software Reliability Growth Models (SRGMs) have been proposed for the estimation of reliability growth of products during software development processes [51], [79-80] particularly when it has been developed and is in the testing stage. A commonly accepted metric for measuring a product's reliability is the number of failures one expects to find within a certain time. Failures are the results of some faults in the software code and several failures can result from even a single fault [108].

Objective in this study is to assess the effectiveness of SRGMs during the testing stage of the software. In this chapter we propose a method to select an

appropriate SRGM from amongst the commonly used ones to obtain reliability estimations during product testing stage and use it to estimate the likely release date of the product. The method essentially uses the available test failure data to decide appropriate SRGM model that best fits available fault data.

The chapter is organized as follows. A brief introduction to SRGM's is given in section 5.2. The proposed method is described in section 5.3. The proposed method is next in section 5.4 applied on ten real datasets given in Appendix B. Conclusions based on the present study are finally drawn in section 5.5.

5.2 NHPP SRGMs

The non-homogeneous Poisson Process (NHPP) class of SRGMs is fault counting models that have been widely used in literature for estimating the future behavior of software [27], [54], [56], [107], [113]. These models can forecast the software's expected number of failures and future reliability using test data from failure history data. These models have also been used to estimate the expected residual number of failures in the software and the test time required to detect them. These models have different mean value functions based on different assumptions of the error content function, and the defect detection rate. According to the patterns of their mean value functions the NHPP SRGMs models are usually classified as (i) concave models and (ii) S Shaped models. The concave models depict the defect debugging process naturally (i.e. the defects found cumulate as testing process continues and the cumulative number of defects gradually grows at a slower rate and then eventually approaches asymptotic behavior as the software behavior stabilizes). The S-shaped curve type models on the other hand, indicate a slower defect detection rate at the beginning of the debugging process. Then the detection of defects picks up at a faster rate as testing continues and eventually the cumulative defects curve approaches asymptotic behavior.

Furthermore, finite and infinite failure models can be distinguished in this class. Finite failure models assume the possibility of finally developing a fault-free product (an asymptotic approach to a finite value), whereas the infinite failure

models assume that the number of failures observed is infinite, which means that the mean value function is unbounded. Some of the models also assume that while fixing detected bugs some new bugs can also get inadvertently introduced. These are referred to as imperfect debugging models.

A large number of SRGMs have been proposed during the past 35 years to estimate various software reliability measures such as the number of remaining failures, software failure rate, and achieved software reliability. Some of the well-known software reliability growth models that have been used in our proposed study are Goel Okumoto [29], Generalized Goel [29], Modified Duane [53] Musa Okumoto [29], Yamada Exponential [126], Gompertz [53], Inflection S Shaped [39], Logistic Growth [53], Delayed S Shaped [14], Yamada Imperfect Debugging Model I [70], Yamada Imperfect Debugging Model II [87], Yamada Rayleigh [90], Pham Zhang IFD [90], Zhang-Teng-Pham model (ZT Pham) [134], Pham Nordman Zhang (PNZ Model) [87] and Pham Zhang model (PZ Model) [88]. Their silent features are listed in table 5.1, of these models 1 to 5 are concave, 6 to 14 are S shaped, Models 15 and 16 behave as concave or S shaped depending upon value of fault introduction rate parameter. Moreover models 12 to 16 are imperfect debugging models.

Software reliability growth models can capture the quantitative aspects of the software testing process, and thus can be used to provide a reasonable estimate for future. During the software testing phase, the developers can use the SRGMs to determine the most appropriate SRGM which best fits the available fault data thus far and use it to estimate when to stop testing and release the software. A number of tools and techniques for software reliability model selection have been proposed in literature [3], [51], [75], [79], [80-81], [105], [120]. However most of these are situation specific and cannot be used in general with high confidence.

5.3 PROPOSED METHOD

In the present study we propose a method which appears relatively simple to select an appropriate SRGM for available failure data. The method is not situation specific, reasonably fits and can be used in general. The method consists in first estimating

the parameters of appropriate models and then ranking them based on their performance on presently available data during testing stage. The proposed method works as under.

5.3.1 Estimation of Model Parameters

Parameter estimation is generally achieved by applying MLE (maximum likelihood estimation) or LSE (Least squares estimation) technique. The maximum likelihood technique estimates parameters by solving a set of simultaneous equations. The method of least squares minimizes the sum of squares of the deviations between actual errors and expected errors based on the selected theoretical model. For this an effective Matlab tool is available. Our personal experience has shown that compared to MLE, the LSE generally gives values for parameters for which the model fits actual data more appropriately. We therefore decided to use the LSE technique for parameters estimation using curve fitting tool of Matlab. After observing the available fault data using this tool we first fit the likely appropriate models to the experimental data and then use the difference between theoretically computed and observed errors to determine the value of Goodness of fit parameters RSq and RMSE for these selected models. For the fitted model along with the values of the unknown parameters of the model the values of RMSE and RSq are also provided by curve fitting tool of Matlab software. The working of tool is shown in fig 5.1 that how it fits the dataset and estimates the value of unknown parameters, RMSE and RSq.

5.3.2 Ranking of Models

In order to select an appropriate software reliability growth model, various comparison criteria have been proposed to in literature [3] [51], [75], [79], [80], [120] to compare models quantitatively. Most of these take into account more than ten parameters. Our experience has shown that using such a large set of parameters is not very necessary and in most of the cases even these do not ensure very reliable predictions. It was observed that the following relatively simple criteria can be used to rank competing software reliability models.

$$Rank\ Index_j = \frac{1}{2}\left[\frac{RSq_j}{max_j^n(RSq_j)} + \frac{min_j^n(RMSE_j)}{RMSE_j}\right] \quad (5.1)$$

where j denotes the index of the model. Also

$$Sq = 1 - \frac{SSR}{SST} \quad (5.2)$$

with

$$SSR = \sum_{i=1}^{n}(\widehat{m(t_i)} - \sum_{i=1}^{n} m(t_i)/n)^2 \quad (5.3)$$

$$SST = \sum_{i=1}^{n}(m(t_i) - \sum_{i=1}^{n} m(t_i)/n)^2 \quad (5.4)$$

In (5.3) and (5.4) i denotes the index of the observed failure and $m(t_i)$ the cumulative number of failures observed up to time t_i. Again $\widehat{m(t_i)}$ is the predicted value of cumulative failures at time t_i using the theoretical model under consideration.

RMSE in (5.1) is the fit standard error of the regression. It is an estimate of the standard deviation of the random component in the data and is defined as:

$$RMSE = \sqrt{MSE} \quad (5.5)$$

where $MSE = SSE/v$is the mean square error or (the residual mean square), SSE is the total deviation of the actual observed errors from the errors computed based on SRGM being used and $SSE = \sum_{i=1}^{n}(m(t_i) - \widehat{m(t_i)})^2$ and v being the degree of freedom. Degree of freedom is defined as the number of response values n minus the number of fitted coefficients m estimated from the response values.

Values of rank index in (5.1) are computed for competing models which are then ranked in ascending order of this value. Rank 1 being assigned to the model with highest rank index value. In case there is a tie (two models getting very close value of rank index) then both are considered to be of equal rank.

5.4 APPLICATION ON TEST DATA SETS

In this section we present our experience of applying the proposed method on ten datasets taken from literature [27], [80], [89], [107], [113], [134], [136]. These are listed in appendix B. In this study, we have considered consider 16 NHPP SRGMS given in table 5.1. The features of these SRGMs are also summarized in the same table.

The ranking of models is done at three stages of testing. First when 50% of test plan is completed, then when 75% of data is available and finally when the software testing is considered complete. The working of the proposed method is explained below in detail on dataset 1.

Dataset 1 [134] has 100 reported failures in 20 weeks. Accepting the actual data of testing for 20 weeks; the estimations of unknown parameter, RSq, and RMSE and then ranking of models is done at three stages 50% of testing data (i.e. data upto10 weeks), 75% (i.e. 15 weeks) of data and 100% of data (i.e. 20 weeks data).

The estimated values of the RSq, RMSE are obtained by using curve fitting toobox of Matlab. Next using (5.1) their rank index is calculated. Based on the value of obtained rank index the models are then ranked according to ascending value of rank index. The estimated value of RMSE, RSq, rank index and their ranks for dataset 1 using 1 to 10 weeks failure data are given in table 5.2 (a) to 5.2(c).

Table 5.2 shows that Gompertz is the rank one model for 50% of data considered, Logistic Growth for 75% of data and Zeng Teng Pham for 100 % of data. The results of other data sets are summarized in tables 5.3 to 5.11.

The approach is again applied to 75% of data (i.e. 1 to 15 weeks data from dataset 1) and using full 100% data (i.e. 1 to 20 weeks data). The results are summarized in tables 5.2(b) and 5.2(c) respectively. A comparison of results with results available in literature is given in fig 5.2 to 5.11 and table 5.12.

5.5 DISCUSSION ON RESULTS AND CONCLUSIONS

This chapter addresses the issue of optimal selection of software reliability model for a given dataset from a set of sixteen SRGMs. The comparison of results with those available in literature shows that the proposed approach is able to select a model that fits the present data closely. Therefore the selected model can be used for future predictions. The selected models estimates by our proposed method are more close to the actual number of failures found by that time in each of the case as shown in fig 5.2 to 5.11. There is only a 20% difference between the estimate and the actual number of failures for dataset 2, 3, 4 and 6 and in other case the estimates are very close. In fact model thus selected can be used for future predictions. The proposed approach was applied when 50% of test plan is completed, 75% of test plan is completed and on date of actual release. The predictions compared well with actual data. However when the approach was used on less than 50% of test data in all cases and observed that predictions in general were not reliable.

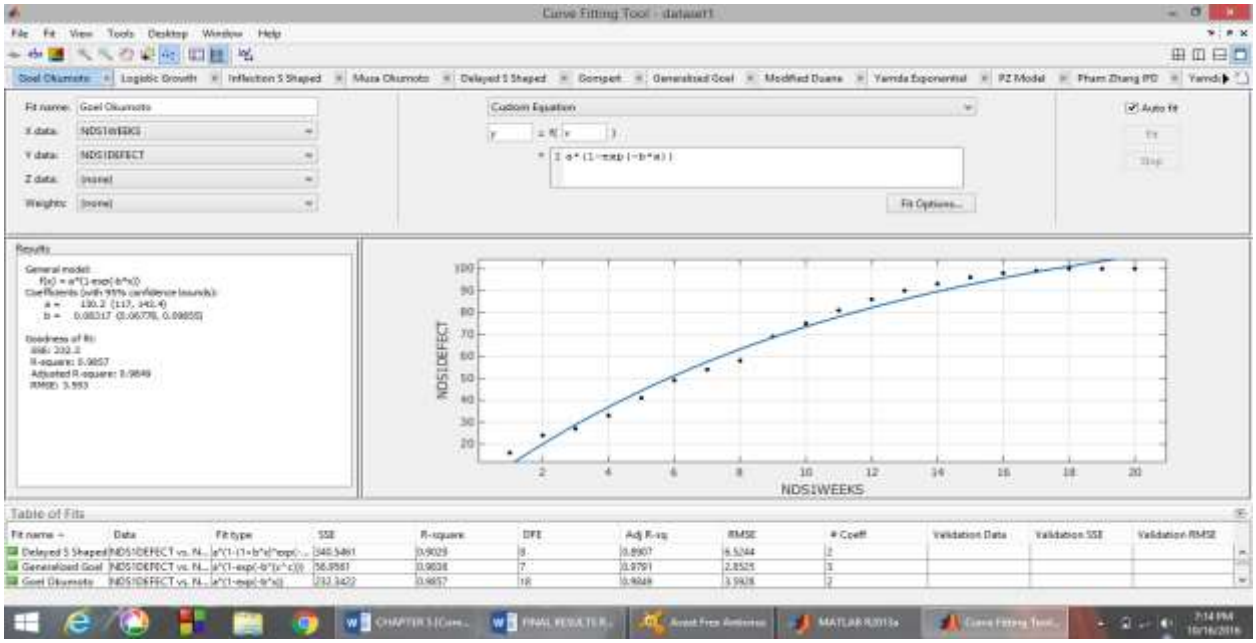

Fig 5.1. Snapshot of Matlab Curve fitting Tool used in Unknown Parameter Estimation

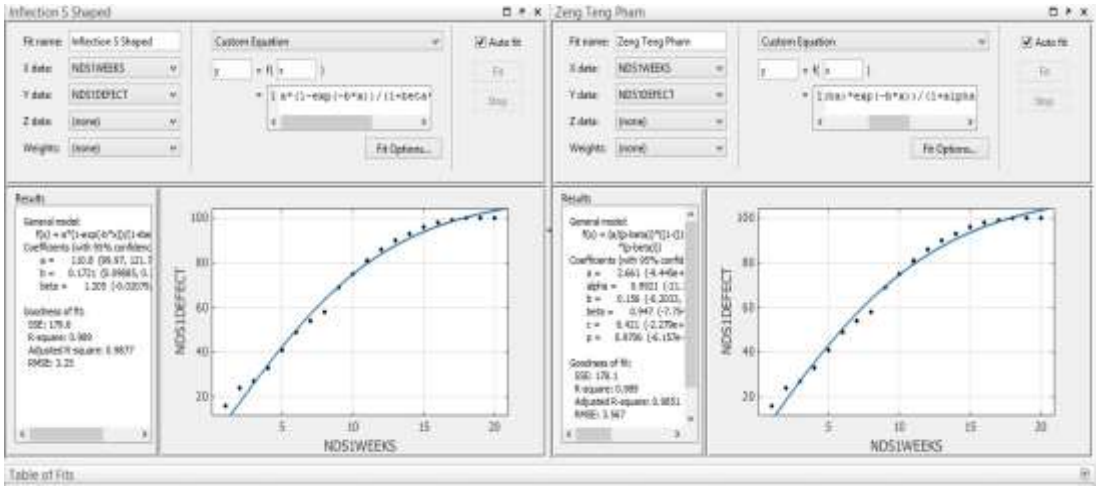

Fig 5.2: A comparison of model selected by our proposed method with the model suggested in literature for dataset 1

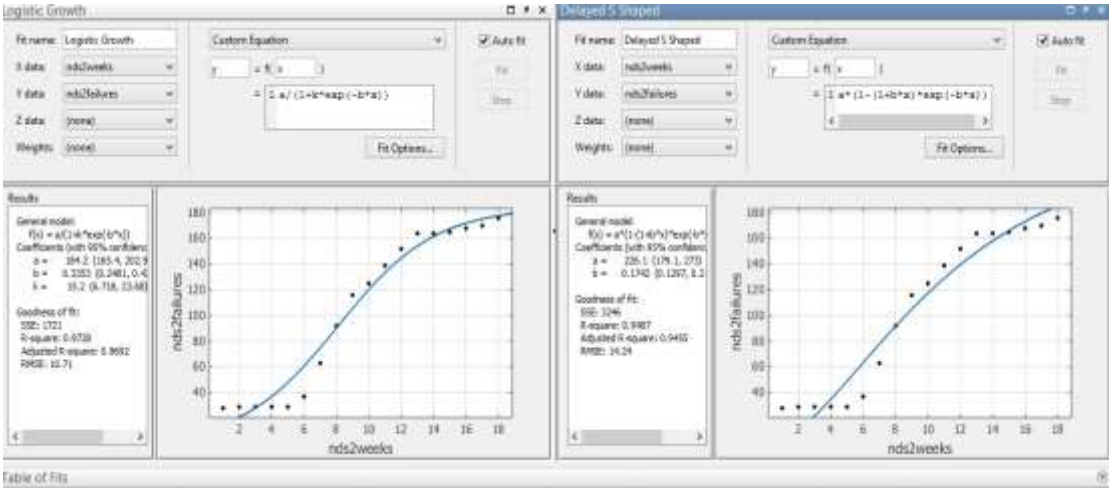

Fig 5.3: A comparison of model selected by our proposed method with the model suggested in literature for dataset 2

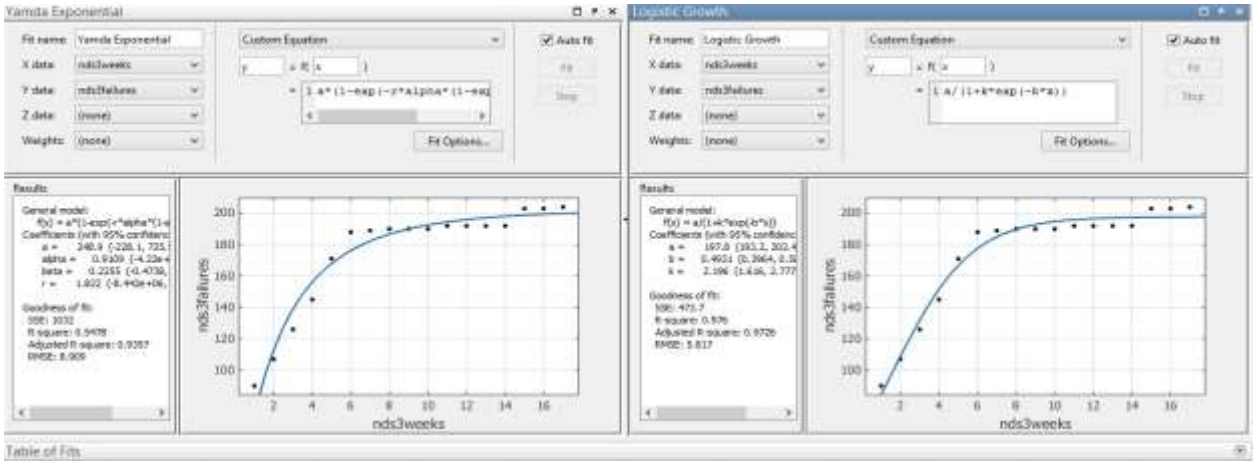

Fig 5.4: A comparison of model selected by our proposed method with the model suggested in literature for dataset 3

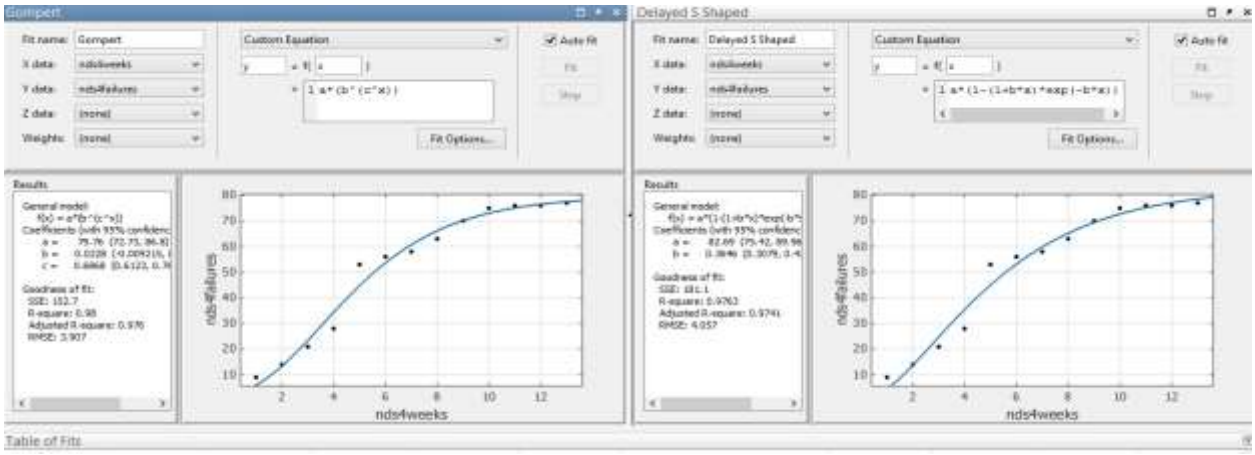

Fig 5.5: A comparison of model selected by our proposed method with the model suggested in literature for dataset 4

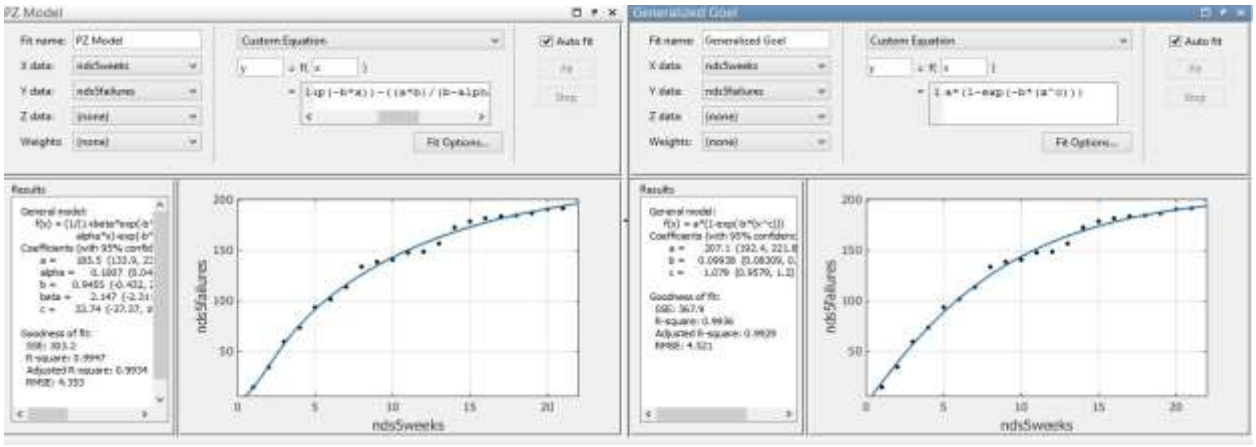

Fig 5.6: A comparison of model selected by our proposed method with the model suggested in literature for dataset 5

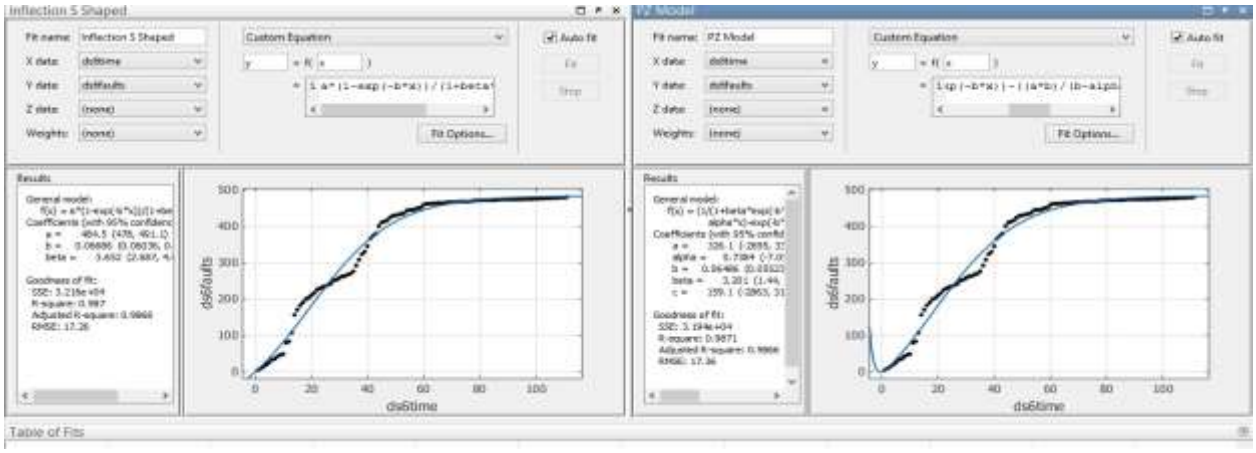

Fig 5.7: A comparison of model selected by our proposed method with the model suggested in literature for dataset 6

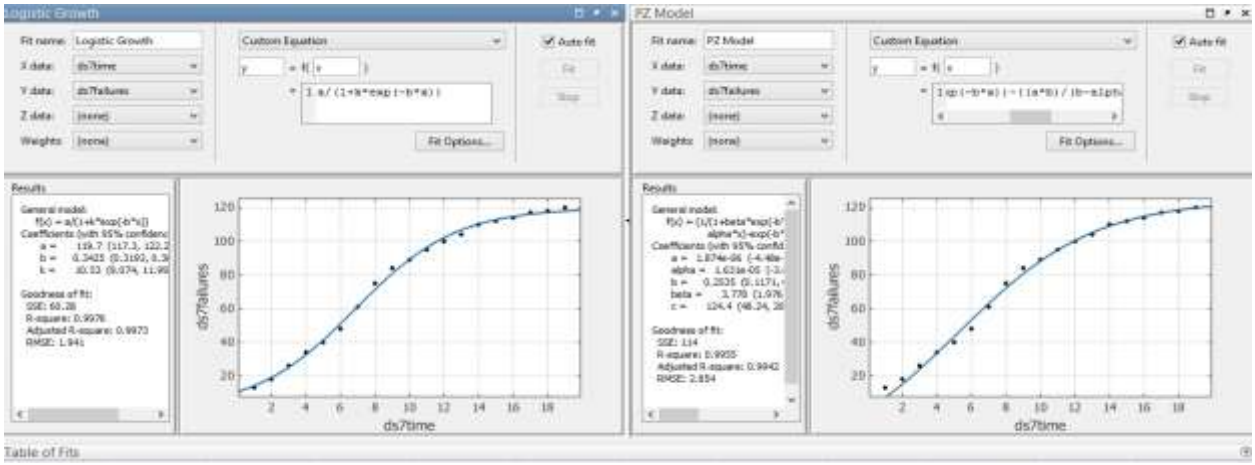

Fig 5.8: A comparison of model selected by our proposed method with the model suggested in literature for dataset 7

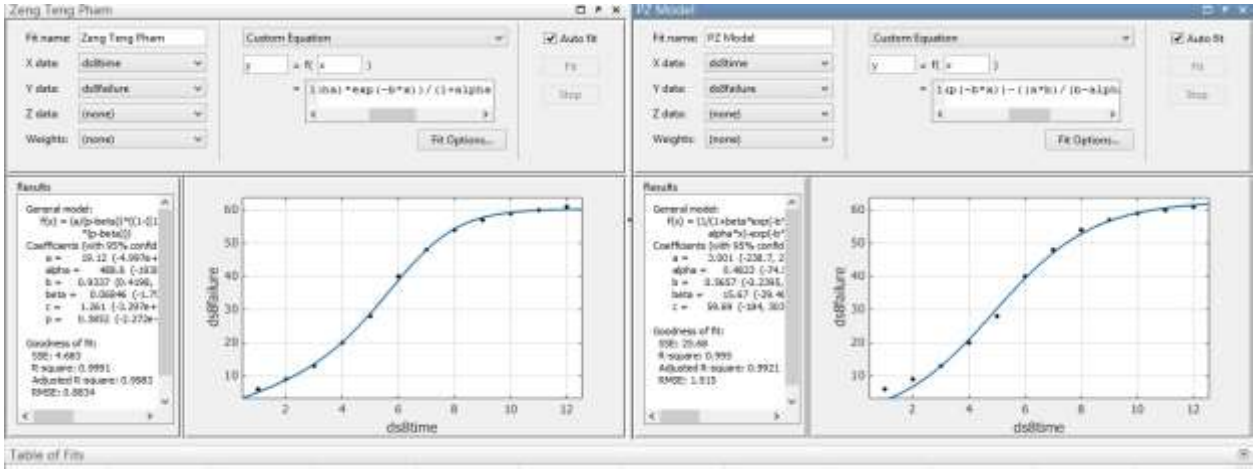

Fig 5.9: A comparison of model selected by our proposed method with the model suggested in literature for dataset 8

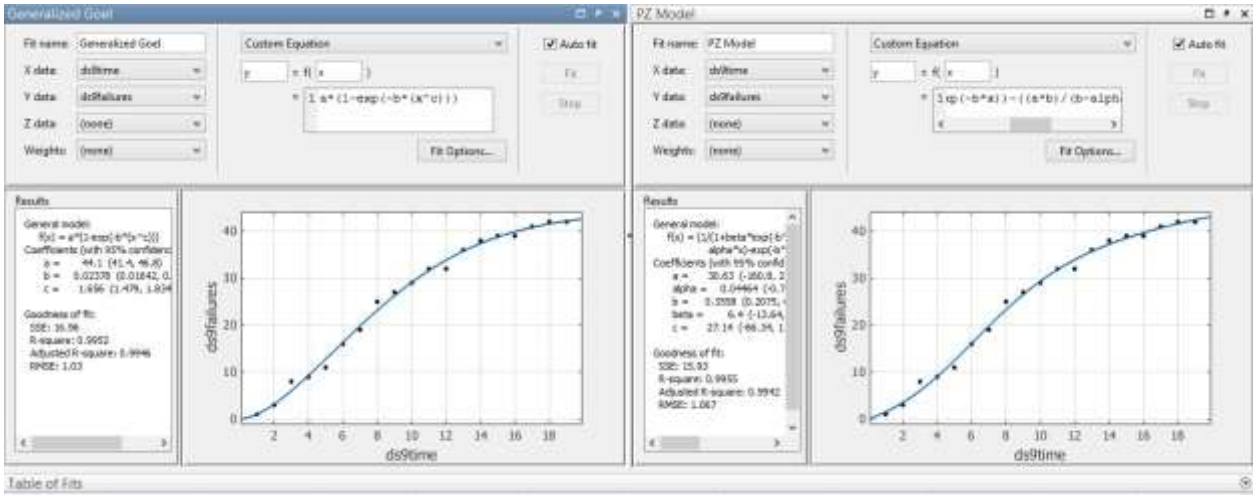

Fig 5.10: A comparison of model selected by our proposed method with the model suggested in literature for dataset 9

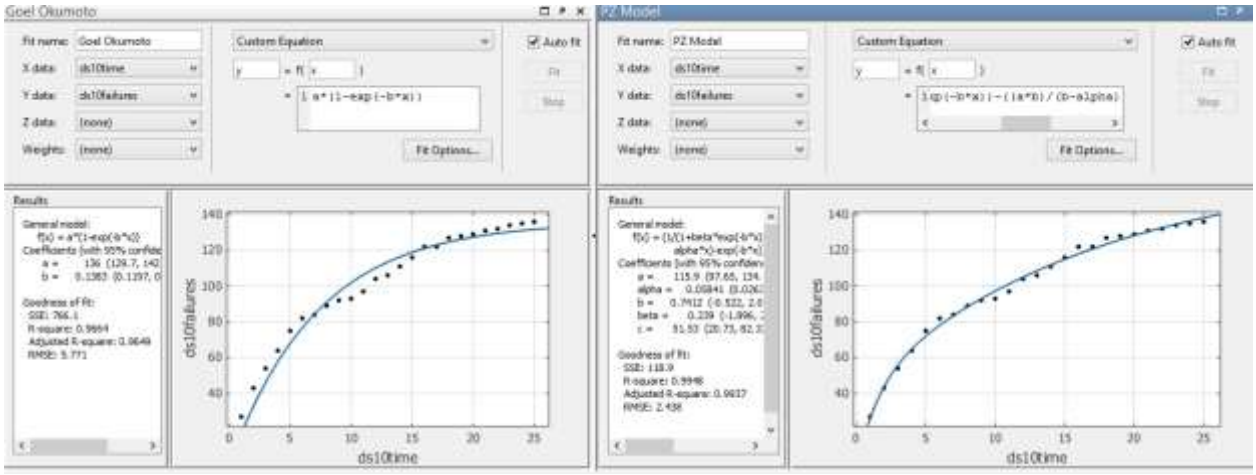

Fig 5.11: A comparison of model selected by our proposed method with the model suggested in literature for dataset 10

Table 5.1. Software Reliability Models Considered in Present study

S. No.	Model Name	Model Type	Mean Value Function (m(t))	Comments
1.	**Goel Okumoto [29]**	Concave	$m(t) = a(1 - e^{-bt})$ $a(t) = a$ $b(t) = b$	Also called exponential model
2.	**Generalized Goel [29]**	Concave	$m(t) = a\left(1 - e^{-bt^c}\right)$ $a(t) = a$ $b(t) = b$	Gives better goodness-of-fit than G-O Model. Same as GO-Model for c=1.
3.	**Modified Duane [53]**	Concave	$m(t) = a[1 - \left(\frac{b}{b+t}\right)^c]$ $a(t) = a$ $b(t) = b$	Assume independence of failure occurrences.
4.	**Musa- Okumoto [29]**	Concave	$m(t) = aln(1 + bt)$ $a(t) = a$ $b(t) = b$	Assumes failure intensity decreases exponentially with the expected number of errors experienced.
5.	**Yamada Exponential [79]**	Concave	$m(t) = a(1 - e^{r\alpha\left(1-e^{(\beta t)}\right)})$ $a(t) = a$ $b(t) = r\alpha\beta e^{-\beta t}$	Attempt to account for testing effort.
6.	**Gompert [53]**	S Shaped	$m(t) = ake^{-bt}$ $a(t) = a$ $b(t) = b$	Estimates software error contents. Also, predicts demand trend, economic growth or future population.
7.	**Inflection S Shaped [39]**	S-Shaped	$m(t) = \frac{a(1 - e^{-bt})}{1 + \beta e^{-bt}}$ $a(t) = a$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	Solves a technical condition with GO model. Becomes the same as GO if k=0.
8.	**Logistic Growth [39]**	S-Shaped	$m(t) = \frac{a}{1 + ke^{-bt}}$	Estimates the error content of software systems.
9.	**Delayed S Shaped [125]**	S-Shaped	$m(t) = a(1 - (1 + bt)e^{-bt}$ $a(t) = a$ $b(t) = \frac{b^2 t}{1 + bt}$	Modification of GO model to make it S-Shaped.
10.	**Yamada Imperfect Debugging Model I [126]**	S-Shaped	$m(t) = \frac{ab}{\alpha + b}(e^{\alpha t} - e^{bt})$ $a(t) = \alpha e^{\alpha t}$ $b(t) = b$	Assumes exponential fault content function and constant fault detection rate.
11.	**Yamada Imperfect Debugging Model II [126]**	S Shaped	$m(t) = a[1 - e^{-bt}]\left[1 - \frac{\alpha}{b}\right] + \alpha at$ $a(t) = a(1 + \alpha t)$ $b(t) = b$	Assumes constant introduction rate α and the fault detection rate.
12.	**Yamada Rayleigh [126]**	S-Shaped	$m(t) = a(1 - e^{-r\alpha\left(1-e^{(\beta t^2/2)}\right)})$ $a(t) = a$ $b(t) = r\alpha\beta t e^{-\beta t^2/2}$	Attempt to account for testing effort.
13.	**Pham Zhang IFD [89]**	S-Shaped	$m(t) = a - ae^{-bt}(1 + (b + d)t + bdt^2)$ $a(t) = a$ $b(t) = b$	Assumes a constant initial fault content function, and the imperfect fault detection rate combining the fault introduction phenomenon.
14.	**Zhang-Teng-Pham model (ZT Pham) [134]**	S-Shaped	$m(t) = \frac{a}{p - \beta}[\left(1 - \frac{(1 + \alpha)e^{-bt}}{1 + \alpha e^{-bt}}\right)^{\frac{c}{b}(p-\beta)}]$ $a(t) = \beta(t)m(t)$ $b(t) = \frac{c}{1+\alpha e^{-bt}}, \beta(t) = \beta$	Assume constant fault introduction rate, and the fault detection rate function is non-decreasing with an inflection S-shaped model.
15.	**Pham Nordman Zhang (PNZ Model) [87]**	S Shaped & Concave	$m(t) = \frac{a\left(1 - e^{-bt}\right)\left(1 - \frac{\alpha}{b}\right) + \alpha at}{1 + \beta e^{-bt}}$ $a(t) = a(1 + \alpha t)$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	Assume introduction rate is a linear function of testing time, and the fault detection rate function is non-decreasing with an Inflection S-shaped model.
16.	**Pham Zhang model (PZ Model) [88]**	S-Shaped & Concave	$m(t) = \frac{1}{(1 + \beta e^{-bt})}((c + a)\left(1 - e^{-bt}\right) - \frac{ab}{b - \alpha}\left(e^{-\alpha t} - e^{-bt}\right))$ $a(t) = c + a(1 - e^{-\alpha t})$ $b(t) = \frac{b}{1 + \beta e^{-bt}}$	Assume introduction rate is an exponential function of testing time, and the fault detection rate function is non-decreasing with an Inflection S-shaped model

Table 5.2 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 1 Using 1 to 10 Weeks Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9029	6.5244	0.5848	14
Generalized Goel	0.9838	2.8524	0.7939	8
Goel Okumoto	0.9716	3.5296	0.7303	11
Gompertz	0.9942	1.7071	0.9999	1
Inflection S Shaped	0.9716	3.7733	0.7147	12
Logistic Growth	0.9932	1.8494	0.9609	2
Modified Duane	0.9838	2.8487	0.7943	7
Musa Okumoto	0.9739	3.3822	0.7420	9
Pham Zhang IFD	0.9029	6.5244	0.5848	15
PNZ Model	0.9931	2.0010	0.9259	4
PZ Model	0.9910	2.5155	0.8376	6
Yamada Exponential	0.9270	6.5318	0.5968	13
Yamada Imperfect Debugging Model 1	0.9749	3.5437	0.7310	10
Yamada Imperfect Debugging Model II	0.9931	1.8526	0.9601	3
Yamada Rayleigh	0.8657	8.8602	0.5316	16
Zeng Teng Pham	0.9945	2.2041	0.8873	5

Table 5.2(b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 1 Using 1 to 15 Weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9520	6.1577	0.5939	13
Generalized Goel	0.9918	2.6472	0.7689	6
Goel Okumoto	0.9889	2.9614	0.7386	9
Gompert	0.9970	1.5966	0.9506	3
Inflection S Shaped	0.9889	3.0824	0.7291	12
Logistic Growth	0.9976	1.4404	0.9998	1
Modified Duane	0.9918	2.6452	0.7692	5
Musa Okumoto	0.9894	2.8985	0.7441	8
Pham Zhang IFD	0.9520	6.1577	0.5939	14
PNZ Model	0.9905	2.9752	0.7383	10
PZ Model	0.9939	2.3882	0.7995	4
Yamada Exponential	0.9522	6.6842	0.5848	15
Yamada Imperfect Debugging Model 1	0.9891	3.0557	0.7312	11
Yamada Imperfect Debugging Model II	0.9905	2.8486	0.7491	7
Yamada Rayleigh	0.9283	8.1860	0.5530	16
Zeng Teng Pham	0.9980	1.4964	0.9813	2

Table 5.2(c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 1 Using 1 to 20 Weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9689	5.2974	0.6000	14
Generalized Goel	0.9866	3.5762	0.6641	6
Goel Okumoto	0.9857	3.5928	0.6629	7
Gompert	0.9952	2.1534	0.7808	3
Inflection S Shaped	0.9890	3.2502	0.6823	5
Logistic Growth	0.9980	1.3822	0.9399	2
Modified Duane	0.9834	3.9838	0.6451	11
Musa Okumoto	0.9805	4.1955	0.6359	12
Pham Zhang IFD	0.9689	5.2974	0.6000	13
PNZ Model	0.9890	3.2502	0.6823	4
PZ Model	0.9857	3.6923	0.6583	10
Yamada Exponential	0.9462	7.3943	0.5560	16
Yamada Imperfect Debugging Model 1	0.9857	3.5928	0.6629	8
Yamada Imperfect Debugging Model II	0.9857	3.5928	0.6629	9
Yamada Rayleigh	0.9514	7.0300	0.5629	15
Zeng Teng Pham	0.9987	1.2171	1.0000	1

Table 5.3 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 2 Using 1 to 10 Weeks Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDE	RANK
Delayed S Shaped	0.8635	15.3126	0.8279	6
Generalized Goel	0.8692	16.0196	0.8148	8
Goel Okumoto	0.7962	18.7071	0.7255	13
Gompertz	0.9318	11.5728	0.9828	2
Inflection S Shaped	0.7968	19.9707	0.7068	14
Logistic Growth	0.8985	14.1138	0.8777	3
Modified Duane	0.7908	20.2631	0.6995	15
Musa Okumoto	0.7962	18.7068	0.7255	12
Pham Zhang IFD	0.8635	15.3126	0.8279	7
PNZ Model	0.8961	15.4268	0.8426	4
PZ Model	0.8828	17.9435	0.7845	9
Yamada Exponential	0.6926	26.5316	0.5816	16
Yamada Imperfect Debugging Model 1	0.9358	11.2254	1.0000	1
Yamada Imperfect Debugging Model II	0.8825	15.1828	0.8412	5
Yamada Rayleigh	0.8618	17.7890	0.7760	11
Zeng Teng Pham	0.8984	18.6803	0.7805	10

Table 5.3 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 2 Using 1 to 15 Weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9446	14.1557	0.8455	8
Generalized Goel	0.9533	13.5169	0.8671	7
Goel Okumoto	0.9222	16.7730	0.7773	14
Gompertz	0.9577	12.8732	0.8884	5
Inflection S Shaped	0.9635	11.9527	0.9221	3
Logistic Growth	0.9659	11.5512	0.9383	2
Modified Duane	0.9380	14.9701	0.8223	11
Musa Okumoto	0.9222	16.7727	0.7773	13
Pham Zhang IFD	0.9446	14.1557	0.8455	9
PNZ Model	0.9635	11.9527	0.9221	4
PZ Model	0.9635	13.0936	0.8846	6
Yamada Exponential	0.7628	31.8292	0.5509	16
Yamada Imperfect Debugging Model 1	0.8899	20.7644	0.7019	15
Yamada Imperfect Debugging Model II	0.9397	15.3682	0.8143	12
Yamada Rayleigh	0.9513	14.4279	0.8420	10
Zeng Teng Pham	0.9797	10.2888	1.0000	1

Table 5.3 (c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 2 Using 1 to 18 Weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9487	14.2424	0.8103	9
Generalized Goel	0.9649	12.1689	0.8745	6
Goel Okumoto	0.9243	17.3011	0.7399	12
Gompertz	0.9638	12.3483	0.8684	7
Inflection S Shaped	0.9724	10.7839	0.9276	3
Logistic Growth	0.9728	10.7122	0.9307	2
Modified Duane	0.9434	15.4539	0.7819	11
Musa Okumoto	0.9241	17.3288	0.7394	14
Pham Zhang IFD	0.9489	14.6787	0.8007	10
PNZ Model	0.9724	10.7839	0.9276	4
PZ Model	0.9724	11.5838	0.8977	5
Yamda Exponential	0.7888	30.8914	0.5523	16
Yamda Imperfect Debugging Model 1	0.9243	17.3011	0.7399	13
Yamda Imperfect Debugging Model II	0.9222	18.1136	0.7268	15
Yamda Rayleigh	0.9611	13.2560	0.8411	8
Zeng Teng Pham	0.9834	9.3434	1.0000	1

Table 5.4 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 3 Using 1 to 10 Weeks Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.7857	19.0288	0.5243	14
Generalized Goel	0.9516	9.6658	0.7325	4
Goel Okumoto	0.9246	11.2881	0.6825	8
Gompertz	0.9721	7.3463	0.8227	3
Inflection S Shaped	0.9246	11.2881	0.6825	9
Logistic Growth	0.9791	6.3577	0.8781	2
Modified Duane	0.9503	9.7998	0.7283	6
Musa Okumoto	0.9440	9.7264	0.7271	7
Pham Zhang IFD	0.7857	19.0288	0.5243	15
PNZ Model	0.9276	11.8249	0.6741	11
PZ Model	0.9633	9.9614	0.7309	5
Yamada Exponential	0.9343	12.1658	0.6717	13
Yamada Imperfect Debugging Model 1	0.9272	11.8585	0.6733	12
Yamada Imperfect Debugging Model II	0.9276	11.8249	0.6741	10
Yamada Rayleigh	0.6971	26.1222	0.4448	16
Zeng Teng Pham	0.9929	4.8963	1.0000	1

Table 5.4 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 3 Using 1 to 15 Weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.8210	15.6711	0.5999	15
Generalized Goel	0.9526	8.3930	0.8236	4
Goel Okumoto	0.9382	9.2109	0.7863	7
Gompertz	0.9720	6.4535	0.9348	2
Inflection S Shaped	0.9382	9.5870	0.7743	11
Logistic Growth	0.9785	5.6545	1.0000	1
Modified Duane	0.9425	9.2460	0.7874	6
Musa Okumoto	0.9133	10.9081	0.7259	13
Pham Zhang IFD	0.8210	15.6711	0.5999	14
PNZ Model	0.9393	9.9252	0.7648	12
PZ Model	0.9594	8.5056	0.8227	5
Yamada Exponential	0.9448	9.4654	0.7815	8
Yamada Imperfect Debugging Model 1	0.9392	9.5034	0.7774	10
Yamada Imperfect Debugging Model II	0.9393	9.5027	0.7775	9
Yamada Rayleigh	0.7450	20.3349	0.5197	16
Zeng Teng Pham	0.9723	7.4087	0.8784	3

Table 5.4 (c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 3 Using 1 to 17 Weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.8234	15.2521	0.6125	15
Generalized Goel	0.9562	7.8651	0.8596	4
Goel Okumoto	0.9388	8.9820	0.8047	9
Gompertz	0.9716	6.3349	0.9568	2
Inflection S Shaped	0.9388	8.9820	0.8047	10
Logistic Growth	0.9760	5.8170	1.0000	1
Modified Duane	0.9481	8.5631	0.8253	5
Musa Okumoto	0.9163	10.5043	0.7463	12
Pham Zhang IFD	0.8234	15.2521	0.6125	14
PNZ Model	0.9442	9.2142	0.7993	11
PZ Model	0.9648	7.6162	0.8761	3
Yamada Exponential	0.9478	8.9088	0.8120	6
Yamada Imperfect Debugging Model 1	0.9442	8.8785	0.8113	7
Yamada Imperfect Debugging Model II	0.9442	8.8790	0.8112	8
Yamada Rayleigh	0.7509	19.4593	0.5342	16
Zeng Teng Pham	0.9184	12.1050	0.7108	13

Table 5.5 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 4 Using 1 to 10 Weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9675	4.6520	0.9880	3
Generalized Goel	0.9685	4.8964	0.9640	6
Goel Okumoto	0.9541	5.5288	0.9033	11
Gompertz	0.9716	4.6473	0.9906	2
Inflection S Shaped	0.9708	4.7114	0.9835	5
Logistic Growth	0.9726	4.5650	1.0000	1
Modified Duane	0.9647	5.1797	0.9366	8
Musa Okumoto	0.9534	5.5690	0.9000	12
Pham Zhang IFD	0.9675	4.6520	0.9880	4
PNZ Model	0.9726	4.9350	0.9625	7
PZ Model	0.9708	5.5804	0.9081	10
Yamada Exponential	0.9321	7.7630	0.7732	16
Yamada Imperfect Debugging Model 1	0.9541	5.9106	0.8766	13
Yamada Imperfect Debugging Model II	0.9541	5.9106	0.8766	14
Yamada Rayleigh	0.9669	5.4181	0.9184	9
Zeng Teng Pham	0.9711	6.1982	0.8675	15

Table 5.5 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 4 Using 1 to 13 Weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9763	4.0570	0.9797	4
Generalized Goel	0.9779	4.1108	0.9742	5
Goel Okumoto	0.9533	5.6901	0.8298	12
Gompertz	0.9800	3.9074	1.0000	1
Inflection S Shaped	0.9793	3.9758	0.9910	2
Logistic Growth	0.9792	3.9846	0.9899	3
Modified Duane	0.9695	4.8259	0.8995	11
Musa Okumoto	0.9466	6.0891	0.8038	15
Pham Zhang IFD	0.9763	4.2550	0.9573	7
PNZ Model	0.9800	4.1216	0.9740	6
PZ Model	0.9793	4.4427	0.9394	8
Yamada Exponential	0.9440	6.8927	0.7651	16
Yamada Imperfect Debugging Model 1	0.9533	5.6901	0.8298	14
Yamada Imperfect Debugging Model II	0.9533	5.6901	0.8298	13
Yamada Rayleigh	0.9767	4.4496	0.9374	9
Zeng Teng Pham	0.9793	4.7508	0.9109	10

Table 5.6 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 5 Using 1 to 10 Weeks Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9887	4.9787	0.8749	11
Generalized Goel	0.9943	3.7624	0.9999	1
Goel Okumoto	0.9894	4.8007	0.8893	7
Gompertz	0.9919	4.4953	0.9172	6
Inflection S Shaped	0.9937	3.9535	0.9754	3
Logistic Growth	0.9858	5.9527	0.8117	14
Modified Duane	0.9943	3.7747	0.9983	2
Musa Okumoto	0.9877	5.1814	0.8597	13
Pham Zhang IFD	0.9887	4.9787	0.8749	12
PNZ Model	0.9937	3.9535	0.9754	4
PZ Model	0.9945	4.3856	0.9289	5
Yamada Exponential	0.9785	7.9040	0.7300	15
Yamada Imperfect Debugging Model 1	0.9894	4.8007	0.8893	9
Yamada Imperfect Debugging Model II	0.9894	4.8007	0.8893	8
Yamada Rayleigh	0.9741	8.6858	0.7063	16
Zeng Teng Pham	0.9944	4.9532	0.8797	10

Table 5.6 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 5 Using 1 to 15 Weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9774	7.7779	0.7857	12
Generalized Goel	0.9903	5.2958	0.9299	3
Goel Okumoto	0.9891	5.3966	0.9212	4
Gompertz	0.9842	6.7668	0.8330	11
Inflection S Shaped	0.9895	5.5095	0.9128	7
Logistic Growth	0.9745	8.5982	0.7562	15
Modified Duane	0.9920	4.8110	0.9742	2
Musa Okumoto	0.9865	6.0041	0.8771	9
Pham Zhang IFD	0.9774	7.7779	0.7857	13
PNZ Model	0.9934	4.5695	1.0000	1
PZ Model	0.9893	6.0900	0.8731	10
Yamada Exponential	0.9787	8.2020	0.7712	14
Yamada Imperfect Debugging Model 1	0.9891	5.3966	0.9212	5
Yamada Imperfect Debugging Model II	0.9891	5.3966	0.9212	6
Yamada Rayleigh	0.9588	11.4093	0.6829	16
Zeng Teng Pham	0.9906	5.7044	0.8991	8

Table 5.6 (c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 5 Using 1 to 21 Weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9748	8.7277	0.7293	14
Generalized Goel	0.9936	4.5212	0.9613	2
Goel Okumoto	0.9929	4.6382	0.9493	3
Gompertz	0.9867	6.5203	0.8163	11
Inflection S Shaped	0.9932	4.6501	0.9484	7
Logistic Growth	0.9764	8.6751	0.7316	13
Modified Duane	0.9945	4.1759	1.0000	1
Musa Okumoto	0.9864	6.4110	0.8216	10
Pham Zhang IFD	0.9748	8.7277	0.7293	15
PNZ Model	0.9936	4.6480	0.9488	6
PZ Model	0.9921	5.3337	0.8902	9
Yamada Exponential	0.9792	8.3824	0.7414	12
Yamada Imperfect Debugging Model 1	0.9929	4.6382	0.9493	4
Yamada Imperfect Debugging Model II	0.9929	4.6382	0.9493	5
Yamada Rayleigh	0.9524	12.6768	0.6435	16
Zeng Teng Pham	0.9938	4.7365	0.9404	8

Table 5.7 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 6 Using 1 to 50 Days Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9639	25.2137	0.8316	11
Generalized Goel	0.9727	22.1682	0.8830	3
Goel Okumoto	0.9705	22.4193	0.8775	8
Gompertz	0.9620	26.1443	0.8184	14
Inflection S Shaped	0.9720	22.4219	0.8782	7
Logistic Growth	0.9484	30.4548	0.7649	16
Modified Duane	0.9733	21.9068	0.8879	2
Musa Okumoto	0.9720	22.1828	0.8824	4
Pham Zhang IFD	0.9639	25.2137	0.8316	12
PNZ Model	0.9838	17.2308	1.0000	1
PZ Model	0.9722	22.4924	0.8771	9
Yamada Exponential	0.9634	25.9326	0.8218	13
Yamada Imperfect Debugging Model 1	0.9722	22.3518	0.8795	5
Yamada Imperfect Debugging Model II	0.9721	22.4051	0.8785	6
Yamada Rayleigh	0.9505	30.1567	0.7687	15
Zeng Teng Pham	0.9728	22.8358	0.8717	10

Table 5.7 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 6 Using 1 to 75 Days Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9812	21.2380	0.9681	1
Generalized Goel	0.9836	19.9366	1.0000	2
Goel Okumoto	0.9743	24.7918	0.8974	3
Gompertz	0.9807	21.6527	0.9589	4
Inflection S Shaped	0.9833	20.1395	0.9948	5
Logistic Growth	0.9746	24.8363	0.8968	6
Modified Duane	0.9824	20.7064	0.9808	7
Musa Okumoto	0.9706	26.5586	0.8687	8
PNZ Model	0.9833	20.1395	0.9948	9
PZ Model	0.9746	25.1940	0.8911	10
Pham Zhang IFD	0.9812	21.2380	0.9681	11
Yamada Exponential	0.9734	25.6225	0.8838	12
Yamada Imperfect Debugging Model 1	0.9743	24.7918	0.8974	13
Yamada Imperfect Debugging Model II	0.9743	24.7918	0.8974	14
Yamada Rayleigh	0.9720	26.2761	0.8734	15
Zeng Teng Pham	0.9835	20.4680	0.9869	16

Table 5.7 (c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 6 Using 1 to 111 Days Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9855	18.1490	0.9746	6
Generalized Goel	0.9870	17.2818	0.9992	3
Goel Okumoto	0.9647	28.3258	0.7933	13
Gompertz	0.9853	18.3488	0.9693	8
Inflection S Shaped	0.9870	17.2571	1.0000	1
Logistic Growth	0.9820	20.3102	0.9223	9
Modified Duane	0.9764	23.2386	0.8659	11
Musa Okumoto	0.9367	37.9202	0.7020	16
Pham Zhang IFD	0.9855	18.1490	0.9746	7
PNZ Model	0.9870	17.2571	1.0000	2
PZ Model	0.9871	17.3594	0.9970	4
Yamada Exponential	0.9583	31.0583	0.7632	15
Yamada Imperfect Debugging Model 1	0.9647	28.3258	0.7933	14
Yamada Imperfect Debugging Model II	0.9647	28.3258	0.7933	12
Yamada Rayleigh	0.9791	21.9917	0.8883	10
Zeng Teng Pham	0.9871	17.4377	0.9948	5

Table 5.8 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 7 Using 1 to 10 weeks Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9635	5.5454	0.6766	12
Generalized Goel	0.9703	5.3427	0.6874	11
Goel Okumoto	0.9849	3.5606	0.7948	6
Gompertz	0.9939	2.4237	0.9400	2
Inflection S Shaped	0.9850	3.7936	0.7764	9
Logistic Growth	0.9953	2.1361	1.0000	1
Modified Duane	0.9870	3.5438	0.7972	5
Musa Okumoto	0.9849	3.5639	0.7945	7
Pham Zhang IFD	0.9635	5.5454	0.6766	13
PNZ Model	0.9882	3.6442	0.7895	8
PZ Model	0.9868	4.2096	0.7495	10
Yamada Exponential	0.9289	8.9359	0.5862	16
Yamada Imperfect Debugging Model 1	0.9885	3.3218	0.8181	3
Yamada Imperfect Debugging Model II	0.9882	3.3742	0.8130	4
Yamada Rayleigh	0.9500	7.4917	0.6198	15
Zeng Teng Pham	0.9667	7.4884	0.6283	14

Table 5.8 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 7 Using 1 to 15 weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9855	4.3726	0.7141	12
Generalized Goel	0.9905	3.6828	0.7579	6
Goel Okumoto	0.9862	4.2644	0.7201	9
Gompertz	0.9955	2.5354	0.8786	2
Inflection S Shaped	0.9888	3.9986	0.7364	7
Logistic Growth	0.9974	1.9249	1.0000	1
Modified Duane	0.9885	4.0491	0.7332	8
Musa Okumoto	0.9856	4.3647	0.7146	11
Pham Zhang IFD	0.9855	4.3726	0.7141	13
PNZ Model	0.9934	3.0625	0.8123	3
PZ Model	0.9926	3.5726	0.7670	4
Yamada Exponential	0.9799	5.5997	0.6631	15
Yamada Imperfect Debugging Model 1	0.9862	4.2644	0.7201	10
Yamada Imperfect Debugging Model II	0.9862	4.4385	0.7112	14
Yamada Rayleigh	0.9767	6.0354	0.6491	16
Zeng Teng Pham	0.9933	3.5775	0.7670	5

Table 5.8 (c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 7 Using 1 to 19 weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9900	3.8325	0.7494	7
Generalized Goel	0.9929	3.3436	0.7879	6
Goel Okumoto	0.9824	5.0992	0.6827	13
Gompertz	0.9967	2.2817	0.9249	2
Inflection S Shaped	0.9889	4.1764	0.7280	9
Logistic Growth	0.9976	1.9410	1.0000	1
Modified Duane	0.9876	4.4150	0.7148	10
Musa Okumoto	0.9781	5.6785	0.6612	16
Pham Zhang IFD	0.9900	3.8325	0.7494	8
PNZ Model	0.9955	2.6696	0.8625	4
PZ Model	0.9955	2.7571	0.8509	5
Yamada Exponential	0.9810	5.6421	0.6637	15
Yamada Imperfect Debugging Model 1	0.9824	5.0992	0.6827	14
Yamada Imperfect Debugging Model II	0.9824	5.0992	0.6827	12
Yamada Rayleigh	0.9845	5.0946	0.6839	11
Zeng Teng Pham	0.9964	2.6375	0.8674	3

Table 5.9(a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 8 Using 1 to 10 weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9845	2.7338	0.6994	8
Generalized Goel	0.9887	2.4965	0.7211	7
Goel Okumoto	0.9653	4.0928	0.6214	13
Gompertz	0.9911	2.2199	0.7504	6
Inflection S Shaped	0.9940	1.8260	0.8066	2
Logistic Growth	0.9977	1.1264	1.0000	1
Modified Duane	0.9818	3.1675	0.6699	11
Musa Okumoto	0.9653	4.0932	0.6214	14
Pham Zhang IFD	0.9845	2.9225	0.6861	9
PNZ Model	0.9940	1.8260	0.8066	3
PZ Model	0.9940	2.1612	0.7587	4
Yamada Exponential	0.8896	8.4335	0.5126	16
Yamada Imperfect Debugging Model 1	0.9680	4.2001	0.6192	15
Yamada Imperfect Debugging Model II	0.9747	3.7377	0.6392	12
Yamada Rayleigh	0.9858	3.0267	0.6801	10
Zeng Teng Pham	0.9949	2.2159	0.7528	5

Table 5.9(b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 8 Using 1 to 12 weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9811	3.1020	0.6352	9
Generalized Goel	0.9908	2.2804	0.6920	6
Goel Okumoto	0.9524	4.9235	0.5675	13
Gompertz	0.9907	2.2985	0.6904	7
Inflection S Shaped	0.9951	1.6714	0.7656	3
Logistic Growth	0.9981	1.0366	0.9309	2
Modified Duane	0.9762	3.6696	0.6104	11
Musa Okumoto	0.9513	4.9760	0.5660	14
Pham Zhang IFD	0.9822	3.1730	0.6325	10
PNZ Model	0.9951	1.6714	0.7656	4
PZ Model	0.9951	1.8952	0.7340	5
Yamada Exponential	0.9319	6.5807	0.5344	16
Yamada Imperfect Debugging Model 1	0.9524	4.9235	0.5675	12
Yamada Imperfect Debugging Model II	0.9524	5.1898	0.5628	15
Yamada Rayleigh	0.9904	2.4737	0.6764	8
Zeng Teng Pham	0.9991	0.8944	1.0000	1

Table 5.10(a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 9 Using 1 to 10 weeks Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9865	1.2326	0.9818	3
Generalized Goel	0.9876	1.2620	0.9711	6
Goel Okumoto	0.9598	2.1230	0.7657	15
Gompertz	0.9877	1.2574	0.9729	5
Inflection S Shaped	0.9889	1.1913	0.9998	1
Logistic Growth	0.9873	1.2758	0.9659	7
Modified Duane	0.9872	1.2810	0.9640	8
Musa Okumoto	0.9598	2.1226	0.7658	14
Pham Zhang IFD	0.9865	1.2326	0.9818	4
PNZ Model	0.9889	1.1913	0.9998	2
PZ Model	0.9889	1.4102	0.9222	10
Yamada Exponential	0.9059	3.7529	0.6166	16
Yamada Imperfect Debugging Model II	0.9869	1.2958	0.9585	9
Yamada Imperfect Debugging Model 1	0.9819	1.5244	0.8870	11
Yamada Rayleigh	0.9737	1.9839	0.7924	13
Zeng Teng Pham	0.9893	1.5531	0.8835	12

Table 5.10(b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 9 Using 1 to 15 weeks Data (i.e. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9930	1.1268	0.9996	1
Generalized Goel	0.9933	1.1499	0.9897	4
Goel Okumoto	0.9802	1.8955	0.7904	13
Gompertz	0.9932	1.1537	0.9880	5
Inflection S Shaped	0.9932	1.1563	0.9869	6
Logistic Growth	0.9898	1.4148	0.8962	11
Modified Duane	0.9922	1.2395	0.9537	7
Musa Okumoto	0.9802	1.8959	0.7903	14
Pham Zhang IFD	0.9930	1.1268	0.9996	2
PNZ Model	0.9938	1.1495	0.9901	3
PZ Model	0.9934	1.2444	0.9525	8
Yamada Exponential	0.9797	2.0843	0.7632	16
Yamada Imperfect Debugging Model 1	0.9802	1.9730	0.7787	15
Yamada Imperfect Debugging Model II	0.9825	1.8534	0.7983	12
Yamada Rayleigh	0.9914	1.3571	0.9139	10
Zeng Teng Pham	0.9934	1.3096	0.9300	9

Table 5.10(c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 9 Using 1 to 19 weeks Data (i.e. 100% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9947	1.0465	0.9916	4
Generalized Goel	0.9952	1.0297	0.9999	1
Goel Okumoto	0.9760	2.2362	0.7204	12
Gompertz	0.9952	1.0346	0.9975	3
Inflection S Shaped	0.9950	1.0568	0.9869	5
Logistic Growth	0.9910	1.4138	0.8619	11
Modified Duane	0.9926	1.2781	0.9014	10
Musa Okumoto	0.9743	2.3149	0.7117	15
Pham Zhang IFD	0.9948	1.0763	0.9780	6
PNZ Model	0.9955	1.0316	0.9991	2
PZ Model	0.9953	1.0936	0.9707	7
Yamada Exponential	0.9433	3.6599	0.6144	16
Yamada Imperfect Debugging Model 1	0.9760	2.2362	0.7204	14
Yamada Imperfect Debugging Model II	0.9760	2.2362	0.7204	13
Yamada Rayleigh	0.9937	1.2198	0.9212	9
Zeng Teng Pham	0.9953	1.1273	0.9566	8

Table 5.11(a). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 10 Using 1 to 10 hours Data (i.e. 50% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.9408	5.8418	0.5749	14
Generalized Goel	0.9962	1.5879	0.8801	4
Goel Okumoto	0.9947	1.7498	0.8441	6
Gompertz	0.9978	1.2103	0.9998	1
Inflection S Shaped	0.9947	1.7498	0.8441	7
Logistic Growth	0.9962	1.5830	0.8813	3
Modified Duane	0.9949	1.8375	0.8277	11
Musa Okumoto	0.9911	2.2664	0.7635	13
Pham Zhang IFD	0.9408	5.8418	0.5749	15
PNZ Model	0.9949	1.8280	0.8295	9
PZ Model	0.9969	1.6813	0.8593	5
Yamada Exponential	0.9952	1.9191	0.8139	12
Yamada Imperfect Debugging Model 1	0.9949	1.8299	0.8291	10
Yamada Imperfect Debugging Model II	0.9949	1.8280	0.8295	8
Yamada Rayleigh	0.8984	8.8354	0.5185	16
Zeng Teng Pham	0.9981	1.4684	0.9121	2

Table 5.11(b). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 10 Using 1 to 15 hours Data (i.e. Approx. 75% of data)

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.8972	8.5568	0.5621	14
Generalized Goel	0.9911	2.6248	0.8611	7
Goel Okumoto	0.9756	4.1643	0.7191	11
Gompertz	0.9791	4.0148	0.7293	9
Inflection S Shaped	0.9756	4.3343	0.7101	12
Logistic Growth	0.9700	4.8084	0.6856	13
Modified Duane	0.9913	2.5842	0.8669	6
Musa Okumoto	0.9925	2.3157	0.9102	5
Pham Zhang IFD	0.8972	8.5568	0.5621	15
PNZ Model	0.9948	2.0946	0.9549	3
PZ Model	0.9948	2.1983	0.9334	4
Yamada Exponential	0.9797	4.1325	0.7229	10
Yamada Imperfect Debugging Model 1	0.9953	1.9066	1.0000	1
Yamada Imperfect Debugging Model II	0.9947	2.0176	0.9722	2
Yamada Rayleigh	0.8528	11.1290	0.5141	16
Zeng Teng Pham	0.9904	3.1482	0.8003	8

Table 5.11(c). Estimated Values of RMSE and RSqure (RSq) Attributes for Each SRGM for Dataset 10 Using 1 to 25 hours Data (i.e. Approx. 100% of data).

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
Delayed S Shaped	0.8646	11.5859	0.5398	14
Generalized Goel	0.9935	2.5908	0.9699	3
Goel Okumoto	0.9664	5.7714	0.6970	12
Gompertz	0.9823	4.2840	0.7783	9
Inflection S Shaped	0.9664	5.7714	0.6970	13
Logistic Growth	0.9752	5.0659	0.7308	10
Modified Duane	0.9934	2.6131	0.9659	4
Musa Okumoto	0.9933	2.5770	0.9723	2
Pham Zhang IFD	0.8646	11.5859	0.5398	15
PNZ Model	0.9911	3.1057	0.8907	7
PZ Model	0.9948	2.4383	1.0000	1
Yamada Exponential	0.9759	5.1109	0.7291	11
Yamada Imperfect Debugging Model 1	0.9893	3.3377	0.8625	8
Yamada Imperfect Debugging Model II	0.9911	3.0343	0.8999	6
Yamada Rayleigh	0.8114	14.3102	0.4930	16
Zeng Teng Pham	0.9933	2.8362	0.9291	5

Table 5.12. Comparison of results obtained using proposed method with the models suggested in literature

S.No.	Dataset	Data Used Upto	Model Selected	Model suggested in literature
1	**Dataset 1**	10 weeks	Gompertz	Inflection S Shaped [107]
		15 weeks	Logistic Growth	
		20 weeks	ZT Pham	
2	**Dataset 2**	10 weeks	Logistic Growth	Delayed S Shaped [113]
		15 weeks	Logistic Growth	
		18 weeks	Logistic Growth	
3	**Dataset 3**	10 weeks	ZT Pham	Yamada Exponential [113]
		15 weeks	Logistic Growth	
		17 weeks	Logistic Growth	
4	**Dataset 4**	7 weeks	Logistic Growth	Delayed S Shaped [113]
		13 weeks	Gompertz	
5	**Dataset 5**	10 weeks	Generalized Goel	PZ Model [27]
		15 weeks	Generalized Goel	
		21 weeks	Generalized Goel	
6	**Dataset 6**	55 Days	Generalized Goel	PZ Model [8]
		84 Days	Generalized Goel	
		111 Days	Inflection S Shaped	
7	**Dataset 7**	10 weeks	Logistic Growth	PZ Model [134]
		15 weeks	Logistic Growth	
		19 weeks	Logistic Growth	
8	**Dataset 8**	7 weeks	Logistic Growth	PZ Model [134]
		12 weeks	ZT Pham	
9	**Dataset 9**	10 weeks	Inflection S Shaped	PZ Model [134]
		15 weeks	Delayed S Shaped	
		19 weeks	Generalized Goel	
10	**Dataset 10**	13 Hours	Generalized Goel	Goel Okumoto [89]
		19 Hours	Gompertz	
		25 Hours	PZ Model	

CHAPTER 6

CHAPTER 6

ON THE USE OF PERFECT DEBUGGING VERSUS IMPERFECT DEBUGGING MODELS FOR ANALYSIS OF OBSERVED FAILURE DATA

This chapter addresses the issue of the choice of a perfect debugging model versus an imperfect debugging model in capturing actual behavior of observed failures in software in its testing stage. For this purpose the approach proposed in chapter 5 was used and tested on eleven datasets taken from literature.

6.1 MOTIVATION

Development of software is a time consuming and costly process. One is often interested in estimating its likely release date so that further necessary planning can be done in advance. Since the users of the software clamor for faster deliveries and there is also a constraint on the development cost, software is usually released when a desired reliability level has been achieved. For this purpose usually some theoretical models (commonly known as software reliability growth models (SRGMs)) are fitted on the available test data to estimate its likely release date.

These models include both perfect debugging models as well as imperfect debugging models. Since most of the software testing processes are by nature imperfect debugging ones, it would be naturally more appropriate to use imperfect debugging models while estimating the software release date [56], [88], [93]. However imperfect debugging models being usually more complex, perfect debugging models have been often used for this purpose [16], [29], [125], [30], [83], [120].

Many software reliability growth models have been proposed in literature [29], [87], [125], [134]. Most of these models are based on the assumption of perfect debugging i.e., when a failure is observed, the corresponding fault is identified and removed and no new faults get introduced. This assumption is, however, known to be unrealistic. Goel [30] mentions it as one of the major practical limitations of

perfect debugging models because on account of the complexity and incomplete understanding of the software, the testing team may not be able to remove the fault completely on its detection and the original fault may remain in the software in some form resulting in the phenomenon known as imperfect debugging.

There are two types of imperfect debugging possibilities [56]. In the first one on observing a failure, the corresponding fault is identified. However because of incomplete understanding of the nature of the working of software, the detected fault does not get removed completely. As a result the fault count in the software remains unchanged even after this removal action. Kapur [54] calls it imperfect fault debugging. In the second case, on a failure a fault is identified and is removed with certainty. However, inadvertently some new faults creeps in the software during the removal process [83]. This type of imperfect debugging can lead to further increase in the fault content of the software. This is known as error generation. Since a software cannot be tested indefinitely in order to make it completely bug free, therefore it is often released when specified reliability level of the software is expected to have been achieved.

In recent years several perfect and imperfect debugging SRGMs have been proposed and studied in literature [93], [111]. Kapur et al. [59] unified the existing SRGMs under a general formulation known as generalized non-homogeneous Poisson process (GNHPP) models. These account for both types of imperfect debugging and perfect debugging models. With their formulation one can generate both perfect debugging and imperfect debugging existing models as well as develop new ones.

The rest of this chapter is organized as follows. Section 6.2 presents a brief overview of the generalized non-homogeneous Poisson Process models which include both perfect and imperfect debugging models. The assumptions made while formulating these models are also mentioned there. In section 6.3 we present the proposed method of selecting an appropriate model for a software in its testing stage in brief. It next applied in section 6.4 to same eleven real software failure datasets

taken from literature and given in appendix B and compare its performance on some of the currently used perfect and imperfect debugging models. Conclusions based on the present study are next drawn in section 6.5.

6.2 GENERALIZED NON-HOMOGENEOUS POISSON PROCESS MODELS

The NHPP models are based on the assumption that a software system is subject to failures at random times caused by manifestation of the remaining faults in the system. Therefore, NHPP models are often used to describe the failure phenomenon during the testing phase. The counting process $\{N(t), t \geq 0\}$ of an NHPP process is as follows [89].

$$\{N(t) = k\} = \frac{(m(t))^k}{k!} e^{-m(t)}, k = 0,1,2, \ldots \quad (6.1)$$

$$m(t) \int_0^t \lambda(x)dx \quad (6.2)$$

The intensity function $\lambda(x)$ (or mean value function $(m(t))$ is the basic building block of all NHPP models currently in use in software reliability engineering literature. These NHPP models are based upon the following basic assumptions [89].

i. The failure observation and fault removal phenomenon is modeled using a NHPP.
ii. Software is subject to failures during execution caused by faults remaining in the software.
iii. The subsequent failure removal rate is affected by the faults remaining in the software.
iv. Each time a failure is observed, immediately debugging effort takes place to find the cause of the failure and effort is made to remove it.
v. When a software failure occurs, instantaneous repair effort starts and then either a the fault content is reduced by one with probability p, or (b) the fault content remains unchanged with probability $(1 - p)$.

vi. During the fault removal process, whether the fault is removed successfully or not, new faults are generated with a constant probability α.

Whereas as assumptions i to iv are common to both perfect and imperfect debugging models, assumptions v, and vi capture the effect of imperfect debugging, and error generation respectively.

According to Kapur et al. [136] GNHPP SRGMs incorporate two testing processes; the failure observation process defined by $F(t)$ distribution function and the fault removal process defined by the $G(t)$ distribution function and these can be represented in a generalized framework as:

$$m(t) = \frac{A}{1-\alpha}\left[1 - \left(1 - (F \otimes G)(t)\right)^{p(1-\alpha)}\right] \qquad (6.3)$$

Here A is the initial number of faults lying dormant in the software when the testing starts, α is the rate at which the faults get introduced during the debugging process ($0 \leq \alpha \leq 1$) and p is the probability of fault removal on a failure (i.e., the probability of perfect debugging). Substituting different types of distribution functions in (6.3) one can obtain different mean value functions corresponding to different forms of distribution functions $F(t)$ and $G(t)$. Assuming $G(t) = 1$ and substituting $p = 1$, and $\alpha = 0$ in (6.3) perfect non-homogeneous Poisson process models can be obtained. Following [59] some of the commonly used NHPP models which are commonly used and can be obtained using (6.3) are listed in table 6.1. In table 6.2 we list the perfect debugging models corresponding to specific choice of $F(t)$ and *G(t)* functions.

6.3 PROPOSED METHOD FOR SELECTING AN APPROPRIATE GENERALIZED GROWTH MODEL

In order to decide the most appropriate model for a test data different approaches have been proposed in literature [27], [54], [56], [107], [120], [16]. Our own experience has been observed that the approach proposed in chapter 5 can be quite effective. We may state in brief the method which is used for selecting a most appropriate software reliability models.

i. During testing phase of the software under development keep periodic record of failure test data from time to time.
ii. Looking at the available test data in hand, decide the models which are more likely to fit this data.
iii. Using the curve fitting tool of Matlab, fit these models to the available data and estimate the values of goodness of fit parameters namely RSq and RMSE for each of these models.
iv. Rank the competing models using ranking parameter. This is same as given in (5.1) of chapter 5.

6.4 APPLICATION ON TEST DATA

In this section we present our experience of using this method for choosing most appropriate theoretical model on eleven data sets taken from literature as listed in Appendix B. The main objective of this study has been to compare the appropriateness of proposed approach on various test data sets for selecting suitable perfect or imperfect debugging models. The values of RSq, RMSE is estimated by using curve fitting toolbox of Matlab first and then using (6.4) their rank index is calculated. According to the value of rank index the models are then ranked according to descending value of rank index (i.e. model with larger rank index is assigned rank one and so on). The estimated value of RMSE, RSq, rank index and their ranks for dataset 1 to 11 are given in table 6.3 and 6.4. Based on the study the rank one model is considered the best model which can further used for predictions of release date of software. The obtained results are also compared with the results available in literature.

We have also comparatively ranked the best fitted perfect debugging as well as imperfect debugging models for each dataset as given in table 6.5 and fig 6.1 to 6.10. The relative rank shows that except for dataset 5 imperfect debugging models seem to be more appropriate. In the case of dataset 11 the model suggested in literature is also same as the imperfect debugging model based on our proposed criteria.

6.5 ANALYSIS OF RESULTS AND CONCLUSIONS

The fitting of the models selected by proposed approach to the eleven datasets listed in appendix B are shown graphically in fig 6.1 to 6.10. A critical observation of graphs shows that except for dataset 5 all other datasets, the imperfect debugging models shows better filling as compared to the perfect debugging models. In fact SRGM-4.1 model fits excellently for datasets 1, 2, 3, 5, 7, 8 and 11. Surprisingly only in the case of dataset 5 perfect debugging models seems to give better fit to actual valves as compared to imperfect debugging model.

It thus appears that even though in general imperfect debugging models seem more appropriate, both perfect and imperfect debugging models which are likely to fit the failure data should be tried and the better of two selected has been chosen.

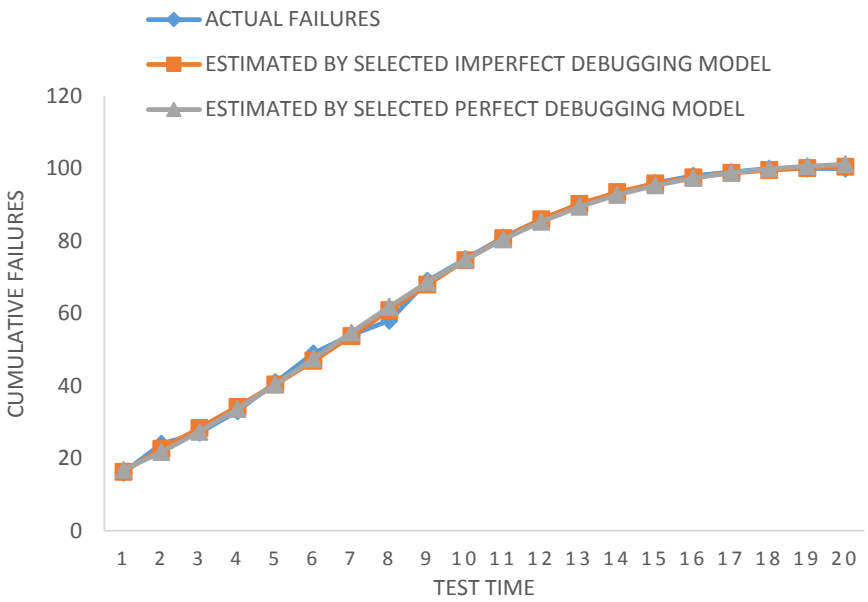

Fig 6.1: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 1

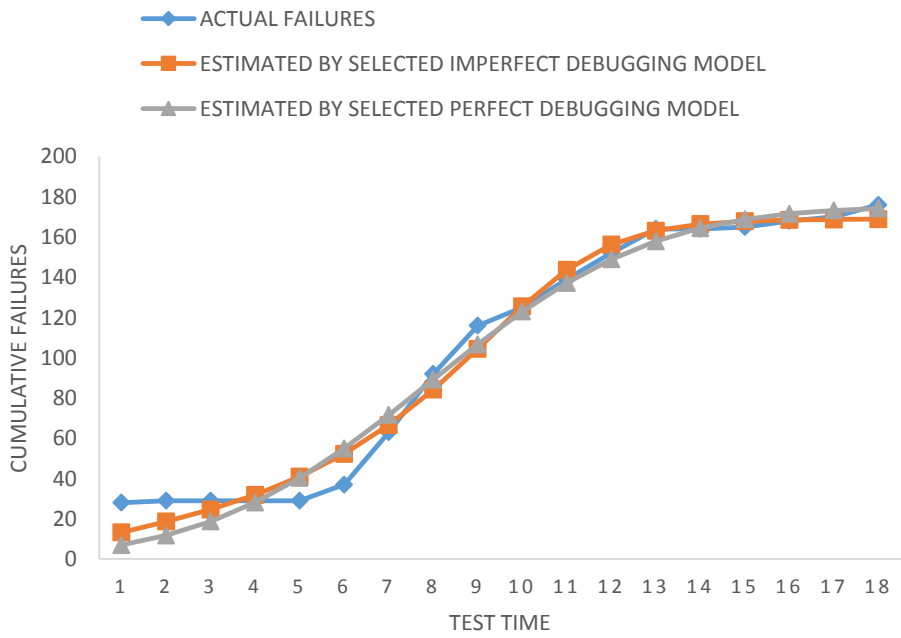

Fig 6.2: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 2

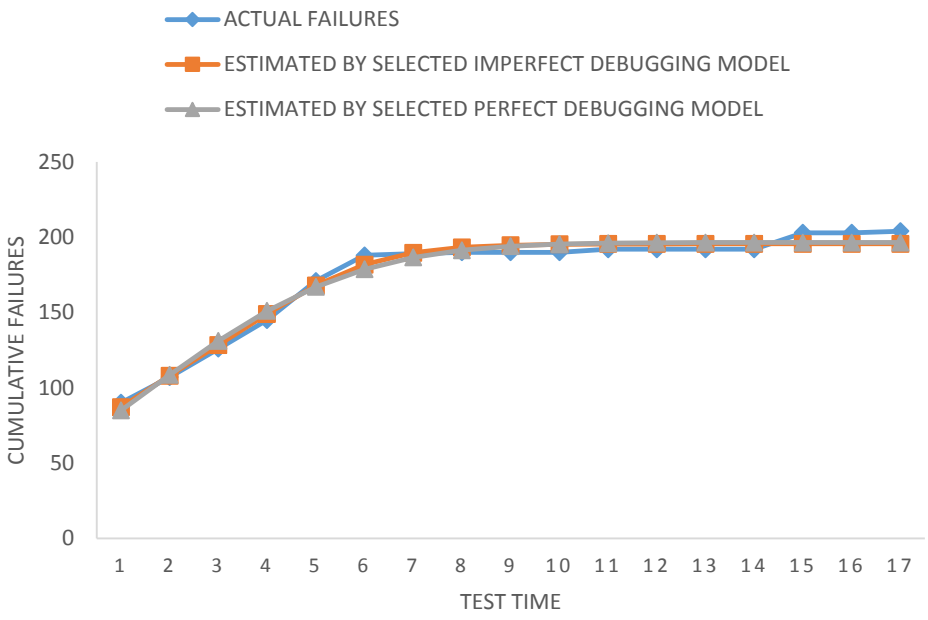

Fig 6.3: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 3

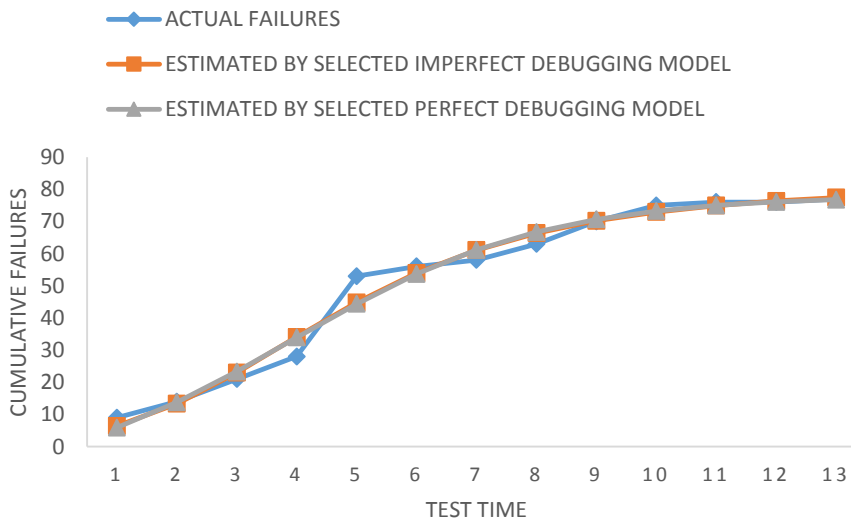

Fig 6.4: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 4

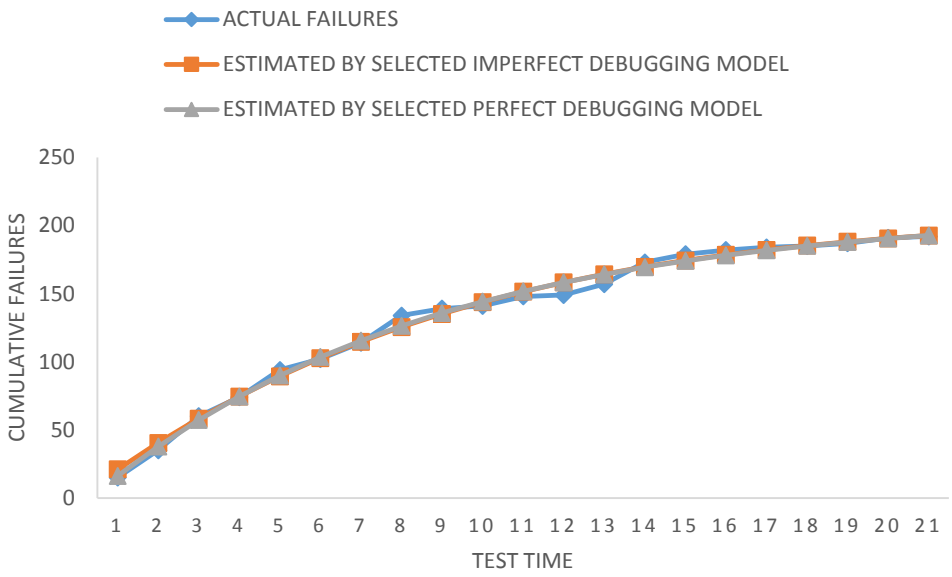

Fig 6.5: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 5

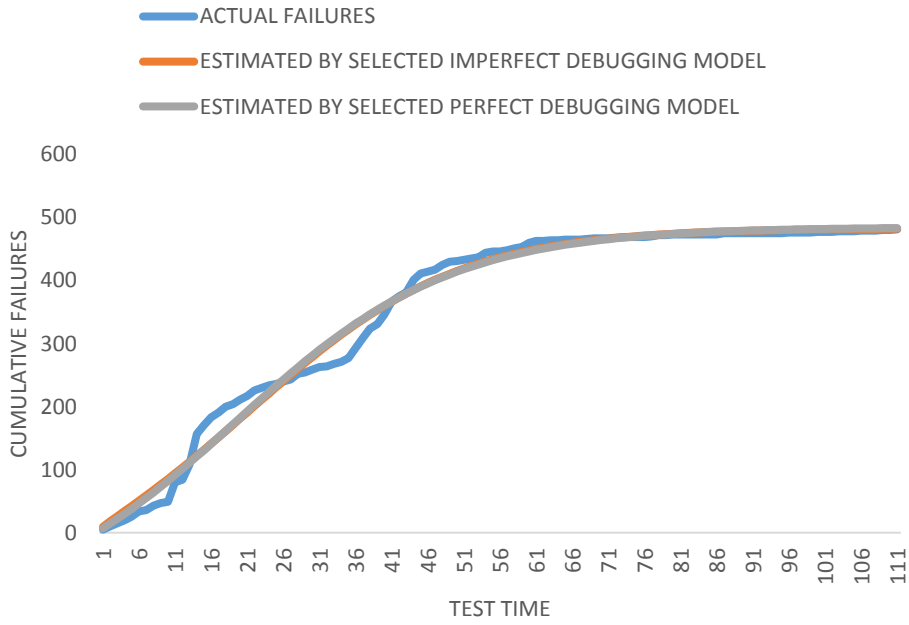

Fig 6.6: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 6

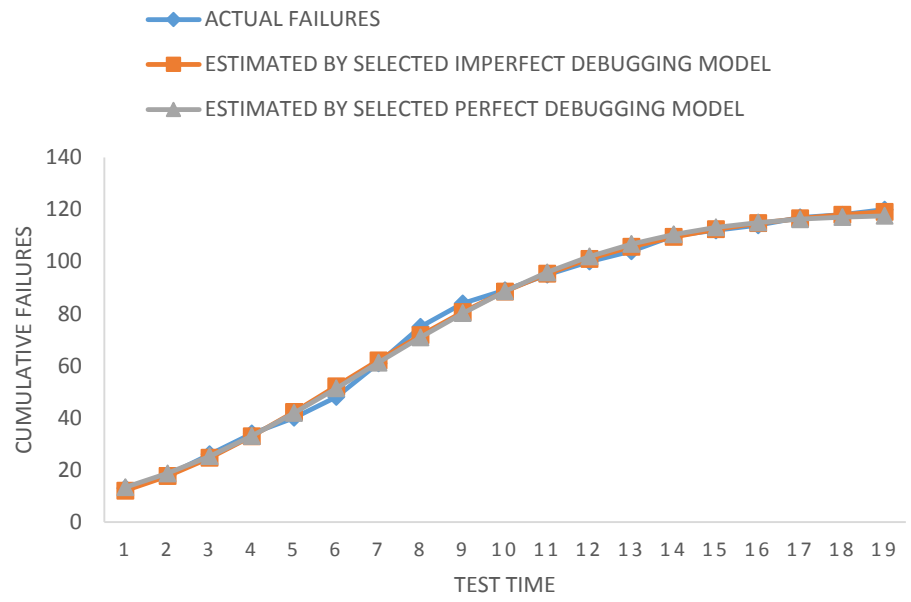

Fig 6.7: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 7

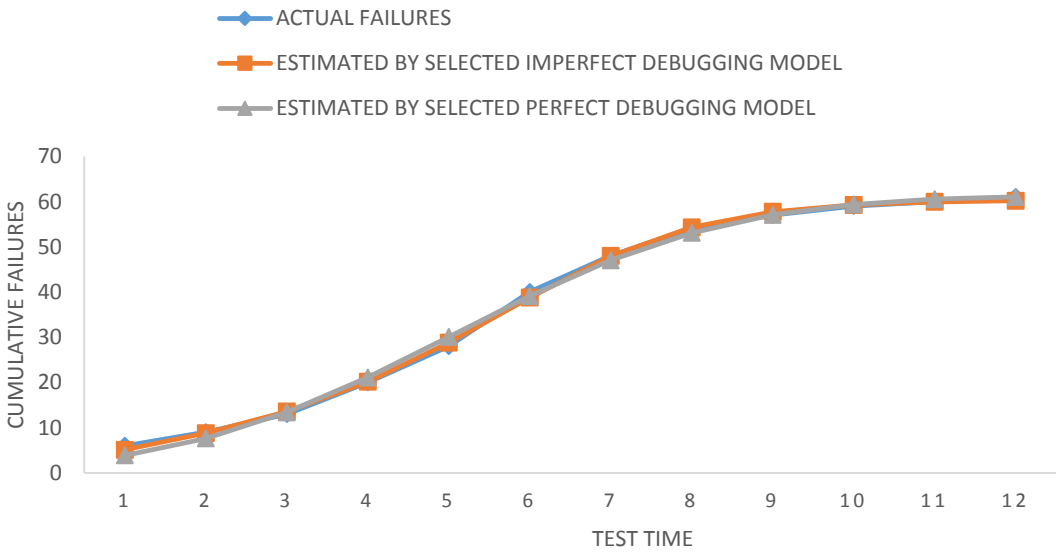

Fig 6.8: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 8

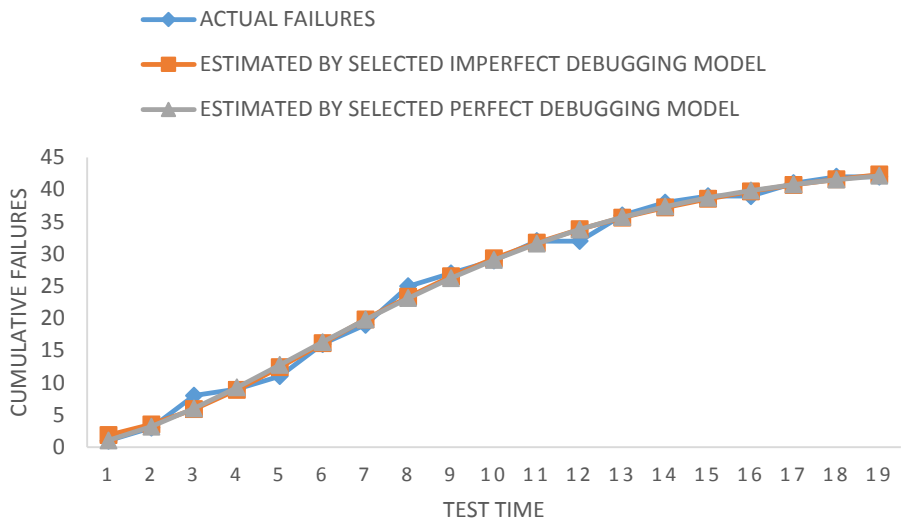

Fig 6.9: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 9

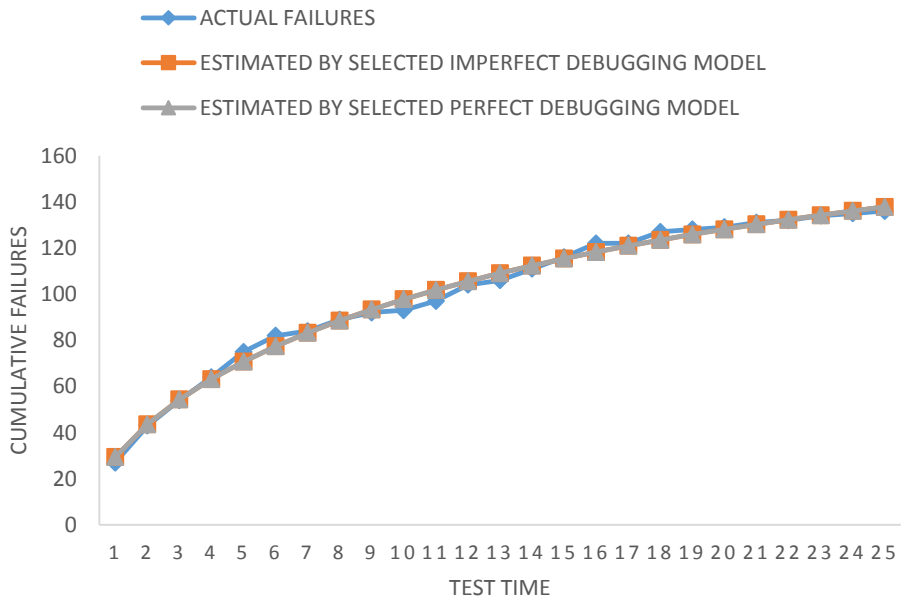

Fig 6.10: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 10

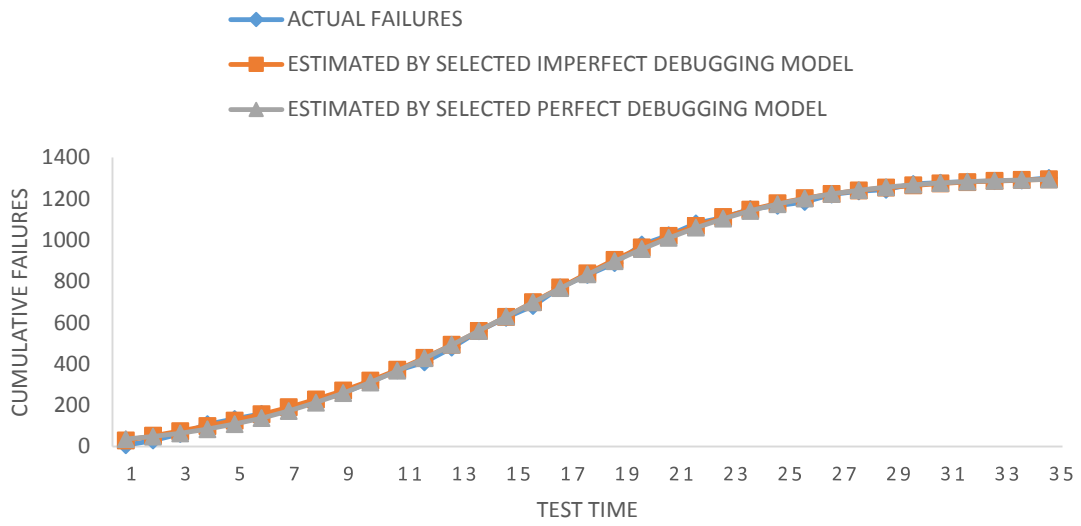

Fig 6.11: Comparison of estimated value of failures by selected perfect debugging and imperfect debugging model with actual failures for dataset 11

Table 6.1: Generalized Imperfect NHPP Software Reliability Growth Models

Model Name	$F(t)$	$G(t)$	$m(t)$
SRGM 1	$\exp(b)$	1	$\frac{A}{1-\alpha}\left[1-e^{-bp(1-\alpha)t}\right]$
SRGM 2	$\exp(b)$	1	$\frac{A}{1-\alpha}\left[1-((1+bt)e^{-bt})^{p(1-\alpha)}\right]$
SRGM 3	$\exp(b)$	1	$\frac{A}{1-\alpha}\left[1-\left(\left(\sum_{i=0}^{k-1}\frac{(bt)^i}{i!}\right)e^{-bt}\right)^{p(1-\alpha)}\right]$
SRGM 4	Probability distribution given by: $P(t) = \frac{\left(1-\left(\sum_{i=0}^{k-1}\frac{(bt)^i}{i!}\right)e^{-bt}\right)}{(1+\beta e^{-bt})}$	1	$\frac{A}{1-\alpha}\left[\left(\frac{\left(1-\left(\beta+\sum_{i=0}^{k-1}\frac{(bt)^i}{i!}\right)e^{-bt}\right)}{(1+\beta e^{-bt})}\right)^{p(1-\alpha)}\right]$
SRGM 5	$T\sim Wei(b,k)$ Weibull Distribution	1	$\frac{A}{1-\alpha}\left[1-e^{-bp(1-\alpha)t^k}\right]$
SRGM 6	$T\sim N(\mu,\sigma^2)$ Normal Distribution	1	$\frac{A}{1-\alpha}\left[1-(1-\varphi(t,\mu,\sigma))^{p(1-\alpha)}\right]$
SRGM 7	$T\sim\gamma(\alpha_1,\beta_1)$ Gamma Distribution	1	$\frac{A}{1-\alpha}\left[1-(1-\Gamma(t,\alpha_1,\beta_1))^{p(1-\alpha)}\right]$
SRGM 8	T$\sim\exp(b)$	$1(t)$	$\frac{A}{1-\alpha}\left[1-e^{-bp(1-\alpha)t}\right]$
SRGM 9	T$\sim\exp(b)$	T$\sim\exp(b)$	$\frac{A}{1-\alpha}\left[1-\left(\left(\sum_{i=0}^{k-1}\frac{(bt)^i}{i!}\right)e^{-bt}\right)^{p(1-\alpha)}\right]$
SRGM 10	T$\sim\exp(b_1)$	T$\sim\exp(b_2)$	$\frac{A}{1-\alpha}\left[1-\left(\frac{1}{b_1-b_2}(b_1e^{-b_2t}-b_2e^{-b_1t})\right)^{p(1-\alpha)}\right]$
SRGM 11	$T\sim Erlang-2$(b)	T$\sim\exp(b)$	$\frac{A}{1-\alpha}\left[1-\left(\left(1+bt+\frac{b^2t^2}{2}\right)e^{-bt}\right)^{p(1-\alpha)}\right]$
SRGM 12	T$\sim\exp(b)$	$T\sim N(\mu,\sigma^2)$	$\frac{A}{1-\alpha}\left[1-\left(1-\varphi(t,\mu,\sigma)+e^{\left(-bt+\mu b\frac{(b\sigma)^2}{2}\right)}\varphi(t,\mu+b\sigma^2,\sigma)^{p(1-\alpha)}\right)\right]$
SRGM 13	T$\sim\exp(b)$	$T\sim\gamma(\alpha_1,\beta_1)$	$\frac{A}{1-\alpha}\left[1-\left(1-\Gamma(t,\alpha_1,\beta_1)+\frac{e^{-bt}}{(1-b\beta_1)^{\alpha_1}}\Gamma\left(t,\alpha_1,\frac{\beta_1}{1-b\beta_1}\right)\right)^{p(1-\alpha)}\right]$

Note: Here SRGM 8, 9, and 11 are similar to SRGM 1, 2, and 3, so only one case is be used in present study.

Table 6.2: Perfect NHPP Software Reliability Growth Models corresponding to specific choice of $F(t)$ and $G(t)$

Model Name	$F(t)$	$G(t)$	Already Known Model Name	m(t)
SRGM 1	$\exp(b)$	1	Goel Okumoto Model [42]	$A(1-e^{-bt})$
SRGM 2	$\exp(b)$	1	Delayed S-shaped [60]	$A[1-((1+bt)e^{-bt})]$
SRGM 3	$\exp(b)$	1	3-stage Erlang Growth Curve $(if\ k=3)$ [66]	$A\left[1-\left(\left(1+bt+\frac{b^2t^2}{2}\right)e^{-bt}\right)\right]$
SRGM 4	Probability distribution given by: $P(t)=\frac{\left(1-\left(\sum_{i=0}^{k-1}\frac{(bt)^i}{i!}\right)e^{-bt}\right)}{(1+\beta e^{-bt})}$	1	Inflection S-shaped [57] $(if\ k=1)$	$A\left(\frac{1-e^{-bt}}{1+\beta e^{-bt}}\right)$
			Kapur et.al. [71] $(if\ k=2)$	$A\left(\frac{1-(1+bt)e^{-bt}}{1+\beta e^{-bt}}\right)$
SRGM 5	$T\sim Wei(b,k)$ Weibull Distribution	1	Generalized Goel [73]	$A\left[1-e^{-bt^k}\right]$
SRGM 6	$T\sim N(\mu,\sigma^2)$ Normal Distribution	1	Normal Distribution [51]	$A\left(1-\left(1-\varphi(t,\mu,\sigma)\right)\right)$
SRGM 7	$T\sim\gamma(\alpha_1,\beta_1)$ Gamma Distribution	1	Gamma Distribution [51]	$A\left[1-\left(1-\Gamma(t,\alpha_1,\beta_1)\right)\right]$
SRGM 8	T~exp(b)	$1(t)$	Goel Okumoto Model [55]	$A(1-e^{-bt})$
SRGM 9	T~exp(b)	T~exp(b)	Delayed S-shaped [60]	$A[1-((1+bt)e^{-bt})]$
SRGM 10	T~exp(b_1)	T~exp(b_2)	Exponential Growth [66]	$A\left[1-\left(\frac{1}{b_1-b_2}(b_1e^{-b_2t}-b_2e^{-b_1t})\right)\right]$
SRGM 11	$T\sim Erlang-2$(b)	T~exp(b)	3-stage Erlang Growth Curve $(if\ k=3)$ [66]	$A\left[1-\left(\left(1+bt+\frac{b^2t^2}{2}\right)e^{-bt}\right)\right]$
SRGM 12	T~exp(b)	$T\sim N(\mu,\sigma^2)$	Wu et.al. I [73]	$A\left[1-\left(1-\varphi(t,\mu,\sigma)+e^{(-bt+\mu b+\frac{(b\sigma)^2}{2}}\varphi(t,\mu,b\sigma^2,\sigma)\right)\right]$
SRGM 13	T~exp(b)	$T\sim\gamma(\alpha_1,\beta_1)$	Wu et.al. II [73]	$A\left[1-\left(1-\left(\Gamma(t,\alpha_1,\beta_1)+\frac{e^{-bt}}{(1-b\beta_1)^{\alpha_1}}\Gamma\left(t,\alpha_1,\frac{\beta_1}{b\beta_1}\right)\right)\right)\right]$

Note: Here SRGM 8, 9, and 11 are similar to SRGM 1, 2, and 3, so only one case is be used in present study.

Table 6.3 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 1 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9689	5.2974	0.6069	8
EXPONENTIAL	0.9857	3.7016	0.6677	7
GENERALIZED GOEL	0.9866	3.5762	0.6743	3
INFLECTION S SHAPED	0.9890	3.2502	0.6935	2
NORMAL DISTRIBUTION	0.9983	1.2883	1.0000	1
SRGM-1	0.9857	3.5928	0.6730	4
SRGM-12	0.9857	3.6969	0.6680	6
SRGM-13	0.9356	8.0896	0.5482	11
SRGM-3	0.9388	7.4343	0.5569	10
SRGM-4.2	0.9689	5.4510	0.6035	9
SRGM-7	0.9862	3.6362	0.6711	5

Table 6.3 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 2 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9487	14.2424	0.8451	8
EXPONENTIAL	0.9418	15.6646	0.8090	9
GENERALIZED GOEL	0.9649	12.1689	0.9145	4
INFLECTION S SHAPED	0.9724	10.7839	0.9722	2
NORMAL DISTRIBUTION	0.9753	10.2166	1.0000	1
SRGM-1	0.9243	17.3011	0.7691	10
SRGM-12	0.9271	18.1513	0.7567	11
SRGM-13	0.9590	13.6193	0.8667	7
SRGM-3	0.9564	13.1297	0.8794	6
SRGM-4.2	0.9660	11.9788	0.9217	3
SRGM-7	0.9590	13.1575	0.8799	5

Table 6.3 (c). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 3 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.8234	15.2521	0.6100	9
EXPONENTIAL	0.9388	9.2973	0.7897	6
GENERALIZED GOEL	0.9562	7.8651	0.8549	3
INFLECTION S SHAPED	0.9388	9.2972	0.7897	7
NORMAL DISTRIBUTION	0.9766	5.7477	1.0000	1
SRGM-1	0.9388	8.9820	0.8006	5
SRGM-12	0.9388	9.2972	0.7897	8
SRGM-13	0.9588	7.9142	0.8540	4
SRGM-3	0.7162	19.3383	0.5153	11
SRGM-4.2	0.8234	15.2521	0.6100	10
SRGM-7	0.9586	7.6473	0.8666	2

Table 6.3 (d). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 4 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9487	14.2424	0.8451	8
EXPONENTIAL	0.9418	15.6646	0.8090	9
GENERALIZED GOEL	0.9649	12.1689	0.9145	4
INFLECTION S SHAPED	0.9724	10.7839	0.9722	2
NORMAL DISTRIBUTION	0.9753	10.2166	1.0000	1
SRGM-1	0.9243	17.3011	0.7691	10
SRGM-12	0.9271	18.1513	0.7567	11
SRGM-13	0.9590	13.6193	0.8667	7
SRGM-3	0.9564	13.1297	0.8794	6
SRGM-4.2	0.9660	11.9788	0.9217	3
SRGM-7	0.9590	13.1575	0.8799	5

Table 6.3 (e). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 5 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9748	8.7277	0.7354	9
EXPONENTIAL	0.9943	4.2805	1.0000	1
GENERALIZED GOEL	0.9936	4.5212	0.9730	4
INFLECTION S SHAPED	0.9932	4.6501	0.9597	7
NORMAL DISTRIBUTION	0.9762	8.7230	0.7362	8
SRGM-1	0.9929	4.6382	0.9607	6
SRGM-12	0.9943	4.3887	0.9877	2
SRGM-13	0.9939	4.5347	0.9718	5
SRGM-3	0.9410	13.3506	0.6335	11
SRGM-4.2	0.9748	8.7277	0.7354	10
SRGM-7	0.9937	4.4727	0.9782	3

Table 6.3 (f). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 6 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9855	18.1490	0.9747	3
EXPONENTIAL	0.9855	18.2395	0.9723	6
GENERALIZED GOEL	0.9870	17.2818	0.9993	2
INFLECTION S SHAPED	0.9870	17.2571	1.0000	1
NORMAL DISTRIBUTION	0.9838	19.2556	0.9465	7
SRGM-1	0.9647	28.3258	0.7933	11
SRGM-12	0.9695	26.5626	0.8160	9
SRGM-13	0.9691	26.7168	0.8139	10
SRGM-3	0.9757	23.4977	0.8615	8
SRGM-4.2	0.9855	18.2329	0.9725	5
SRGM-7	0.9856	18.1574	0.9745	4

Table 6.3 (g). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 7 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9900	3.8325	0.7692	5
EXPONENTIAL	0.9902	3.9254	0.7628	6
GENERALIZED GOEL	0.9929	3.3436	0.8105	3
INFLECTION S SHAPED	0.9955	2.6696	0.8907	2
NORMAL DISTRIBUTION	0.9972	2.0906	1.0000	1
SRGM-1	0.9824	5.0992	0.6976	10
SRGM-12	0.9854	4.9354	0.7059	9
SRGM-13	0.9899	4.1095	0.7507	8
SRGM-3	0.9781	5.6854	0.6743	11
SRGM-4.2	0.9900	3.9504	0.7610	7
SRGM-7	0.9915	3.6437	0.7840	4

Table 6.3 (h). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 8 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM-1	0.9524	4.9235	0.6096	11
SRGM-12	0.9655	4.6870	0.6228	10
EXPONENTIAL	0.9811	3.2699	0.6907	9
DELAYED S SHAPED	0.9811	3.1020	0.7015	8
SRGM-7	0.9866	2.7552	0.7306	7
SRGM-13	0.9886	2.6983	0.7366	6
SRGM-3	0.9865	2.6202	0.7427	5
SRGM-4.2	0.9900	2.3757	0.7700	4
GENERALIZED GOEL	0.9908	2.2804	0.7818	3
INFLECTION S SHAPED	0.9951	1.6714	0.8878	2
NORMAL DISTRIBUTION	0.9970	1.2996	1.0000	1

Table 6.3 (i). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 9 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9947	1.0465	0.9917	2
EXPONENTIAL	0.9947	1.0819	0.9756	7
GENERALIZED GOEL	0.9952	1.0297	1.0000	1
INFLECTION S SHAPED	0.9950	1.0568	0.9870	4
NORMAL DISTRIBUTION	0.9917	1.3533	0.8787	8
SRGM-1	0.9760	2.2362	0.7206	10
SRGM-12	0.9263	4.1720	0.5888	11
SRGM-13	0.9950	1.0812	0.9761	6
SRGM-3	0.9908	1.3853	0.8694	9
SRGM-4.2	0.9950	1.0564	0.9872	3
SRGM-7	0.9949	1.0662	0.9827	5

Table 6.3 (j). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 10 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.8646	11.5859	0.5469	9
EXPONENTIAL	0.9664	5.9012	0.7059	6
GENERALIZED GOEL	0.9935	2.5908	1.0000	1
INFLECTION S SHAPED	0.9664	5.9011	0.7059	7
NORMAL DISTRIBUTION	0.9751	5.0822	0.7456	4
SRGM-1	0.9664	5.7714	0.7108	5
SRGM-12	0.9664	5.9011	0.7059	8
SRGM-13	0.9933	2.6958	0.9804	3
SRGM-3	0.7855	14.5852	0.4841	11
SRGM-4.2	0.8646	11.5859	0.5469	10
SRGM-7	0.9933	2.6338	0.9917	2

Table 6.3 (k). Estimated Values of RMSE and RSqure (RSq) Attributes for each perfect debugging SRGM for Dataset 11 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
DELAYED S SHAPED	0.9873	53.6438	0.6198	6
EXPONENTIAL	0.9873	54.4759	0.6179	7
GENERALIZED GOEL	0.9973	25.2771	0.7660	2
INFLECTION S SHAPED	0.9928	40.8890	0.6618	5
NORMAL DISTRIBUTION	0.9992	13.4967	1.0000	1
SRGM-1	0.9581	97.2867	0.5488	9
SRGM-12	0.9656	90.9546	0.5574	8
SRGM-13	0.9515	107.9975	0.5386	11
SRGM-3	0.9942	36.0919	0.6845	4
SRGM-4.2	0.9970	26.6066	0.7525	3
SRGM-7	0.9575	98.0121	0.5480	10

Table 6.4 (a). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 1 and their ranking

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM 1	0.9857	3.8107	0.6478	7
SRGM 10	0.9931	2.7382	0.7119	5
SRGM 2	0.9838	3.9380	0.6418	10
SRGM-12	0.9982	1.3169	0.9462	3
SRGM-13	0.9857	3.9357	0.6429	8
SRGM-3	0.9627	5.9742	0.5804	11
SRGM-4.1	0.9987	1.1758	1.0000	1
SRGM-4.2	0.9986	1.2395	0.9742	2
SRGM-5	0.9866	3.8072	0.6484	6
SRGM-6	0.9974	1.6213	0.8620	4
SRGM-7	0.9856	3.9494	0.6423	9

Table 6.4 (b). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 2 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9243	18.4957	0.7126	10
SRGM10	0.9265	18.9133	0.7084	11
SRGM12	0.9750	10.6230	0.9182	4
SRGM13	0.9572	15.0212	0.7855	7
SRGM2	0.9371	15.7710	0.7610	8
SRGM3	0.9558	13.2141	0.8256	6
SRGM4.1	0.9834	8.9768	1.0000	1
SRGM4.2	0.9813	9.5329	0.9698	2
SRGM5	0.9649	13.0716	0.8339	5
SRGM6	0.9752	10.5757	0.9202	3
SRGM7	0.9340	17.9256	0.7252	9

Table 6.4 (c). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 3 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9388	9.6482	0.7551	7
SRGM10	0.9799	5.7584	0.9630	2
SRGM12	0.8993	12.8770	0.6656	10
SRGM13	0.9388	10.4887	0.7329	8
SRGM2	0.9002	11.8706	0.6836	9
SRGM3	0.5992	22.2495	0.4254	11
SRGM4.1	0.9812	5.3404	1.0000	1
SRGM4.2	0.9790	5.8760	0.9533	3
SRGM5	0.9562	8.4953	0.8015	6
SRGM6	0.9766	6.2085	0.9277	4
SRGM7	0.9607	8.0468	0.8214	5

Table 6.4 (d). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 4 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9533	6.2906	0.8244	11
SRGM10	0.9664	5.6609	0.8687	10
SRGM12	0.9810	4.2582	1.0000	1
SRGM13	0.9759	5.1306	0.9124	9
SRGM2	0.9725	4.3685	0.9830	3
SRGM3	0.9756	4.3154	0.9906	2
SRGM4.1	0.9793	4.4440	0.9782	5
SRGM4.2	0.9789	4.4917	0.9729	6
SRGM5	0.9779	4.5960	0.9617	7
SRGM6	0.9794	4.4375	0.9790	4
SRGM7	0.9767	4.7176	0.9491	8

Table 6.4 (e). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 5 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9929	4.9034	0.9735	8
SRGM10	0.9891	6.0634	0.8809	9
SRGM12	0.9943	4.6810	0.9967	3
SRGM13	0.9943	4.6810	0.9967	4
SRGM2	0.9935	4.6898	0.9954	5
SRGM3	0.9681	10.3751	0.7110	11
SRGM4.1	0.9932	4.6505	0.9995	1
SRGM4.2	0.9936	4.8093	0.9831	7
SRGM5	0.9936	4.7955	0.9845	6
SRGM6	0.9802	8.4377	0.7685	10
SRGM7	0.9939	4.6659	0.9982	2

Table 6.4 (f). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 6 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9647	28.5893	0.6082	11
SRGM10	0.9756	23.8823	0.6383	9
SRGM12	0.9829	20.0965	0.6701	8
SRGM13	0.9716	25.7565	0.6254	10
SRGM2	0.9978	7.1375	1.0000	1
SRGM3	0.9848	18.7713	0.6836	6
SRGM4.1	0.9871	17.3553	0.7003	3
SRGM4.2	0.9878	16.8871	0.7063	2
SRGM5	0.9870	17.4440	0.6992	4
SRGM6	0.9855	18.3852	0.6880	5
SRGM7	0.9847	18.8691	0.6826	7

Table 6.4 (g). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 7 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9824	5.4285	0.6832	10
SRGM10	0.9904	4.1538	0.7458	8
SRGM12	0.9978	2.0732	1.0000	1
SRGM13	0.9649	7.9295	0.6142	11
SRGM2	0.9911	3.8533	0.7657	6
SRGM3	0.9876	4.4010	0.7305	9
SRGM4.1	0.9972	2.2525	0.9599	3
SRGM4.2	0.9960	2.5707	0.9024	4
SRGM5	0.9929	3.5744	0.7875	5
SRGM6	0.9975	2.1325	0.9859	2
SRGM7	0.9915	3.8953	0.7630	7

Table 6.4 (h). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 8 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9524	5.5046	0.5472	10
SRGM10	0.9570	5.5914	0.5484	9
SRGM12	0.9967	1.3657	0.7833	4
SRGM13	0.9570	6.0357	0.5434	11
SRGM2	0.9767	3.4468	0.6015	8
SRGM3	0.9863	2.6450	0.6405	6
SRGM4.1	0.9991	0.7771	1.0000	1
SRGM4.2	0.9985	0.9637	0.9030	2
SRGM5	0.9908	2.5857	0.6461	5
SRGM6	0.9970	1.3042	0.7969	3
SRGM7	0.9866	2.7573	0.6347	7

Table 6.4 (i). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 9 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9760	2.3806	0.7135	10
SRGM10	0.9814	2.1680	0.7381	9
SRGM12	0.9954	1.1255	0.9721	4
SRGM13	0.9759	2.5611	0.6977	11
SRGM2	0.9939	1.1299	0.9696	5
SRGM3	0.9944	1.1520	0.9608	6
SRGM4.1	0.9949	1.0628	0.9998	1
SRGM4.2	0.9953	1.0878	0.9885	2
SRGM5	0.9952	1.1008	0.9827	3
SRGM6	0.9938	1.2492	0.9246	8
SRGM7	0.9938	1.2487	0.9248	7

Table 6.4 (j). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 10 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9664	6.0399	0.7059	7
SRGM10	0.9894	3.4794	0.8790	5
SRGM12	0.9664	6.1891	0.7006	8
SRGM13	0.9664	6.3499	0.6952	9
SRGM2	0.8952	10.4252	0.5777	10
SRGM3	0.8360	13.0399	0.5224	11
SRGM4.1	0.9933	2.7638	0.9796	2
SRGM4.2	0.9933	2.7657	0.9793	3
SRGM5	0.9935	2.6518	1.0000	1
SRGM6	0.9765	5.1782	0.7475	6
SRGM7	0.9919	3.0342	0.9362	4

Table 6.4 (k). Estimated Values of RMSE and RSqure (RSq) Attributes for each imperfect debugging SRGM for Dataset 11 and their rankings

MODEL NAME	RSQUARE	RMSE	RANK INDEX	RANK
SRGM1	0.9581	100.3759	0.5318	11
SRGM10	0.9686	88.3001	0.5442	10
SRGM12	0.9992	13.7696	0.8827	2
SRGM13	0.9935	40.7655	0.6263	8
SRGM2	0.9847	58.8052	0.5822	9
SRGM3	0.9942	36.7250	0.6409	7
SRGM4.1	0.9996	10.5428	1.0000	1
SRGM4.2	0.9981	21.8544	0.7405	4
SRGM5	0.9973	25.6816	0.7041	5
SRGM6	0.9985	19.5350	0.7693	3
SRGM7	0.9948	35.4696	0.6462	6

Table 6.5. Relative performance of best imperfect, best perfect models and their comparison with literature.

Dataset	Type of debugging	Model selected	Relative rank	Best Model Given in literature
Dataset 1	Imperfect	SRGM 4.1	I	Inflection S Shaped [107]
	Perfect	Normal Distribution	II	
Dataset 2	Imperfect	SRGM 4.1	I	Delayed S Shaped [113]
	Perfect	Normal Distribution	II	
Dataset 3	Imperfect	SRGM 4.1	I	Yamada Exponential [113]
	Perfect	Normal Distribution	II	
Dataset 4	Imperfect	SRGM 12	I	Delayed S Shaped [113]
	Perfect	Inflection S Shaped	II	
Dataset 5	Imperfect	SRGM 4.1	II	Pham Zhang Model [27]
	Perfect	Exponential	I	
Dataset 6	Imperfect	SRGM 2	I	Pham Zhang Model [89]
	Perfect	Inflection S Shaped	II	
Dataset 7	Imperfect	SRGM 12	I	Pham Zhang Model [120]
	Perfect	Normal Distribution	II	
Dataset 8	Imperfect	SRGM 4.1	I	Pham Zhang Model [121]
	Perfect	Normal Distribution	II	
Dataset 9	Imperfect	SRGM 12	I	Pham Zhang Model [120]
	Perfect	Generalized Goel	II	
Dataset 10	Imperfect	SRGM 5	I	Goel Okumoto [89]
	Perfect	Generalized Goel	II	
Dataset 11	Imperfect	SRGM 4.1	I	SRGM 4.1 [16]
	Perfect	Normal Distribution	II	

Note: Entries in large parenthesis [] are the citations of papers from where results are referred.

CHAPTER 7

CHAPTER 7

USE OF GENETIC ALGORITHM BASED APPROACH IN SELECTING AN APPROPRIATE SOFTWARE RELIABILITY GROWTH MODEL

When a software reliability growth model is to be fitted to an observed failure data, appropriate values of its various parameters have to be determined. In this chapter a genetic algorithm based technique is proposed to estimate the values of unknown parameters of software reliability growth models (SRGMs). The proposed technique is then used to estimate values of parameters of various SRGMs which are then ranked to fit the given fault data. The proposed approach has been used to estimate values of unknown parameters of ten models taken from literature for eleven datasets listed in appendix B. The estimated values of parameters obtained using this approach are next compared with the values estimated using least squared estimation (LSE).

7.1 MOTIVATION

Least Square Estimation methods evaluates the set of unknown parameters with the highest probability of being appropriate for a given set of experimental data. In this method, least squares are taken to estimate parameters by minimizing the squared discrepancies between observed data and their expected values. Compared to ordinary techniques this technique has an advantage of saving both test time and resources [102], [106]. The least squares method obtaining parameter values by selecting their values that best fit the data. This technique is generally considered appropriate for small to medium sample sizes. In the least squared estimation basically curve-fitting [37] is used that which finds parameter values that minimize the "difference" between the data and the function fitting the data, where the difference is defined as the sum of the squared errors. For this approach the quantity to be minimized for one of the basic SRGM known as Goel Okumoto model:

$$\sum_{i=1}^{n}(\ln\left(\frac{y_i - y_{i-1}}{t_i - t_{i-1}}\right) - \ln(b) - \ln(a - y_i))^2 \qquad (7.1)$$

where

n = current number of weeks of QA test

t_i =cumulative test time at the end of ith week

y_i = cumulative number of failures at the end of the ith week

Confidence intervals are given by MUSA as

$$a \pm t_{n-2,\propto/2}\,(Var(a))^{0.5} \qquad (7.2)$$

where $t_{n-2,\propto/2}$ is the upper $\propto/2$ percentage point of the t distribution with $n-2$ degrees of freedom and $Var(a)$ is the variance of a.

The confidence interval of b is the same as above equation with b replacing a. These confidence intervals are derived by assuming that a and b are normally distributed. These confidence intervals are symmetric in contrast to the asymmetric confidence intervals provided by maximum likelihood. An alternative approach to the least squares is to directly minimize the difference between the observed number of failures and the predicted number of failures. For this approach the quantity to be minimized is:

$$\sum_{i=1}^{n}\left(y_i - m(t_i)\right)^2 \qquad (7.3)$$

here n is current number of weeks of QA test, t_i is cumulative test time at the end of ith week and y_i is actual cumulative number of failures at the end of the ith week

This technique is easy to use for any software reliability growth model since the minimization can be done by an optimization package such as curve fitting. For the Goel Okumoto model (7.3) becomes:

$$\sum_{i=1}^{n}(y_i - a(1 - e^{-bt_i}))^2 \qquad (7.4)$$

Confidence intervals for the parameters in (7.4) are the same as for classical least squares and are given by (7.2) and these confidence intervals are symmetric. However, since these are non-linear equations, the solution found may not be appropriate (a local optimum rather than a global optimum). Sometimes it may not be possible to determine a solution in a reasonable amount of time.

Genetic algorithms are a search technique that is being used in computing solutions to optimization and search problems [2]. As a search strategy, genetic algorithm has been applied successfully in many fields. Keeping this in view in this chapter we propose a genetic algorithm based approach to estimate the values of known parameters of SRGMs and select an appropriate SRGM to predict future behavior of software.

The remainder of this chapter is organized as follows. The proposed algorithm to estimate the unknown parameters and ranking of SRGMs is given in section 7.2. The proposed algorithm is next applied to eleven datasets using ten popular SRGMs. In section 7.3, the comparison of obtained software simulations results have been done with the results obtained by using least squared estimation techniques. Conclusions based on the present study are finally drawn in section 7.4.

7.2 PROPOSED METHOD

The proposed algorithm for estimating the unknown parameters of a SRGM uses an objective function which minimize the difference between estimated number of failures and actual failures similar to LSE. The objective function to be minimized is:

$$min\, J = \sqrt{\sum_{t=0}^{n}[y_i - m(t_i)]^2} \qquad (7.5)$$

where y_i is actual number of failure at time i and m(t_i) is estimated number of failures (using mean value function of corresponding models identified for proposed study.).

The proposed method works as under:

Collect experimental data from literature/industry.

Estimate the unknown parameters by minimizing objective function given in (7.5) using GA Tool of Matlab.

Rank the models according to the value of objective function (i.e. the model having minimum value of fitness function is ranked one and so on).

The proposed algorithm has been implemented using GA tool of Matlab which is a part of optimization toolbox. The following parameters have been used in proposed GA:

7.3 APPLICATION OF PROPOSED APPROACH

For the study we used ten of the widely used SRGMs [54], these along with their mean value functions are listed in table 7.2. The main reason for their selection is their wide popularity and availability of their statistical equations. The mean value functions have parameter a which refers to total number of predicted defects and b is generally the shape parameter or growth rate parameter. After collecting data, it is the time to evaluate how well the model fits the observed data. It is determined by fitting the model to the actual failure data. In our case, we have done this using GA given in section 7.2. To evaluate the performance of proposed algorithm we have applied it on ten datasets taken from literature. First using the proposed GA we have estimated the value of unknown parameters for all the ten datasets. For illustration purpose in table 7.5 the value of unknown parameters is given in table 7.3. Based on the value of unknown parameters the value of failures is computed corresponding to each model. The models are next ranked using (7.5) to select the most appropriate model. The ranking of models for all the datasets is given in table 7.4.

We have also compared the value of estimated failures using obtained rank one model with the actual failures and those estimated using LSE which is shown in

fig 7.1 to 7.10 for all the used datasets. From the results it is observed that using proposed GA technique, the estimated values of defects are more close to the actual defects as compared to LSE technique for all ten datasets.

7.4 DISCUSSION ON RESULTS AND CONCLUSIONS

We compared the accuracy of parameter estimation using proposed approach with LSE. The cumulative failures number of failures are estimated based on the value of unknown parameters estimated using proposed GA approach and LSE. The estimated value of cumulative failures is next used to fit to the actual cumulative failures. Because we used actual cumulative failures as the input data, an accurate estimation method should reproduce the values of the parameters that generate this solution. The fitting of estimated curves obtained using both approaches to actual values is given in fig 7.1 to 7.10. The comparison shows that while LSE is good estimator for fitting the data to observed failure data but proposed GA can be used as an estimator for making reliable predictions.

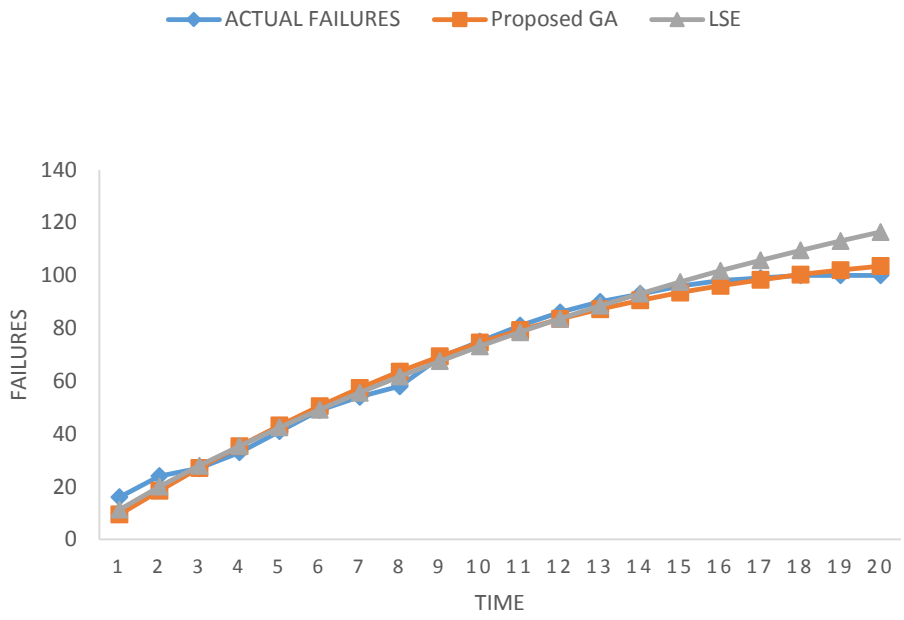

Fig 7.1: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 1

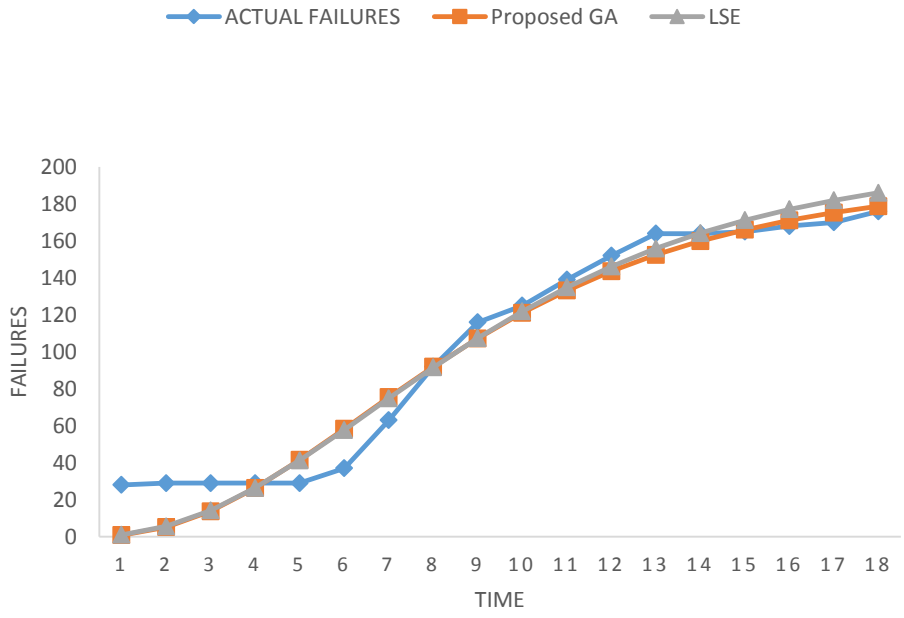

Fig 7.2: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 2

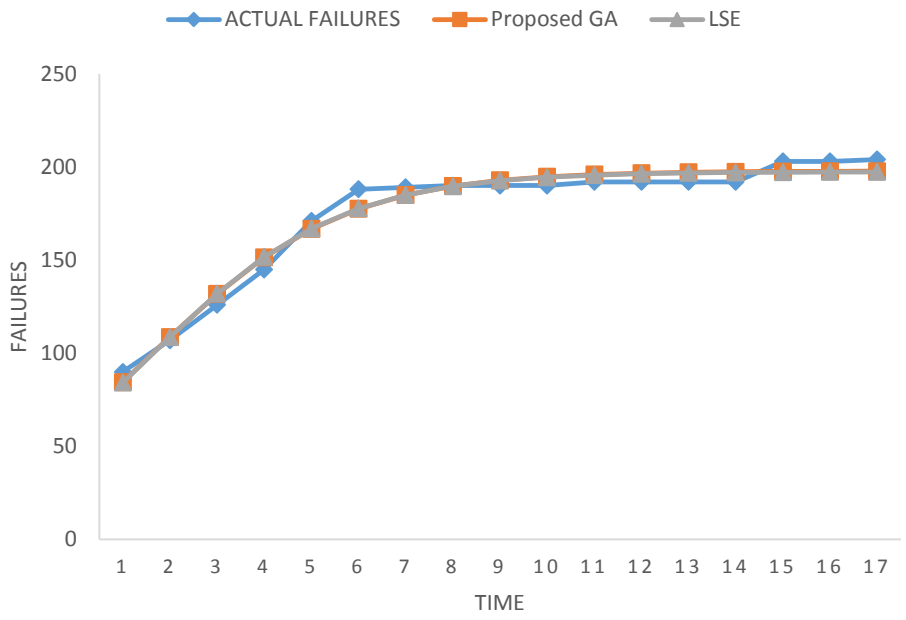

Fig 7.3: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 3

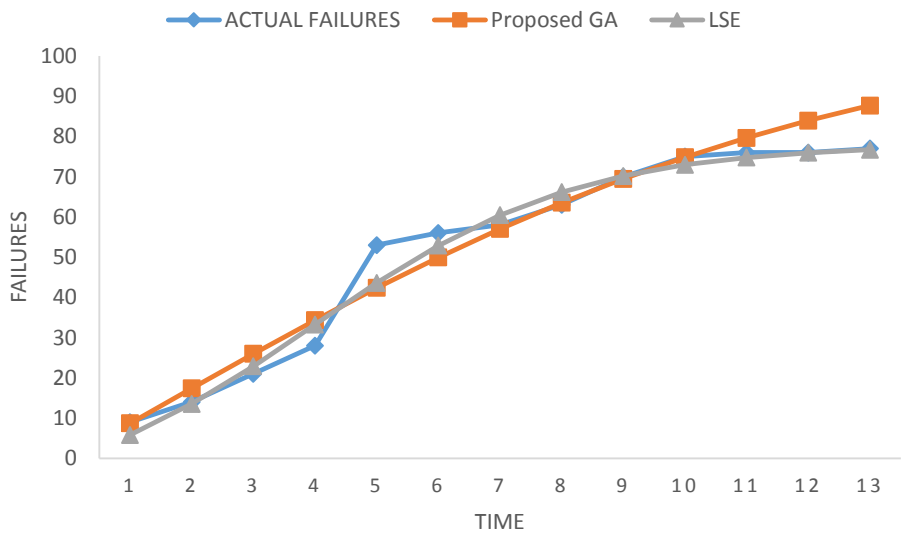

Fig 7.4: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 4

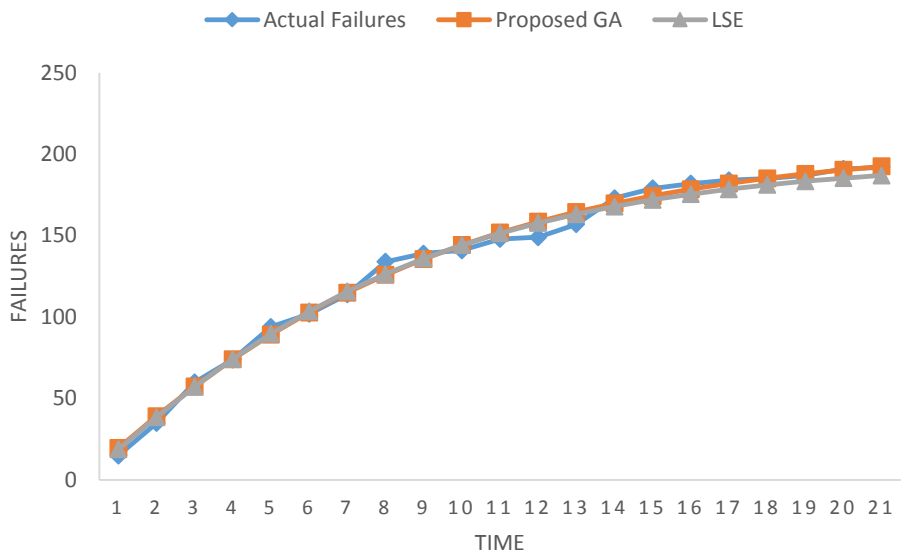

Fig 7.5: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 5

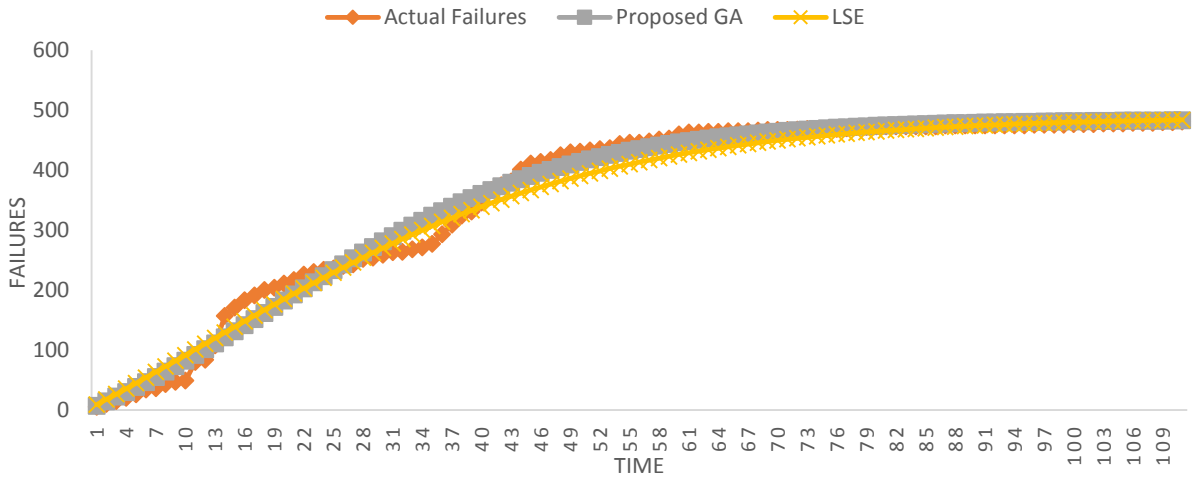

Fig 7.6: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 6

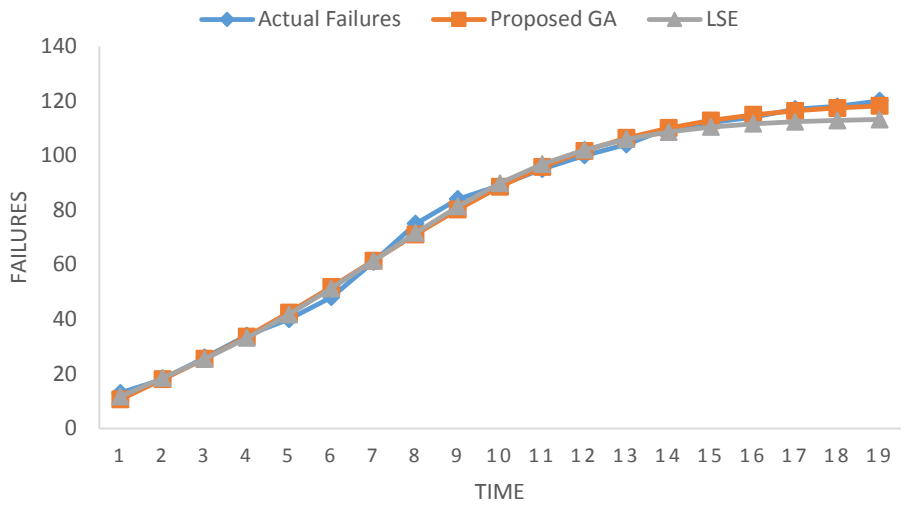

Fig 7.7: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 7

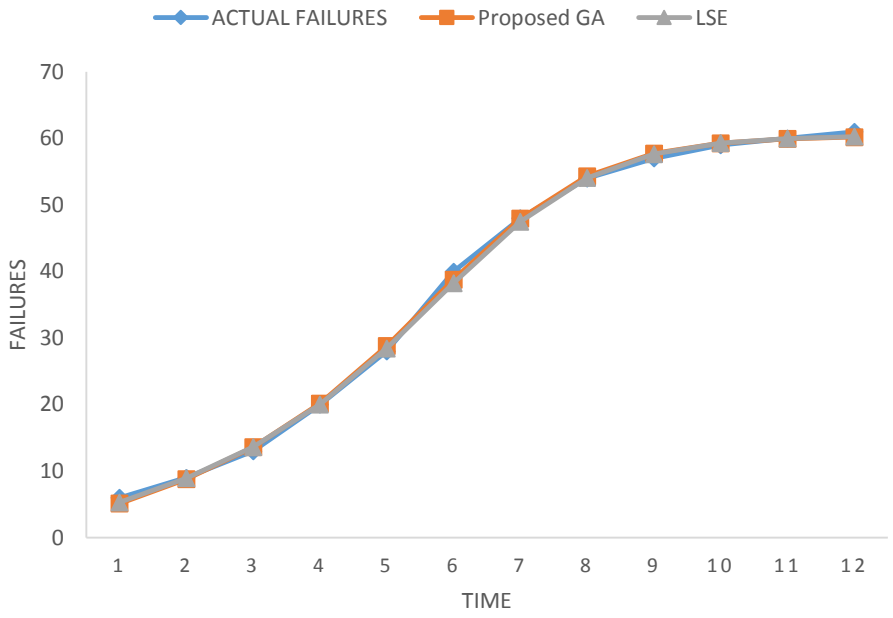

Fig 7.8: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 8

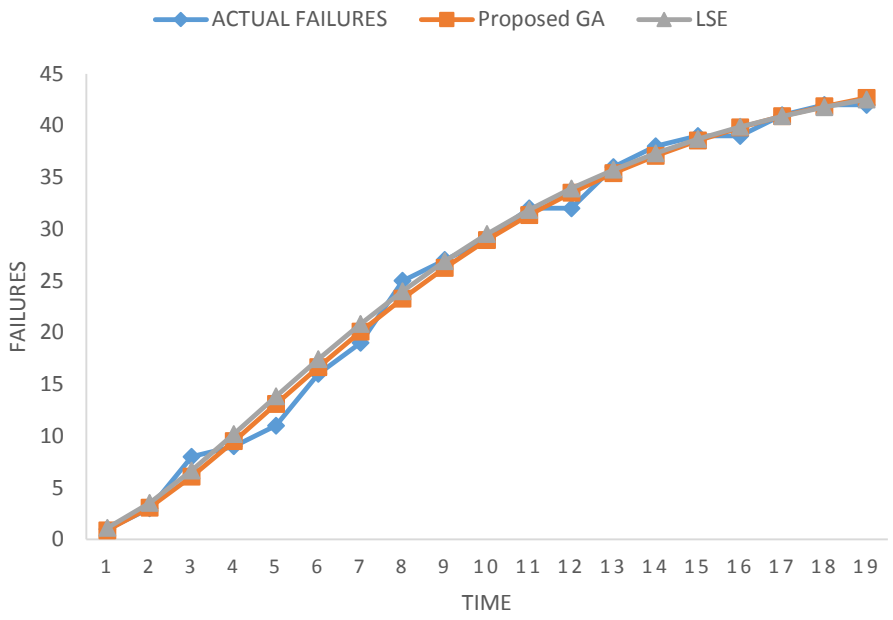

Fig 7.9: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 9

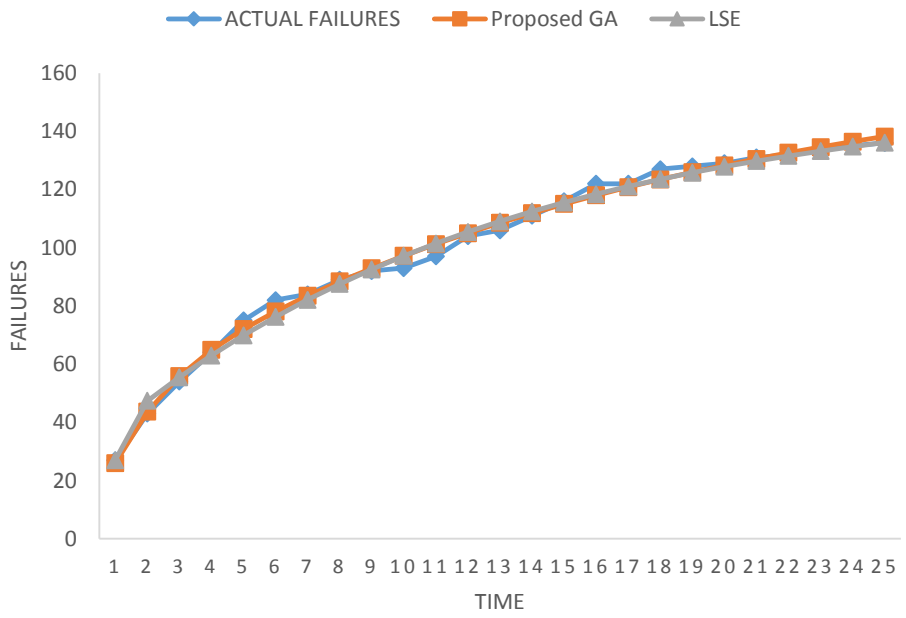

Fig 7.10: Comparison of actual number of failures with the failures estimated by proposed GA and LSE for dataset 10

Table 7.1: Parameters used in proposed GA

Parameter	Type
Population Type	Double Vector
Population Size	20
Scaling Function	Rank
Selection Function	Stochastic Uniform
Reproduction Elite Count	 2
Crossover Fraction	0.8
Crossover Function	Scattered
Migration Direction Fraction Interval	 Forward 0.2 20
Evaluate	Fitness and Constraint Function in serial

Table 7.2: Summary of SRGM used in the current study

S.No.	Model Name	m(t)
1	Goel Okumoto Model [29]	$A(1-e^{-bt})$
2	Generalized Goel [122]	$A\left[1-e^{-bt^k}\right]$
3	Delayed S-shaped [134]	$A[1-((1+bt)e^{-bt})]$
4	Inflection S-shaped [125]	$A\left(\frac{1-e^{-bt}}{1+\beta e^{-bt}}\right)$
5	Yamada Exponential [79]	$\mathrm{m(t)} = \mathrm{a}(1-\mathrm{e}^{\mathrm{r}\alpha\left(1-\mathrm{e}^{(\beta \mathrm{t})}\right)})$
6	Logistic Growth [53]	$\mathrm{m(t)} = \frac{\mathrm{a}}{1+\mathrm{ke}^{-\mathrm{bt}}}$
7	Yamada Rayleigh [126]	$m(t) = a(1-e^{-r\alpha\left(1-\mathrm{e}^{(\beta \mathrm{t}^2/2)}\right)})$
8	Zhang-Teng-Pham model (ZT Pham) [134]	$\mathrm{m(t)} = \frac{\mathrm{a}}{\mathrm{p}-\beta}[\left(1-\frac{(1+\alpha)\mathrm{e}^{-\mathrm{bt}}}{1+\alpha\mathrm{e}^{-\mathrm{bt}}}\right)^{\frac{\mathrm{c}}{\mathrm{b}}(\mathrm{p}-\beta)}]$
9	Pham Zhang model (PZ Model) [88]	$\mathrm{m(t)} = \frac{1}{(1+\beta\mathrm{e}^{-\mathrm{bt}})}((\mathrm{c}+\mathrm{a})\left(1-\mathrm{e}^{-\mathrm{bt}}\right) - \frac{\mathrm{ab}}{\mathrm{b}-\alpha}\left(\mathrm{e}^{-\alpha \mathrm{t}}-\mathrm{e}^{-\mathrm{bt}}\right))$
10	Pham Nordman Zhang (PNZ Model) [87]	$\mathrm{m(t)} = \frac{\mathrm{a}\left(1-\mathrm{e}^{-\mathrm{bt}}\right)\left(1-\frac{\alpha}{\mathrm{b}}\right)+\alpha \mathrm{at}}{1+\beta\mathrm{e}^{-\mathrm{bt}}}$

Table 7.3: Parameter Estimatied using proposed GA and LSE for dataset 5

S.No.	Model Name	Estimated value of Unknown parameters using proposed GA	Estimated value of Unknown parameters using LSE
1	Goel Okumoto Model [29]	a =217, b = 0.1074	a = 212.9, b = 0.1102
2	Generalized Goel [122]	a=207.1, b =0.09938, c =1.079	a = 196.3, b = 0.1009, c =1.119
3	Delayed S-shaped [134]	a =186.1, b = 0.3088	a = 169.9, b = 0.3572
4	Inflection S-shaped [125]	a = 207.7, b = 0.1371, β = 0.3067	a =197.6, b=0.1615, β= 0.5021
5	Yamada Exponential [79]	a=195.5, α=5.038, β=0.005195, r=5.259	a=188.5, α=9.658, β=0.001759, r=8.241
6	Logistic Growth [53]	a=188, b=0.3038, k=5.82	a=181.7, b=0.3317, k=6.174
7	Yamada Rayleigh [58]	a=197.9, α=2.549, β=0.02008, r=0.9533	a=191.6, α=2.441, β=0.02314, r=0.9708
8	Zhang-Teng-Pham Model (ZT Pham) [134]	a=153.9, α=1.214e-13, b=0.1286, β=0.1692, c=0.1975, p=0.908	a=102.4, α=1.789e-13, b=0.1262, β=0.3094, c=0.2908, p=0.7977
9	Pham Zhang model (PZ Model) [88]	a=185.5, α=0.1007, b=0.9455, β=2.147, c=33.74	a=199.6, α=3.769, b=0.1542 β=0.3224, c=0.0002312
10	Pham Nordman Zhang (PNZ Model) [87]	a=138.6, α=0.02361, b=0.2944, β=1.183	a=113.2, α=0.04198, b=0.417, β=1.899

Table 7.4: Ranking of Models for dataset 1 to 10 given in appendix B using Proposed GA

MODEL NAME	Ranks for Dataset 1	Ranks for Dataset 2	Ranks for Dataset 3	Ranks for Dataset 4	Ranks for Dataset 5	Ranks for Dataset 6	Ranks for Dataset 7	Ranks for Dataset 8	Ranks for Dataset 9	Ranks for Dataset 10
Delayed S Shaped	8	8	9	3	9	6	8	8	1	9
Generalized Goel	5	6	3	4	1	3	5	6	3	2
Goel Okumoto	6	9	5	9	2	9	7	9	9	7
Inflection S Shaped	4	3	6	1	4	1	6	3	4	8
Logistic Growth	2	2	1	2	8	7	1	2	8	5
PNZ Model	3	4	7	5	3	2	2	4	2	4
PZ Model	7	5	2	6	6	4	3	5	5	1
Yamada Exponential	10	10	4	10	7	10	9	10	10	6
Yamda Rayleigh	9	7	10	7	10	8	10	7	7	10
Zeng Teng Pham	1	1	8	8	5	5	4	1	6	3

CHAPTER 8

CHAPTER 8

USE OF SOFTWARE RELIABILITY GROWTH MODELS DURING ITS TESTING STAGE IN ESTIMATING THE RELEASE DATE OF SOFTWARE

In this chapter a method is proposed for estimating the release date of a software during its testing stage itself. The method first chooses the most appropriate reliability growth model that best fits the available detected fault at a chosen time during its testing stage and then uses it to predict the likely release date. The performance of proposed method has been tested on ten real datasets taken from literature.

8.1 MOTIVATION

Computer software is now being used in practically every field of human activity. For this appropriate softwares have to be designed, developed and tested for reliability before release. Software development is a time consuming process involving extensive costs. When software has been developed it is subjected to testing before release so that as far as possible it is bug free and therefore reliable. Software reliability is in fact one of the most important features of a developed software system.

Software developers are generally interested in estimating the likely release date of the software during its development stage itself so that future commitments regarding its delivery can be planned [56]. For this purpose usually some theoretical models (commonly known as software reliability growth models (SRGMs)) are fitted on the available test data during its testing stage to estimate its likely release date. In this chapter we propose a method to determine achieved reliability estimates of the product during its testing stage and use these to estimate the likely release date of the product. The method first uses the available test failure data to decide the appropriate SRGM model that best fits available fault data and then use it to estimate its probable release date.

A number of tools and techniques for software reliability model selection and estimation of release date have been proposed in literature [44], [55], [58], [60], [97], [109-110]. However most of these are situation specific and cannot be used in general with high confidence. In the present study we propose a simple method to choose the most appropriate SRGM and then use it to predict the probable release date so that necessary preparations and adjustments can be made, in advance.

The chapter is organized as follows. The proposed method is described in section 8.2. The proposed method is next applied on ten real datasets in section 8.3. Conclusions based on the present study are finally drawn in section 8.4.

8.2 PROPOSED METHOD

The proposed method is simple and has generally proved effective. It consists in selecting an appropriate SRGM using the method proposed in section 5.3 of chapter 5 that best fits the available fault data and then use it to predict the release date. The method consists in first estimating the parameters of likely appropriate models and then ranking them based on their performance on available data and then using the top rank SRGM to estimate the release date. The proposed method works as under.

(i) Estimation of Model Parameters and Ranking of SRGMs

The estimation of parameters using LSE and ranking of selected SRGMs is done with the method proposed in chapter 5 for competing models. The SRGM which is ranked one is next used to estimate the date of release (if rank index of more than one model are close than average of top three is considered).

(ii) Estimating the date of Release

Using the selected model we next estimate the total number of expected errors in the software. This in most of the models is the value of the parameter a itself. We also estimate the reliability of the software at time t using

$$R(t) = m(t)/a(t) \tag{8.1}$$

and the conditional reliability $(R(s|t))$ in interval $(t, t+s)$ using

$$R(s|t) = e^{-[m(t+s)-m(t)]} \qquad (8.2)$$

Conditional reliability gives an idea as to the probability that reliability achieved at time t will not change in the interval $(t, t+s)$. By specifying the reliability level to be achieved before release we can use (8.1) to estimate how much additional testing is needed in order to achieve an acceptable reliability. Next (8.2) gives an indication as to how it is expected to be in the near future. We may sum up in brief the proposed method as under.

Method in Brief

When the software being developed is ready for testing and testing carried out for some time use this information to estimate the likely length of testing period and carry on testing till about 50% of testing time is over.

Choose most appropriate models which are expected to fit available data using the ranking method of chapter 5.

Use this selected model to determine $R(t)$ and $R(s|t)$as given in (8.1) and (8.2) at intended time of release.

In case $R(t)$ ≥desired reliability level (which should normally be close to one but less than one) and the value of $R(s|t)$ for the next two time units. If these are acceptable then take this as the time of release. Else upgrade/modify testing infrastructure suitably if proposed release date is to be adhered.

It is generally advisable to update the estimates again when about 75% of estimated release time is over.

8.3 APPLICATION ON TEST DATA

In this section we demonstrate the application of the proposed method on ten datasets taken from literature and given in appendix B i.e. real time control system test data [8], testing process on a middle size project [86], three releases of a large medical record software data [113], real time control and command system data [136], tandem computers failure data of four releases [120]. Based on our experience,

in our study we have taken time of release t when at least $R(t) \geq 0.96$ and $R(s|t) \geq 0.50$ for $s = 1$ and $R(s|t) \geq 0.35$ and $s = 2$. Working on one dataset is illustrated in some detail.

Example 1- This Dataset has 120 reported failures and given in appendix B as dataset 7. Accepting the actual date of release as 19 weeks, the date of release for the software by the developers parameters have been estimated using 50% of test plan data (i.e. data upto10 weeks). The estimated values of the RSq, RMSE with their rank index and the estimated value of unknown parameters for top models having rank index greater than 0.90 are given in table 8.1. Using rank 1 model we have predicted the release date using reliability and conditional reliability as given in (8.1) and (8.2). We have predicted the release date as the time t when $R(t) \geq 0.96$ and $R(s|t) \geq 0.50$ for $s = 1$ and $R(s|t) \geq 0.35$ for $s = 2$. The predicted value of release time with reliability and conditional reliability to be achieved on this date is given in table 8.2.

In fact 95% reliability is achieved in 15 weeks. However conditional reliability for R(s | 1) and R(s | 2) respectively does not satisfy the requirement. Repeating the same procedure when testing has been done up to 15 week it was observed that the rank 1 model at this stage of testing is also Logistic Growth. The predicted release time with reliability and conditional reliability to be achieved on this date is given in table 8.3. The process was also repeated taking full data of 1 to 19 weeks to find the values of reliability and conditional reliability at actual release date. The values are tabulated in table 8.4.

From the results of dataset 1 it is concluded that the early prediction of release date was possible when 50% of failure data is available. The predicted value of release time for other nine datasets with likely to be achieved value of reliability and conditional reliability is given in table 8.5. The results given in fig 8.1 and table 8.5 shows that early prediction of reliability is possible in almost all the cases except for dataset 3 and 6. To evaluate the performance of our proposed method we have compared the predicted release date by our proposed method with the actual release date and the predicted release date by the models suggested as best models in literature for the datasets used in our study. The comparison is shown in table 8.6.

8.4 ANALYSIS OF RESULTS AND CONCLUSIONS

Our results show that when the proposed approach is applied to the test data set 7 when 50% testing time is over the predicted date of release by the proposed method is almost the same as the actual date of release. However the predicted release date based on 50% of data is within 1 to 2 weeks of actual date of release in the case of datasets 1, 2, 4, 5, 8 and 9. In the case of dataset 3 the predicted release date is however much earlier than the actual release date. Surprisingly in the case of dataset 6 when prediction is made with 50% of dataset predicted date of release is 232 days which is much later than the actual one of 111 days. However when prediction is made with about 75% of data the predicted date of release is again very close to actual date of release, it gets sharply reduced to 90 days. In fact even on using full available data, the predicted date of release is 85 days. This shows that perhaps testing has been overdone or some modifications were carried out in the software in between. We also experimented with the less than 50% of test data in all cases and observed that predictions in general was not reliable.

This chapter addresses the issue of selection of an appropriate software reliability model to predict the likely release date midway during its testing stage. It enables the user to predict the likely release date even when about 50% of estimated test time is over. We have applied the proposed approach on different stages of testing (i.e. when 50% of test plan is completed, 75% of test plan is completed and on the date of actual release. Results presented in table 8.6 and fig 8.1 show that using proposed method the predicted dates of release with models selected by us even using partial data are in general better than the corresponding results obtained for these datasets even when using the models suggested for these datasets in literature.

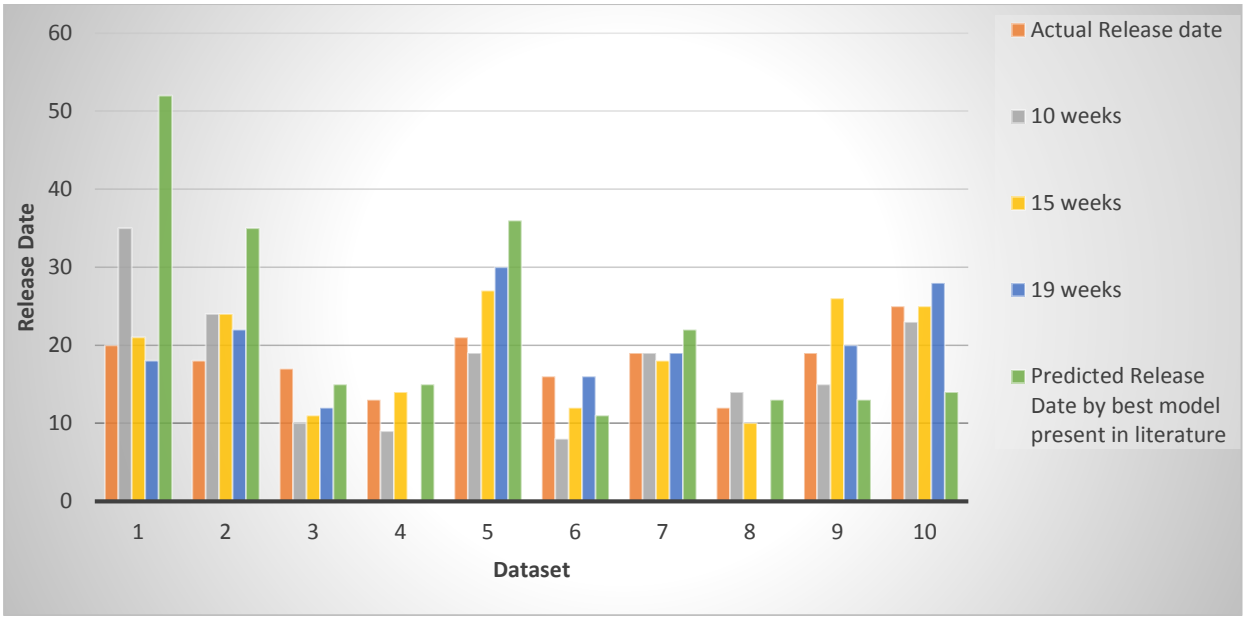

Fig 8.1: Comparison of Predicted release time for datasets used in proposed study with actual release date and by best models suggested in literature.

Table 8.1: Estimated value of RMSE, RSq, unknown parameters taking 1-10 weeks data for models having rank index >0.90

Rank	Rank Index	RMSE	R-Square	Model Name	Estimated value of unknown parameters
1.	1.0000	2.1361	0.9953	Logistic Growth	a=121.3, b=0.3529, 11.4
2.	0.9400	2.4237	0.9939	Gompertz	A=191, b=0.04283, c=0.8645

Table 8.2. Predicted release time for dataset 1using 1-10 weeks of data with likely to be achieved value of reliability and conditional reliability

Time of Prediction	Model Selected	Predicted Release Date	Reliability expected to be achieved	Conditional Reliability to be Achieved	
				For Next week	For next 2 weeks
10 weeks	Logistic Growth	19 weeks	0.99	0.61	0.43

Table 8.3. Predicted release time for dataset 1using 1-15 weeks of data with likely to be achieved value of reliability and conditional reliability

Time of Prediction	Model Selected	Predicted Release Date	Reliability expected to be achieved	Conditional Reliability to be Achieved	
				For Next week	For next 2 weeks
15 weeks	Logistic Growth	18 weeks	0.98	0.56	0.37

Table 8.4. Predicted release time for dataset 1using 1-19 weeks of data with likely to be achieved value of reliability and conditional reliability

Time of Prediction	Model Selected	Predicted Release Date	Reliability expected to be achieved	Conditional Reliability to be Achieved	
				For Next week	For next 2 weeks
19 weeks	Logistic Growth	19 weeks	0.98	0.59	0.40

Table 8.5. Predicted release time for datasets used in proposed study with likely to be achieved value of reliability and conditional reliability

Dataset and Actual Release Date	Time of Prediction	Model Selected	Predicted Release Date	Reliability expected to be achieved	Conditional Reliability to be Achieved	
					For Next week	For next 2 weeks
Dataset 1 (20 weeks)	10 weeks	Gompertz	35 weeks	0.97	0.57	0.35
	15 weeks	Logistic Growth	21 weeks	0.98	0.59	0.40
	20 weeks	ZT Pham	18 weeks	0.98	0.58	0.40
Dataset 2 (18 weeks)	10 weeks	Logistic Growth	24 weeks	0.99	0.55	0.35
	15 weeks	Logistic Growth	24 weeks	0.99	0.69	0.52
	18 weeks	Logistic Growth	22 weeks	0.99	0.61	0.43
Dataset 3 (17 weeks)	10 weeks	ZT Pham	10 weeks	1.00	0.90	0.86
	15 weeks	Logistic Growth	11 weeks	0.99	0.54	0.37
	17 weeks	Logistic Growth	12 weeks	0.99	0.64	0.48
Dataset 4 (13 weeks)	7 weeks	Logistic Growth	9 weeks	0.98	0.53	0.40
	13 weeks	Gompertz	14 weeks	0.98	0.62	0.44
Dataset 5 (21 weeks)	10 weeks	Generalized Goel	19 weeks	0.99	0.56	0.36
	15 weeks	Generalized Goel	27 weeks	0.98	0.58	0.37
	21 weeks	Generalized Goel	30 weeks	0.98	0.58	0.36
Dataset 6 (111 Days)	55 Days	Generalized Goel	232 Days	0.97	0.59	0.35
	84 Days	Generalized Goel	90 Days	0.98	0.60	0.37
	111 Days	Inflection S Shaped	85 Days	0.98	0.62	0.39
Dataset 7 (19 weeks)	10 weeks	Logistic Growth	19 weeks	0.99	0.61	0.43
	15 weeks	Logistic Growth	18 weeks	0.98	0.56	0.37
	19 weeks	Logistic Growth	19 weeks	0.98	0.59	0.40
Dataset 8 (12 weeks)	7 weeks	Logistic Growth	14 weeks	0.98	0.60	0.44
	12 weeks	ZT Pham	10 weeks	0.98	0.74	0.65
Dataset 9 (19 weeks)	10 weeks	Inflection S Shaped	15 weeks	0.98	0.75	0.62
	15 weeks	Delayed S Shaped	26 weeks	0.97	0.77	0.62
	19 weeks	Generalized Goel	20 weeks	0.96	0.69	0.52
Dataset 10 (25 Hours)	13 weeks	Generalized Goel	23 weeks	0.96	0.69	0.52
	19 weeks	Gompertz	25 weeks	0.97	0.59	0.38
	25 weeks	PZ Model	28 weeks	0.99	0.59	0.35

Table 8.6: Comparison of prediction of release date by proposed method with the prediction of models suggested in literature for used datasets

Dataset and Actual Release Date	Time of Prediction	Model Selected	Predicted Release Date	Best Model in literature	Predicted Release Date by model suggested in literature
Dataset 1 (20 weeks)	10 weeks	Gompertz	35 weeks	Inflection S Shaped [107]	52 weeks (using 1 to 11 weeks failure data)
	15 weeks	Logistic Growth	21 weeks		
	20 weeks	ZT Pham	18 weeks		
Dataset 2 (18 weeks)	10 weeks	Logistic Growth	24 weeks	Delayed S Shaped [113]	35 weeks (using full failure Data)
	15 weeks	Logistic Growth	24 weeks		
	18 weeks	Logistic Growth	22 weeks		
Dataset 3 (17 weeks)	10 weeks	ZT Pham	10 weeks	Yamada Exponential [113]	15 weeks (using full failure Data)
	15 weeks	Logistic Growth	11 weeks		
	17 weeks	Logistic Growth	12 weeks		
Dataset 4 (13 weeks)	7 weeks	Logistic Growth	9 weeks	Delayed S Shaped [113]	15 weeks (using full failure Data)
	13 weeks	Gompertz	14 weeks		
Dataset 5 (21 weeks)	10 weeks	Generalized Goel	19 weeks	PZ Model [27]	36 weeks using full failure Data)
	15 weeks	Generalized Goel	27 weeks		
	21 weeks	Generalized Goel	30 weeks		
Dataset 6 (111 Days)	55 Days	Generalized Goel	232 Days	PZ Model [8]	82 Days (using full failure Data)
	84 Days	Generalized Goel	90 Days		
	111 Days	Inflection S Shaped	85 Days		
Dataset 7 (19 weeks)	10 weeks	Logistic Growth	19 weeks	PZ Model [134]	22 weeks (using full failure Data)
	15 weeks	Logistic Growth	18 weeks		
	19 weeks	Logistic Growth	19 weeks		
Dataset 8 (12 weeks)	7 weeks	Logistic Growth	14 weeks	PZ Model [113]	13 weeks (using full failure Data)
	12 weeks	ZT Pham	10 weeks		
Dataset 9 (19 weeks)	10 weeks	Inflection S Shaped	15 weeks	PZ Model [89]	13 weeks (using 1 to 12 weeks failure data)
	15 weeks	Delayed S Shaped	26 weeks		
	19 weeks	Generalized Goel	20 weeks		
Dataset 10 (25 Hours)	13 weeks	Generalized Goel	23 weeks	Goel Okumoto [89]	14 hours (using 1 to 16 hours failure data)
	19 weeks	Gompertz	25 weeks		
	25 weeks	PZ Model	28 weeks		

CHAPTER 9

CHAPTER 9

ESTIMATING COST OF SOFTWARE UNDER DEVELOPED

In this chapter we have first tried to estimate the cost of software under development using existing generalized software cost model. Next we have also proposed a genetic algorithm based approach to estimate the software cost versus reliability. The proposed approach is used to minimize the cost of software keeping reliability as constraint, or maximizing reliability by keeping cost as constraint. The proposed approach is next used to analyze the software cost and reliability tradeoff (maximizing reliability at minimum cost).

9.1 MOTIVATION

In recent years, the cost of developing software and software failures have entailed great expenses in a system development. Therefore, it is important to determine when to stop testing, or when to release the software to the users so that the total system cost is minimized, subject to a desired reliability level and other constraints.

The quality of the software system usually depends upon the length of testing and what testing methodologies are used. Generally speaking, the longer the testing takes, the more reliable the software is expected to be. However the total cost of developing software is expected to increase. On the other hand if the testing time of the software is too short, the cost of software could be reduced but the customer risks buying unreliable software. This will also increase the cost during the operational phase, as the cost of debugging during the operational phase is much higher than that of the testing phase.

Testing is an efficient way to remove faults in software products, but testing of all possible executable paths in general program is impractical. Moreover, after reaching a certain level of software refinement, any effort to increase reliability will require an exponential increase in cost. Thus it is important to determine when to stop testing or when to release the software the software to customers, to keep the

expected total software cost subject to warranty and risk cost minimal. It is common for software companies to provide a warranty period during which they are still responsible for debugging errors.

In this chapter we have estimated the cost of same eleven datasets given in appendix B using generalized cost model proposed by Pham H. [89]. In section 9.3 we have proposed a genetic algorithm based approach to estimate the cost of software subject to reliability constraint and reliability subject to cost constraint. The implementation of proposed approach is given in section 9.4. Conclusions based on the study are drawn in section 9.5.

9.2 GENERALIZED SOFTWARE COST MODEL

We have also computed the cost for datasets used in our earlier studies. In this case we have selected a generalized software cost model based on the mean value function of NHPP software reliability growth models to estimate the cost. In addition to the costs of traditional cost models, this cost model includes the cost to perform testing, the cost of removing detected errors and the risk cost due to software failures [136]. This model can also be used to formulate realistic total software cost projects in many applications to achieve cost for desired level of reliability.

The expected system cost $E(T)$ for expected release time (T) is comprises of the setup cost to do testing, the cost incurred in removing errors during the testing phase and during the warranty period and the risk cost in releasing the software system by time T. Hence the expected total system cost $E(T)$ can be expressed as follows [92]:

$$E(T) = C_0 + C_1 T^{\propto} + C_2 m(T)\mu_y + C_3 \mu_w [m(T + T_w) - m(T)] + C_4 [1 - R(x|T)] \qquad (9.1)$$

here

C_0: setup cost for software testing
C_1: software test cost per unit time

C_2: cost of removing each error per unit time during testing

C_3: cost of removing an error per unit time during the operational phase

C_4: loss due to software failure

W: variable of time to remove an error during the warranty period in the operation phase

μ_y: expected time to remove an error during testing phase which is $E(Y)$

μ_w: expected time to remove an error during the warranty period in the operation phase, which is $E(W)$

T: software release time

T_w: period of warranty time

$\propto$: the discount rate of the testing function

$m(T)$: expected numbers of errors detected by time T

$R(x|T)$: reliability function of software by time T for a mission time x and is calculated as

$$R(x|T) = e^{-(m(T+x)-m(T)} \qquad (9.2)$$

We applied this method on eleven datasets taken from literature [41] [49] [52] [54-55] [59-60] and given in appendix B. Considering $\boldsymbol{C_0}$=\$50, $\boldsymbol{C_1}$=\$700, $\boldsymbol{C_2}$=\$60, $\boldsymbol{C_3}$=\$3600, $\boldsymbol{C_4}$=\$50000, $\boldsymbol{\mu_y}$=0.1, $\boldsymbol{\mu_w}$=0.5, $\boldsymbol{k}$=0.95, $\boldsymbol{T_w}$=20 and $\boldsymbol{x}$=1; the optimal value of $\boldsymbol{E(T)}$ at given expected release time $\boldsymbol{T}$. We have already estimated the release date of each dataset independently in chapter 5. Now using the generalized software cost model the cost and reliability using (9.1) and (9.2) are estimated. The estimated values of cost and reliability for each dataset at the time of release is given in Table 9.1.

9.3 SOFTWARE RELIABILITY AND COST ANALYSIS USING GENETIC ALGORITHM

Several techniques can be used to achieve the high reliability level while keeping the cost minimum. Reliability and cost evaluation are one part of software quality evaluation. The reliability of a software depends on reliability of its modules (i.e.

depends on its internal capabilities so there is a requirement for architecture evaluation of the module). So to estimate the reliability and cost of a software system its architecture has to be analyzed. The purpose of architecture evaluation is to identify potential failures and to verify that the quality requirements are met in the design of software. The architecture of a software application can be used to determine the reliability and cost of that application. The architecture of a software is basically represented with the help of discrete-time Markov chain (DTMC), which is used to show the interdependency of modules with probability of visits from one module to another. Then the cost and reliability of a software application can be computed using algorithms genetic algorithms. [85].

A software system consists of n subsystems or modules within a given architecture and all the components are essential to the system and their failures are statistically independent. Therefore it is assumed that there exists a relationship between the total system reliability R and its components R_i for $(i = 1, 2, \dots, n)$.

It is also assumed that the reliability of a module is directly related to its cost. Various experiments have been conducted to show that an optimal or near optimal solution to the problem of selecting the component comprising the software can be obtained with lower cost and a high reliability of software, i.e. solution can be achieved in minimum cost. For this purpose the problem is divided in three parts in our present study.

9.3.1 Software Reliability Maximization Using Cost Constraint

Software reliability allocation plays an important role during software product design phase which has close relationship between reliability and cost. Software reliability is the probability that software will provide failure free operation in a fixed environment for a fixed interval of time. How to allocate reliability to each component so that system reliability can be maximized when cost is given. The main objective is to achieve maximum reliability using cost as constraint. Considering the software reliability R and the expected software cost C as evaluation criteria the

problem of software reliability maximization using cost as constraint is formulated as:

$$\max_{R} f\,(R) \tag{9.3}$$

$$s.t.\, C \leq C_{0,}\; 0.91 \leq R_1 \leq 0.99$$

where

$$R = \prod_{i=1}^{n} R_i^{Vi} \tag{9.4}$$

Indices:

i=module index, $i = 1, 2 \ldots n$.

Parameters:

R_i=reliability of module i.

C=Overall cost of the software.

R=Overall Reliability of software.

Decision Parameters:

C_0=maximum cost allowable

The reliability of the application, denoted by R, during a single run in this case is given by (9.2). Where V_i is the number of times the application visits component i during a single run of the application. Since V, the number of visits to component i during a single run of the application is a random variable, the reliability of the application R in (9.2) is itself a random variable, and we are interested in its expected value denoted by E[R], which is given by (9.5) [117].

$$E[R] = E\prod_{i=1}^{n} E\left[R_i^{Vi}\right] \tag{9.5}$$

To find the value of $E[R]$, there is a need to obtain $E\left[R_i^{Vi}\right]$ that is the expected reliability of component i during a single run of the application. The expected value of V denoted by V_i, and the variance of V_i denoted by σ_i^2, can be obtained from

DTMC analysis.The effective expected reliability of component i, during a single run of the application is given by (9.4) [133]. Where $E[R_i^{Vi}]$ is the expected reliability of the component i during a single run of the application and it can be obtained as:

$$E[R_i^{Vi}] = R_i^{Vi} + \frac{1}{2}(R_i^{Vi})(logR_i)^2\sigma_i^2 \quad (9.6)$$

Hence from (9.5) and (9.6) the expected reliability of the application is given by:

$$E[R] = \prod_{i=1}^{n}(R_i^{Vi} + \frac{1}{2}(R_i^{Vi})(logR_i)^2\sigma_i^2) \quad (9.7)$$

(9.7) implies that the expected reliability of the application depends not only on the reliabilities of its individual modules, but also on the expected number of times the application visits the module, and the variance of the number of visits.

9.3.2 Software Cost Minimization Using Reliability Constraint

In practical engineering, software project managers are required to estimate the cost needed to complete the development of a software system taking into account the required level of reliability. The area of the optimization of reliability allocation and development cost has gained more and more attention. Various techniques had been used in the past for designing systems under dual and often conflicting constraints of maximizing reliability and minimizing cost. In this section main objective is to minimize the cost taking reliability as a constraint. The objective function for given problem is formulated as given below.

$$\min_C f\,(C) \quad (9.8)$$

$$s.t.R \geq R_0$$

where

$$C = \sum_{i=1}^{n} C_i, \quad (9.9)$$

and

$$C_i = -\sum_{i=1}^{n} \frac{\alpha_i}{\ln R_i} \qquad (9.10)$$

Indices:

i=module index, $i = 1, 2 \ldots n$.

Parameters:

C=overall cost of software.

R=overall reliability of software.

C_i =cost of module i.

α_i=compexity of the software.

β= development cost

Decision Parameters:

R_0=minimum reliability requirement of software.

Software development process consists of several phases, where estimation of development cost means different things with respect to different phases of software development life cycle. The cost of a module is associated with a number of parameters such as the design and development time, and testing time, and can be assumed to be monotonically related to the module reliability. The module cost-reliability relation can be linear, superliner, exponential, and even random while maintaining the monotonic property. The cost of the application is calculated using RCR model as given in (9.8). It can be very expensive to achieve the reliability value of 1. In some cases cost function derived from basic considerations and is usually stated in terms of the reliability.

9.3.3 Software Reliability and Cost Tradeoffs

In this section a set of optimal solutions is provided to the user by considering the constraints of reliability maximization using cost as constraint and cost minimization using reliability as constraint. The relationship between the cost and reliability of individual modules comprising the software is usually nonlinear but monotonic relation exists between them [32-33]. The optimization problem in this case is formulated as given below:

$$\min_{C} \max_{R} f(C, R) \qquad (9.11)$$

$$s.t. R \geq R_o \; and \; C \leq C_0$$

In this section an optimal solution satisfying both the constraints of cost and reliability is also found. The solution space may consist of a single solution, a set of solutions or no solution will be there for the given constraints. The problem is solved using genetic algorithm and the solutions are provided in preceding sections. In this section the use of proposed genetic algorithm for obtaining solutions of above stated problems is explained. A problem from existing literature is selected to conduct the experiment. The application consists of 10 modules. It starts execution from first module and end at the execution of 10th module. The architecture of the problem is shown in Fig 9.1 [104].

The inter-component transition probabilities are given in Table 9.2. The visits statistics for the components of the application from DTMC analysis are presented in Table 9.3. Matlab Genetic Algorithm toolbox is used in work to optimize objective functions. The various Genetic algorithm parameters used are given in Table 9.4.

9.4 IMPLEMENTATION OF PROPOSED APPROACH

In the study the attempt is to maximize the reliability of the software taking cost of the software as constraint. Here allowable cost (C_0) of the system is considered 951. The optimal solution obtained in this case is shown in Table 9.5. From the results it is found that maximum reliability that can be achieved with the cost of 951 is 0.5651.

In the second study algorithm is used to calculate the cost of application by taking the software reliability as constraint. Table 9.6 shows the result of the problem when R0 is considered 0.84. The last study focused on the issue when both the reliability and the cost of the application are taken as constraint. This is a multi-objective problem where R0 is 0.82 and C0 is 1030. There may be some situation where no solution will be available or more than one solution may be present, Table 9.7 shows a solution for the above defined constraints by taking the value of $\beta = 90$ and $\alpha = 0.3$.

9.5 ANALYSIS OF RESULTS AND CONCLUSIONS

In this chapter a genetic algorithm based approach is proposed to calculate optimal solution for the single objective and multi-objective problems of maximizing reliability of the software while minimizing the overall cost. The sensitivity of the performance and reliability predictions of individual components have been also analyzed. The experiments conducted with the proposed approach show that an optimal or near optimal solution to the problem of optimal cost at a desired reliability level of software can be achieved using the proposed approach.

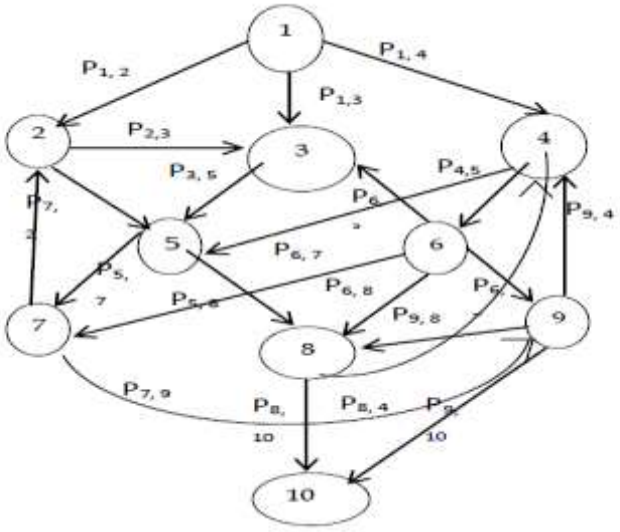

Fig 9.1. Software Application with architecture described by absorbing DTMC

Table 9.1: Estimated value of optimal cost and reliability at estimated time of release

Dataset	Model Selected	Estimated value of unknown parameters	Time of Release (T)	Estimated Reliability ($R(1|T)$)	Optimal Cost (in $) ($E(T)$)
Dataset 1	Zeng Teng Phan	$a = 2.661, \propto = 0.9921, b = 0.156, \beta = 0.947, c = 6.421, p = 0.9706$	26 weeks	0.5886	43128
Dataset 2	Logistic Growth	$a = 184.2, b = 0.3353, k = 15.2$	22 weeks	0.6065	37145
Dataset 3	Logistic Growth	$a = 197.8, b = 0.4931, k = 2.192$	12 weeks	0.6370	28855
Dataset 4	Gompertz	$a = 79.76, b = 0.0228, c = 0.6868$	14 weeks	0.6171	30959
Dataset 5	Generalized Goel	$a = 207.1, b = 0.09938, c = 1.079$	30 weeks	0.5769	47246
Dataset 6	Inflection S Shaped	$a = 484.5, b = 0.06686, \beta = 3{,}652$	85 days	0.6161	79954
Dataset 7	Logistic Growth	$a = 119.7, b = 0.3425, k = 10.53$	19 weeks	0.5880	36136
Dataset 8	Zeng Teng Pham	$a = 1.639, \propto = 0.9999, b = 0.3617, \beta = 0.05981, c = 33.45, p = 0.08445$	14 weeks	0.5655	34152
Dataset 9	Generalized Goel	$a = 44.1, b = 0.02378, c = 1.656$	20 weeks	0.6922	30485
Dataset 10	Pham Zhang Model	$a = 115.9, \propto = 0.05841, b = 0.7412, \beta = 0.239, c = 51.53$	40 hours	0.5015	61909
Dataset 11	Logistic Growth	$a = 1314, b = 0.2219, k = 30.73$	43 weeks	0.5629	59728

Table 9.2: Inter-Component Transition Probabilities for fig 9.1

$P_{1,2} = 0.60$	$P_{1,3} = 0.20$	$P_{1,4} = 0.20$	
$P_{2,3} = 0.70$	$P_{2,5} = 0.30$		
$P_{3,5} = 1.00$			
$P_{4,5} = 0.40$	$P_{4,6} = 0.60$		
$P_{5,7} = 0.40$	$P_{5,8} = 0.60$		
$P_{6,3} = 0.30$	$P_{6,7} = 0.30$	$P_{6,8} = 0.10$	$P_{6,9} = 0.30$
$P_{7,2} = 0.50$	$P_{7,9} = 0.50$		
$P_{8,4} = 0.25$	$P_{8,10} = 0.75$		
$P_{9,8} = 0.10$	$P_{9,10} = 0.60$		

Table 9.3: Visit statistics for the modules of the application

Component	Expected Number	Variance
1	1.000	0.0000
2	0.9077	0.6444
3	0.9107	0.5499
4	0.4184	0.3928
5	1.3504	0.7185
6	0.2510	0.2319
7	0.6155	0.6261
8	0.8737	0.4225
9	0.3831	0.2462
10	1.0000	0.0000

Table 9.4: Parameters used in proposed GA

Parameter	Type
Population Type	Double Vector
Population Size	20
Scaling Function	Rank
Selection Function	Stochastic Uniform
Reproduction 1. Elite Count	 2
Crossover Fraction	0.8
Crossover Function	Scattered
Migration 1. Direction 2. Fraction 3. Interval	 Forward 0.2 20
Evaluate	Fitness and Constraint Function in serial

Table 9.5: An optimal design generated by proposed GA when cost constraint is 951

Component	Relaibility	Cost
1	0.9560	96.66
2	0.9120	93.25
3	0.9810	105.63
4	0.9310	94.19
5	0.9100	93.18
6	0.9100	93.18
7	0.9100	93.18
8	0.9380	94.68
9	0.9100	93.18
10	0.9100	93.18
Overall	**0.5651**	**950.31**

Table 9.6. An optimal design generated by proposed GA when reliability constraint is 0.84

Component	Relaibility	Cost
1	0.9860	111.27
2	0.9740	101.38
3	0.9750	101.84
4	0.9860	111.27
5	0.9860	111.27
6	0.9420	95.02
7	0.9830	107.49
8	0.9800	104.84
9	0.9730	100.96
10	0.9770	102.89
Overall	**0.8490**	**1048.23**

Table 9.7: An optimal design generated by proposed GA for minimizing cost and maximizing reliability

Component	Reliability	Cost
1	0.9729	100.91
2	0.9587	97.11
3	0.9833	107.81
4	0.9704	99.98
5	0.9878	114.43
6	0.9510	95.97
7	0.9560	96.66
8	0.9734	101.12
9	0.9738	101.29
10	0.9863	111.74
Overall	**0.82019**	**1027.02**

CHAPTER 10

CHAPTER 10

CONCLUDING OBSERVATIONS

In this chapter we first critically review in brief the work presented in the earlier chapters of this thesis and then make some suggestions for future work in this direction

The prime objective of the thesis has been to investigate certain multiprocessor scheduling problems and design suitable algorithms for obtaining their optimal solution. We have also tried to investigate the problems of software development process to analyze the effectiveness of some existing software reliability and software cost optimization models. Effort has also been made to suggest methods for selecting an appropriate software reliability growth model that can be used to predict the release date, reliability achieved at time of release and cost estimates.

With this view we have proposed an algorithm for finding optimal schedule for multiprocessor task scheduling problems that take into account the execution time of tasks on multiple processors as well as the communication cost between the tasks in chapter 2. The proposed algorithm is based on genetic algorithm approach and able to handle the problems of encoding, crossover and mutation efficiently witout violating the precedence costarints between tasks. In numerical experimentation we have used several task graphs for the multiprocessor scheduling problem, and compared the results obtained with the proposed algorithm with the results available in literature using alternative approaches. It is conclude that the approach is able to find minimum makespan for a given task graph but the analysis shows that it may need to be improved further in terms of load balancing on each processor.

The proposed algorithm has been modified further to improve its performance in terms of load balancing. The two fitness functions have been applied one after the other. The first fitness function is concerned with minimizing the total

execution time (makespan), and the second one is concerned with the maximizing the load balance on each processor. The algorithm is tested on a set of total 12 problems taken from literature as well as on three benchmark problems. It is concluded from the simulations that the proposed algorithm works better than other algorithms. We have also proposed algorithm to decide optimal number of processor and on the basis of this study we noticed that 2 to 3 processors are in general sufficient for small size scheduling problems of 9 tasks on a homogeneous multiprocessor system and 4 processors will be sufficient for medium problems up to 40 tasks. Not more than 8 processors are necessary for problems of size up to 120.

A method for selecting the most appropriate reliability growth model is also proposed in the present study. The performance of the proposed method has been compared with some of the earlier methods proposed in literature using ten real datasets given in appendix B for this purpose and it is concluded that this approach generally works better than others. The approach proposed for selection of appropriate SRGM is also applied to generalized software reliability growth models to investigate its effectiveness to compare the relative performance imperfect debugging and perfect debugging models. The method is next used to estimate the release date of software. Results obtained show that proposed method can be used to estimate the release date even with partial data.

A technique based on genetic algorithm has also been proposed to estimate the values of unknown parameters of some software reliability growth models. The approach is used to estimate values of unknown parameters of different software reliability growth models taken from literature and the estimated values are next compared with the values estimated using least squared estimation technique. It has been observed that whereas LSE is a good estimator for fitting the data to observed failure data, proposed GA can also be used as an estimator for making predictions.

To conclude whereas some success has been achieved in meeting the set goals. Further work still needs to be done. Some of the possible directions in which further work can proceed are:

- Test the performance of the algorithms proposed in this thesis with suitable modifications if necessary to multicore processor problems.
- Possibility of further improvement in parallelizing the load on each processor to increase the processor utilization and reduce the schedule length.
- Design if possible some more effective approaches for predicting the likely release date of software and improving further cost estimates.

REFERENCES

REFERENCES

[1] Ahmad, I. and Kwok, Y., K.. On parallelizing the multiprocessor scheduling problems. IEEE Transactions on Parallel and Distributed Systems, 10(4): 414-431 (1999).

[2] Altaf, I., Rashid, F., Dar, J.A. and Rafiq, M., 2015, October. Survey on parameter estimation in software reliability. In Soft Computing Techniques and Implementations (ICSCTI), 2015 International Conference on (pp. 22-27). IEEE.

[3] Andersson C. A replicated empirical study of a selection method for software reliability growth models. Empirical Software Engineering 2007; 12(2): 161-182.

[4] Arora, R.K. and Rana, S.P., 1980. Heuristic algorithms for process assignment in distributed computing systems. Information Processing Letters, 11(4), pp.199-203.

[5] Bajaj, R., Agrawal, P., D.: Improving scheduling of tasks in a heterogeneous environment. IEEE Transactions on Parallel and Distributed Systems, 15(2): 107-118 (2004).

[6] Bansal, J.C., Sharma, H., Arya, K.V. and Nagar, A., 2013. Memetic search in artificial bee colony algorithm. Soft Computing, 17(10), pp.1911-1928.

[7] Baskiyar, S. and SaiRanga, P.C., 2003, October. Scheduling directed a-cyclic task graphs on heterogeneous network of workstations to minimize schedule length. In Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on (pp. 97-103). IEEE.

[8] Bauer E, Zhang X, Kimber D A. Practical system reliability. John Wiley & Sons, 2009

[9] Bector, C.R., Gupta, Y.P. and Gupta, M.C., 1989. V-shape property of optimal sequence of jobs about a common due date on a single machine. Computers & operations research, 16(6), pp.583-588.

[10] Boehm, B., Valerdi, R., Lane, J. and Brown, A.W., 2005. COCOMO suite methodology and evolution. CrossTalk, 18(4), pp.20-25.

[11] Bohler, M., Moore, F., W., Pan, Y.: Improved Multiprocessor Task Scheduling Using Genetic Algorithms. FLAIRS Conference (1999).

[12] Boregowda, U., Chakravarthy, V., R.: A Hybrid Task Scheduler for DAG Applications on a Cluster of Processors. IEEE Fourth International Conference on Advances in Computing and Communications (ICACC) (2014).

[13] Braun, T. D., Siegel, H. J., Beck, N., Boloni, L. L., Maheswaran, M., Reuther, A. I., .Freund, R. F. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed computing, 61(6), 810-837 (2001).

[14] Brest, J., Zumer, V. A performance evaluation of list scheduling heuristics for task graphs without communication costs. IEEE International Workshops on Parallel Processing (2000).

[15] Brooks, S.P. and Morgan, B.J., 1995. Optimization using simulated annealing. The Statistician, pp.241-257.

[16] Brooks, W.D. and Motley, R.W., 1980. Analysis of Discrete Software Reliability Models. IBM FEDERAL SYSTEMS DIV GAITHERSBURG MD.

[17] Casavant, T.L. and Kuhl, J.G., 1988. A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on software engineering, 14(2), pp.141-154.

[18] Chitra, P., Venkatesh, P., Rajaram, R.: Comparison of evolutionary computation algorithms for solving bi-objective task scheduling problem on heterogeneous distributed computing systems. Sadhana, 36(2): 167-180 (2011).

[19] Daoud, M., I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. Journal of Parallel and distributed computing, 68(4): 399-409 (2008).

[20] De Jong, K.A. and Spears, W.M., 1989, June. Using Genetic Algorithms to Solve NP-Complete Problems. In ICGA (pp. 124-132).

[21] De Jong, K.A. and Spears, W.M., 1989, June. Using Genetic Algorithms to Solve NP-Complete Problems. In ICGA (pp. 124-132).

[22] Deep, Kusum, Krishna Pratap Singh, Mitthan Lal Kansal, and C. Mohan. "A real coded genetic algorithm for solving integer and mixed integer optimization problems." Applied Mathematics and Computation 212, no. 2 (2009): 505-518.

[23] Du, J., Yu, C., Sun, J., Sun, C., Tang, S., Yin, Y.: EasyHPS: A Multilevel Hybrid Parallel System for Dynamic Programming. In: IEEE 27th International

Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW) (2013).

[24] F. Ercal, "Heuristic Approaches to Task Allocation for Parallel Computing", Ph.D. Thesis, Ohio State University, 1988.

[25] Forbes, J. and Long, E.A., 2008. Dertermining how much to invest in reliability. International Test and Evaluation Association, 29(3), pp.251-253.

[26] Garai, G., Chaudhurii, B., B.: A novel hybrid genetic algorithm with Tabu search for optimizing multi-dimensional functions and point pattern recognition. Information Sciences, 221: 28-48 (2013).

[27] Garg R, Sharma K, Kumar R, Garg R K. Performance analysis of software reliability models using matrix method. World Academy of Science, Engineering and Technology 2010; 71: 31-38.

[28] Garg, K. and Singh, R., 2012. Scheduling Algorithms in Mobile Ad Hoc Networks. International Journal of Computer Science & Applications (TIJCSA), 1(5).

[29] Goel A L, Okumoto K. Time-dependent error-detection rate model for software reliability and other performance measures. IEEE Transactions on Reliability 1979; 28(3): 206-211.

[30] Goel, A.L., 1985. Software reliability models: Assumptions, limitations, and applicability. IEEE Transactions on software engineering, (12), pp.1411-1423.

[31] Goh, C. K., Teoh E. J., Tan K. C.: A hybrid evolutionary approach for heterogeneous multiprocessor scheduling. Soft Computing, 13(8-9): 833-846 (2009).

[32] Gokhale, S.S. and Trivedi, K.S., 2002. Reliability prediction and sensitivity analysis based on software architecture. In Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on (pp. 64-75). IEEE.

[33] Gokhale, S.S., 2004, January. Cost constrained reliability maximization of software systems. In Reliability and Maintainability, 2004 Annual Symposium-RAMS (pp. 195-200). IEEE.

[34] Goldberg, D.E., 1989. Genetic algorithms in search, optimization, and machine learning. Addion wesley, 1989, p.102.

[35] Grefenstette, J., J.: Genetic algorithms and their applications. In: Proceedings of the Second International Conference on Genetic Algorithms Psychology Press (2013).

[36] Guan, H., Chen, W.R., Huang, N. and Yang, H.J., 2010. Estimation of reliability and cost relationship for architecture-based software. International Journal of Automation and Computing, 7(4), pp.603-610.

[37] Harter, W.L., 1974. The method of least squares and some alternatives: Part I. International Statistical Review/Revue Internationale de Statistique, pp.147-174.

[38] Helander, M.E., Zhao, M. and Ohlsson, N., 1998. Planning models for software reliability and cost. IEEE Transactions on Software Engineering,24(6), pp.420-434.

[39] Hossain S A, Dahiya R C. Estimating the parameters of a non-homogeneous Poisson-process model for software reliability. IEEE Transactions on Reliability 1993; 42(4): 604-612.

[40] Hou, E.S., Ansari, N. and Ren, H., 1994. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed systems, 5(2), pp.113-120.

[41] Hryniewicz, O., 2006. An evaluation of the reliability of complex systems using shadowed sets and fuzzy lifetime data. International Journal of Automation and Computing, 3(2), pp.145-150.

[42] http://www.Kasahara.Elec.Waseda.ac.jp/schedule/.

[43] Huang, C.Y. and Lyu, M.R., 2005. Optimal release time for software systems considering cost, testing-effort, and test efficiency. IEEE Transactions on Reliability, 54(4), pp.583-591.

[44] Huang, C.Y. and Lyu, M.R., 2011. Estimation and analysis of some generalized multiple change-point software reliability models. IEEE Transactions on Reliability, 60(2), pp.498-514.

[45] Huang, C.Y., Lo, J.H., Kuo, S.Y. and Lyu, M.R., 1999. Software reliability modeling and cost estimation incorporating testing-effort and efficiency. InSoftware Reliability Engineering, 1999. Proceedings. 10th International Symposium on (pp. 62-72). IEEE.

[46] Huang, C.Y., Lo, J.H., Kuo, S.Y. and Lyu, M.R., 2004, March. Optimal allocation of testing-resource considering cost, reliability, and testing-effort.

In Dependable Computing, 2004. Proceedings. 10th IEEE Pacific Rim International Symposium on (pp. 103-112). IEEE.

[47] Hwang, R., Mitsuo, G., Hiroshi, K.: A comparison of multiprocessor task scheduling algorithms with communication costs. Computers and Operations Research, 35(3): 976-993 (2008).

[48] Ilavarasan, E., Thambidurai, P., Mahilmannan, R.: Performance effective task scheduling algorithm for heterogeneous computing system. IEEE 4th International Symposium on Parallel and Distributed Computing (2005).

[49] Jain, D. and Jain, S.C., 2015, March. Load balancing real-time periodic task scheduling algorithm for multiprocessor environment. In Circuit, Power and Computing Technologies (ICCPCT), 2015 International Conference on (pp. 1-5). IEEE.

[50] Jelodar, M.S., Fakhraie, S.N., Montazeri, F., Fakhraie, S.M. and Ahmadabadi, M.N., 2006, July. A representation for genetic-algorithm-based multiprocessor task scheduling. In 2006 IEEE International Conference on Evolutionary Computation (pp. 340-347). IEEE.

[51] Jiuping X, Meng Z, Lei X. Integrated system of health management-oriented reliability prediction for a spacecraft software system with an adaptive genetic algorithm support vector machine. Eksploatacja i Niezawodność 2014; 16(4).

[52] John, H., 1992. Holland, Adaptation in natural and artificial systems.

[53] Kapur P K, Pham H, Anand S, Yadav K. A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Transactions on Reliability 2011; 60(1): 331-340.

[54] Kapur, P.K. and Garg, R.B., 1990. Optimal sofware release policies for software reliability growth models under imperfect debugging. Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, 24(3), pp.295-305.

[55] Kapur, P.K. and Younes, S., 1996. Modelling an imperfect debugging phenomenon in software reliability. Microelectronics reliability, 36(5), pp.645-650.

[56] Kapur, P.K., Gupta, D., Gupta, A. and Jha, P.C., 2008. Effect of introduction of faults and imperfect debugging on release time. Ratio Mathematica, 18, pp.62-90.

[57] Kapur, P.K., Kumar, D., Gupta, A. and Jha, P.C., 2006. On how to model software reliability growth in the presence of imperfect debugging and error generation. In Proceedings of 2nd International Conference on Reliability and safety Engineering (pp. 515-523).

[58] Kapur, P.K., Pham, H., Aggarwal, A.G. and Kaur, G., 2012. Two dimensional multi-release software reliability modeling and optimal release planning. IEEE Transactions on Reliability, 61(3), pp.758-768.

[59] Kapur, P.K., Pham, H., Anand, S. and Yadav, K., 2011. A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation. IEEE Transactions on Reliability, 60(1), pp.331-340.

[60] Kapur, P.K., Pham, H., Singh, J.N. and Sachdeva, N., 2014. When to stop testing multi upgradations of software based on cost criteria. International Journal of Systems Science: Operations & Logistics, 1(2), pp.84-93.

[61] Keesari, H. S., Rao, R., V.: Optimization of job shop scheduling problems using teaching-learning-based optimization algorithm. OPSEARCH 51(4): 545-561 (2014).

[62] Keiller, P.A. and Miller, D.R., 1991. On the use and the performance of software reliability growth models. Reliability Engineering & System Safety,32(1-2), pp.95-117.

[63] Kleinrock, L., Huang, J., H.: On parallel processing systems: Amdahl's law generalized and some results on optimal design. IEEE Transactions on Software Engineering, 18(5): 434-447 (1992).

[64] Kumar, S., Dave, M. and Dahiya, S., 2014. ACO based QoS aware routing for wireless sensor networks with heterogeneous nodes. In Emerging Trends in Computing and Communication (pp. 157-168). Springer India.

[65] Kumar, S., Maulik, U., Bandyopadhyay, S. and Das, S.K., 2001. Efficient Task Mapping on Distributed Heterogeneous System for Mesh Applications. In International workshop on Distributed Computing (IWDC 2001).

[66] Kumar, V. and Gupta, A., 1991, June. Analysis of scalability of parallel algorithms and architectures: a survey. In Proceedings of the 5th international conference on Supercomputing (pp. 396-405). ACM.

[67] Kuo, S.Y., Huang, C.Y. and Lyu, M.R., 2001. Framework for modeling software reliability, using various testing-efforts and fault-detection rates.IEEE Transactions on Reliability, 50(3), pp.310-320.

[68] Kwiatuszewska-Sarnecka, B., 2006. Reliability improvement of large multi-state series-parallel systems. International Journal of Automation and Computing, 3(2), pp.157-164.

[69] Kwok, Y., K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. Journal of Parallel and Distributed Computing 59(3): 381-422 (1999).

[70] Kwok, Y., K.: Parallel program execution on a heterogeneous PC cluster using task duplication. IEEE 9th Workshop on Heterogeneous Computing (2000).

[71] Lihua, X.Y., 1996. The First Fit Algorithm for Distributing Dependent Tasks in Multiprocessors System. Journal of Qingdao University Engineering & Technology Edition, 3.

[72] Liu, C., H., Li, C., F., Lai, K., C., Wu, C., C.: Dynamic critical path duplication task scheduling algorithm for distributed heterogeneous computing systems. In: Proceedings of the 12th IEEE International Conference on Parallel and Distributed Systems, 365-374 (2006).

[73] Lu, H. and Carey, M.J., 1986. Load-balanced task allocation in locally distributed computer systems. University of Wisconsin-Madison, Computer Sciences Department.

[74] Luo, J., Dong, F., Cao, J., Song, A.: A novel task scheduling algorithm based on dynamic critical path and effective duplication for pervasive computing environment. Wireless Communications and Mobile Computing, 10(10): 1283-1302 (2010).

[75] Lyu, M R. Handbook of software reliability engineering. CA: IEEE computer society press 1996; 222.

[76] Mehrabi, A., Mehrabi, S. and Mehrabi, A.D., 2009, October. An adaptive genetic algorithm for multiprocessor task assignment problem with limited memory. In Proc. of World Congress on Engineering and Computer Science (Vol. 2, pp. 1018-1023).

[77] Montazeri, F., Salmani-Jelodar, M., Fakhraie, S.N. and Fakhraie, S.M., 2006, September. Evolutionary multiprocessor task scheduling. In International

Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)(pp. 68-76). IEEE.

[78] Musa J D, Anthony I, Okumoto K. Software reliability: measurement, prediction, application. McGraw-Hill, Inc., 1987.

[79] Musa J D. Software reliability engineering: more reliable software, faster and cheaper. Tata McGraw-Hill Education, 2004.

[80] Musa, J.D. and Okumoto, K., 1984, March. A logarithmic Poisson execution time model for software reliability measurement. In Proceedings of the 7th international conference on Software engineering (pp. 230-238). IEEE Press.

[81] Nanda, A.K., Singh, H., Misra, N. and Paul, P., 2003. Reliability properties of reversed residual lifetime. Communications in Statistics-Theory and Methods, 32(10), pp.2031-2042.

[82] Ning, W., Chen, Y. and Tian, X., 2002. A balance between software reliability and cost. Systems Engineering and Electronics, 24(11), pp.117-119.

[83] Ohba, M. and Chou, X.M., 1989, May. Does imperfect debugging affect software reliability growth?. In Proceedings of the 11th international conference on Software engineering (pp. 237-244). ACM.

[84] Omara, F.A. and Arafa, M.M., 2010. Genetic algorithms for task scheduling problem. Journal of Parallel and Distributed Computing, 70(1), pp.13-22.

[85] Painton, L. and Campbell, J., 1995. Genetic algorithms in optimization of system reliability. IEEE Transactions on Reliability, 44(2), pp.172-178.

[86] Peng, R., Hu, Q.P. and Ng, S.H., 2008, September. Incorporating fault dependency and debugging delay in software reliability analysis. InManagement of Innovation and Technology, 2008. ICMIT 2008. 4th IEEE International Conference on (pp. 641-645). IEEE.

[87] Pham H, Nordmann L, Zhang X. A general imperfect-software-debugging model with S-shaped fault-detection rate. IEEE Transactions on Reliability 1999; 48(2): 169-175.

[88] Pham H, Zhang X. An NHPP software reliability model and its comparison. International Journal of Reliability, Quality and Safety Engineering 1997; 4(03): 269-282.

[89] Pham H. Software Reliability, Springer-Verlag, 2000.

[90] Pham H. System software reliability. Springer, 2007.

[91] Pham, H. and Wang, H., 2001. A quasi-renewal process for software reliability and testing costs. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 31(6), pp.623-631.

[92] Pham, H. and Zhang, X., 1999. A software cost model with warranty and risk costs. IEEE Transactions on Computers, 48(1), pp.71-75.

[93] Pham, H., 1996. A software cost model with imperfect debugging, random life cycle and penalty cost. International Journal of Systems Science, 27(5), pp.455-463.

[94] Polychronopoulos, C.D. and Kuck, D.J., 1987. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. Ieee transactions on computers, 100(12), pp.1425-1439.

[95] Price, C.C. and Krishnaprasad, S., 1984, May. Software Allocation Models for Distributed Computing Systems. In ICDCS (pp. 40-48).

[96] Qinma, K., He, H.: A novel discrete particle swarm optimization algorithm for meta-task assignment in heterogeneous computing systems. Microprocessors and Microsystems, 35(1): 10-17 (2011).

[97] Quadri, S.M.K., Ahmad, N. and Farooq, S.U., 2011. Software Reliability Growth modeling with Generalized Exponential testing–effort and optimal SOFTWARE RELEASE policy. Global Journal of Computer Science and Technology, 11(2).

[98] Rallis, N.E. and Lansdowne, Z.F., 2001. Reliability estimation for a software system with sequential independent reviews. IEEE Transactions on Software Engineering, 27(12), pp.1057-1061.

[99] Ramani, S., Gokhale, S.S. and Trivedi, K.S., 2000. SREPT: software reliability estimation and prediction tool. Performance evaluation, 39(1), pp.37-60.

[100] Rao, K.D., Gopika, V., Rao, V.S., Kushwaha, H.S., Verma, A.K. and Srividya, A., 2009. Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. Reliability Engineering & System Safety,94(4), pp.872-883.

[101] Ritchie, G., & Levine, J.: A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In proceedings of of 23rd workshop of the UK planning and scheduling special interest group (2004).

[102] S. Bhatnagar, Karmeshu and V. Mishra, "Optimal parameter trajectory estimation in parameterized SDEs: An algorithmic procedure", ACM

Transactions on Modeling and Computer Simulation (TOMACS), Vol. 19, No. 2, Article No. 8, p. 8:1-8:27.

[103] Sagar, G., Sarje, A.K. and Ahmed, K.U., 1989. On module assignment in two-processor distributed systems: a modified algorithm. Information processing letters, 32(3), pp.151-153.

[104] Salcedo, R.M.J.G., Goncalves, M.J. and De Azevedo, S.F., 1990. An improved random-search algorithm for non-linear optimization. Computers & chemical engineering, 14(10), pp.1111-1126.

[105] Sarhan, A.M. and Zaindin, M., 2009. Modified Weibull distribution. Applied Sciences, 11(1), pp.123-136.

[106] Shanmugam, L. and Florence, L., 2012. A comparison of parameter best estimation method for software reliability models. International Journal of Software Engineering & Applications, 3(5), p.91.

[107] Sharma K, Garg R, Nagpal C K, Garg R K. Selection of optimal software reliability growth models using a distance based approach. IEEE Transactions on Reliability 2010; 59(2): 266-276.

[108] ShAtnAwi O. Measuring commercial software operational reliability: an interdisciplinary modelling approach. Eksploatacja i Niezawodność 2014; 16(4).

[109] Singh, O., Kapur, P.K., Shrivastava, A.K. and Das, L., 2014, October. A unified approach for successive release of a software under two types of imperfect debugging. In Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 2014 3rd International Conference on (pp. 1-6). IEEE.

[110] Singh, Y. and Kumar, P., 2010. Determination of software release instant of three-tier client server software system. International Journal of Software Engineering (IJSE), 1(3), pp.51-62.

[111] Slud, E., 1997. Testing for imperfect debugging in software reliability. Scandinavian journal of statistics, 24(4), pp.555-572.

[112] Stone, H.S., 1977. Multiprocessor scheduling with the aid of network flow algorithms. IEEE transactions on Software Engineering, (1), pp.85-93.

[113] Stringfellow C, Andrews A A. An empirical method for selecting software reliability growth models. Empirical Software Engineering 2002; 7(4): 319-343.

[114] Su, S., Huang, Q., Li, J., Cheng, X., Xu, P., Shuang, K.: Enhanced energy-efficient scheduling for parallel tasks using partial optimal slacking. The Computer Journal (2014), Doi: 10.1093/comjnl/bxu002.

[115] Topcuoglu, H., Hariri, S., Wu, M., Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3): 260-274 (2002).

[116] Verma, A.K., Srividya, A. and Karanki, D.R., 2010. Reliability and safety engineering (pp. 5-7). London: Springer.

[117] Wadekar, S.A. and Gokhale, S.S., 1999. Exploring cost and reliability tradeoffs in architectural alternatives using a genetic algorithm. In Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on(pp. 104-113). IEEE.

[118] Wang, J., Duan, Q., Jiang, Y., Zhu, X.: A new algorithm for grid independent task schedule: genetic simulated annealing. IEEE World Automation Congress (WAC), 165-171 (2010).

[119] Williams, D.P., 2007. Study of the warranty cost model for software reliability with an imperfect debugging phenomenon. Turkish Journal of Electrical Engineering & Computer Sciences, 15(3), pp.369-381.

[120] Wood A. Predicting software reliability. Computer 1996; 29(11): 69-77.

[121] Wu, A. S., Yu, H., Jin, S., Lin, K. C., Schiavone, G.: An incremental genetic algorithm approach to multiprocessor scheduling. Parallel and Distributed Systems, IEEE Transactions on, 15(9), 824-834 (2004).

[122] Wu, Y.P., Hu, Q.P., Xie, M. and Ng, S.H., 2007. Modeling and analysis of software fault detection and correction process by considering time dependency. IEEE Transactions on Reliability, 56(4), pp.629-642.

[123] Xie, M., Hong, G.Y. and Wohlin, C., 1997, November. A practical method for the estimation of software reliability growth in the early stage of testing. In Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on (pp. 116-123). IEEE.

[124] Yadav, P.K., Jumindera, S. and Singh, M.P., 2009. An efficient method for task scheduling in computer communication network. International Journal of Intelligent Information Processing, 3(1), pp.81-89.

[125] Yamada S, Ohba M, Osaki S. S-shaped reliability growth modeling for software error detection. IEEE Transactions on Reliability 1983; 32(5): 475-484.

[126] Yamada, S, Tokuno K, Osaki S. Imperfect debugging models with fault introduction rate for software reliability assessment. International Journal of Systems Science 1992; 23(12): 2241-2252.

[127] Yang, B., Hu, H. and Zhou, J., 2008, January. Optimal software release time determination with risk constraint. In Reliability and Maintainability Symposium, 2008. RAMS 2008. Annual (pp. 393-398). IEEE.

[128] Yang, J., Ma, X., Hou, C. and Yao, Z., 2008, June. A Static Multiprocessor Scheduling Algorithm for Arbitrary Directed Task Graphs in Uncertain Environments. In International Conference on Algorithms and Architectures for Parallel Processing (pp. 18-29). Springer Berlin Heidelberg.

[129] Yellapu, G., Penmetsa, S., K. Modeling of a scheduling problem with expected availability of resources. OPSEARCH, 1(11) (2015), Doi. 10.1007/s12597-015-0203-z.

[130] Yuming, X., Li, K., Khac, T., T., Qiu, M.: A multiple priority queueing genetic algorithm for task scheduling on heterogeneous computing systems. IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 639-646 (2012).

[131] Zahedi, F. and Ashrafi, N., 1991. Software reliability allocation based on structure, utility, price, and cost. IEEE Transactions on Software Engineering, 17(4), pp.345-356.

[132] Zahorjan, J. and McCann, C., 1990. Processor scheduling in shared memory multiprocessors (Vol. 18, No. 1, pp. 214-225). ACM.

[133] Zhang X, Pham H. Comparisons of nonhomogeneous Poisson process software reliability models and its applications. International Journal of Systems Science 2000; 31(9): 1115-1123.

[134] Zhang X, Teng X, Pham H. Considering fault removal efficiency in software reliability assessment. Systems. IEEE Transactions on Man and Cybernetics, Part A: Systems and Humans 2003; 33(1): 114-120.

[135] Zhang, K., Qi, B., Jiang, Q. and Tang, L., 2012, September. Real-time periodic task scheduling considering load-balance in multiprocessor environment. In 2012 3rd IEEE International Conference on Network Infrastructure and Digital Content (pp. 247-250). IEEE.

[136] Zhang, X. and Pham, H., 1998. A software cost model with error removal times and risk costs. International Journal of Systems Science, 29(4), pp.435-442.

[137] Zhi-qiang, X. and Sheng-hui, L., 2003. A Scheduling Algorithm based on ACPM and BFSM [J]. Applied Science and Technology, 30(30), pp.36-38.

[138] Zhu, D., Huang, H. and Yang, S.X., 2013. Dynamic task assignment and path planning of multi-AUV system based on an improved self-organizing map and velocity synthesis method in three-dimensional underwater workspace. IEEE Transactions on Cybernetics, 43(2), pp.504-514.

APPENDICES

APPENDIX A: MULTIPROCESSOR SCHEDULING PROBLEMS

A.1 HOMOGENEOUS MULTIPROCESSOR PROBLEMS FROM LITERATURE

Example 1: This is a homogeneous multiprocessor system with eighteen executable tasks $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}\}$ connected by an arbitrary network as shown in Fig A. taken from Kwok and Ahmad [47]. The tasks have to be scheduled on a set of two processors P={P_1, P_2}. The numerals in red color in Fig A.1 are the communication costs between tasks and other numerals in blue color denote the execution time of that task on each processor.

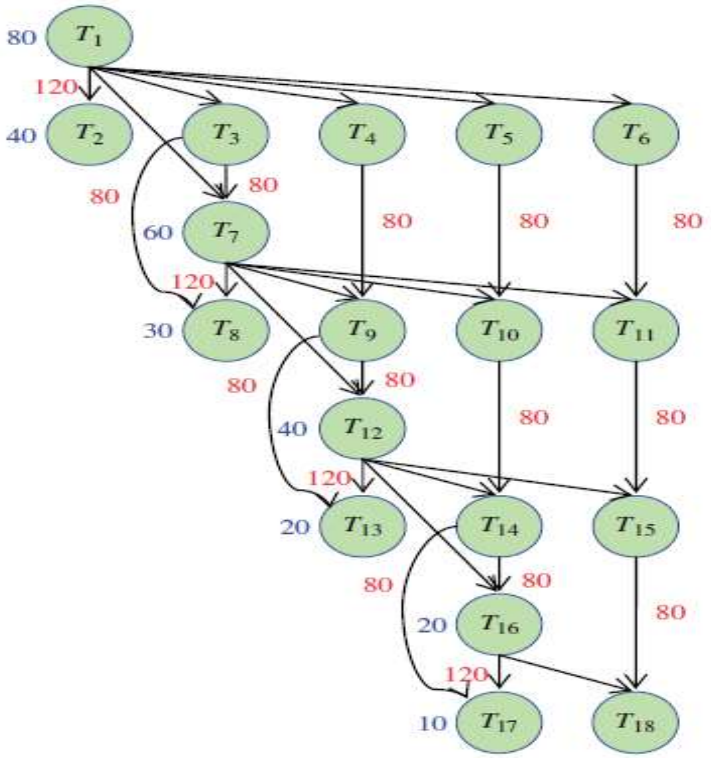

Fig A.1: Example DAG with eighteen tasks

Example 2: This example taken from Hwang et al [47] consisting of nine tasks $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9\}$ to be scheduled on a multiprocessor system with two processors $P = \{P_1, P_2\}$. The value with the node is execution time of each task and the weight assigned to edge is the communication cost between the tasks which is depicted in fig A.2.

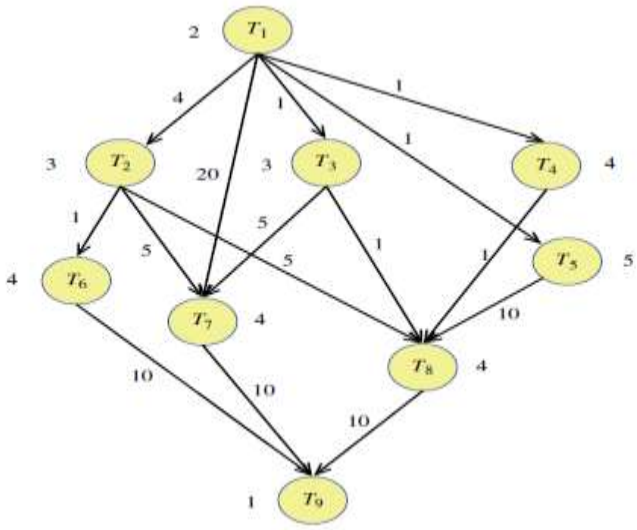

Fig A.2: Example DAG with nine tasks

Example 3: Example taken from [14] consisting of twelve tasks $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}\}$ to be scheduled on a multiprocessor system with two processors $P = \{P_1, P_2\}$. The problem is depicted by DAG given in fig A.3, here value in the node is represented as (a/b), here a is the task number i.e. 1 for T_1 etc. and b represents the execution time of each task and communication cost between tasks is considered zero so edges are not labelled.

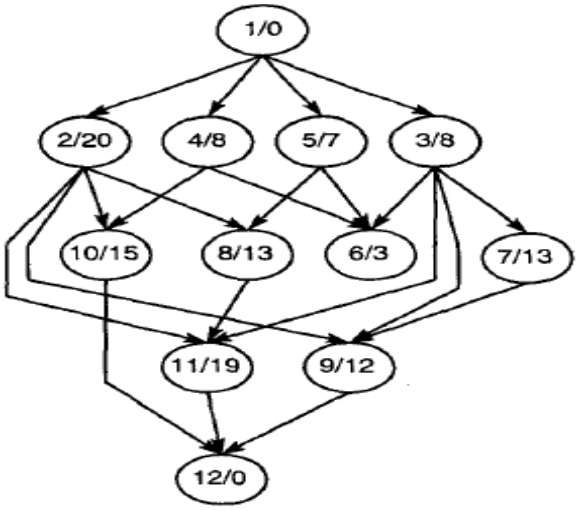

Fig A.3: Example DAG with twelve tasks

Example 4: **E**xample taken from [69] consisting of nine tasks $T = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}$ to be scheduled on a multiprocessor system with three processors $P = \{P_1, P_2, P_3\}$. The problem is depicted by DAG given in fig A.4, where value in the

node is represented as (a/b), here a is the task and b represents the execution time of each task and communication cost between tasks is given as label on edges.

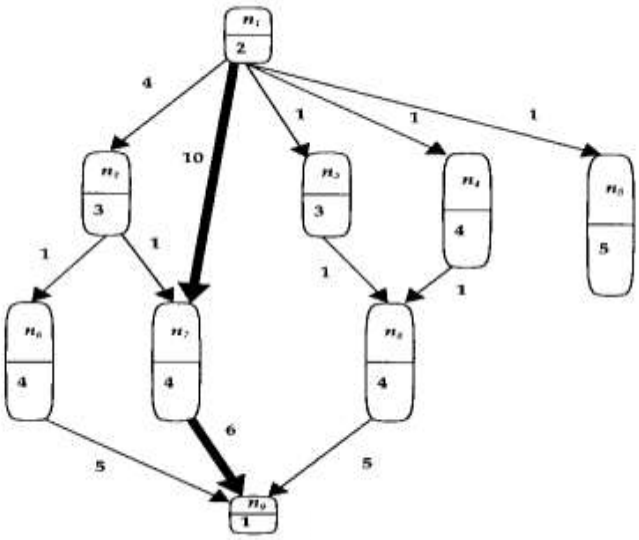

Fig A.4: Example DAG with nine tasks

Example 5: This example is taken from [1] consisting of nine tasks $T = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}$ to be scheduled on a multiprocessor system with three processors $P = \{P_1, P_2, P_3\}$. The problem is depicted by DAG given in fig A.5, where value in the node is represented as (a/b), here a is the task and b represents the execution time of each task and communication cost between tasks is given as label on edges.

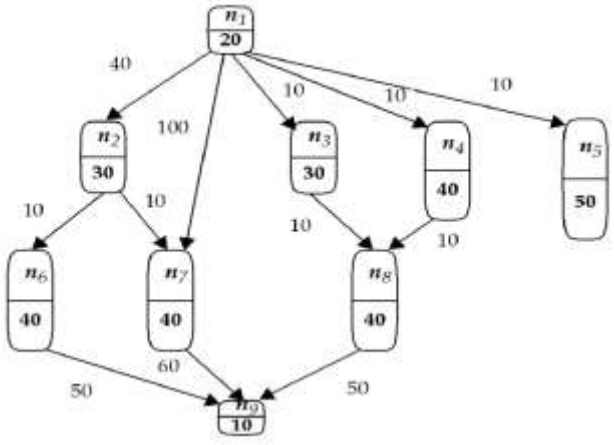

Fig A.5: Example DAG with nine tasks

Example 6: Example given in fig A.6 taken from [70] consisting of eighteen tasks $T = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14}, T_{15}, T_{16}, T_{17}, T_{18}\}$ to be scheduled on a

multiprocessor system with three processors $P = \{P_1, P_2, P_3\}$. Here value in the node is represented as (a/b), here a is the task and b represents the execution time of each task and communication cost between tasks is given as label on edges.

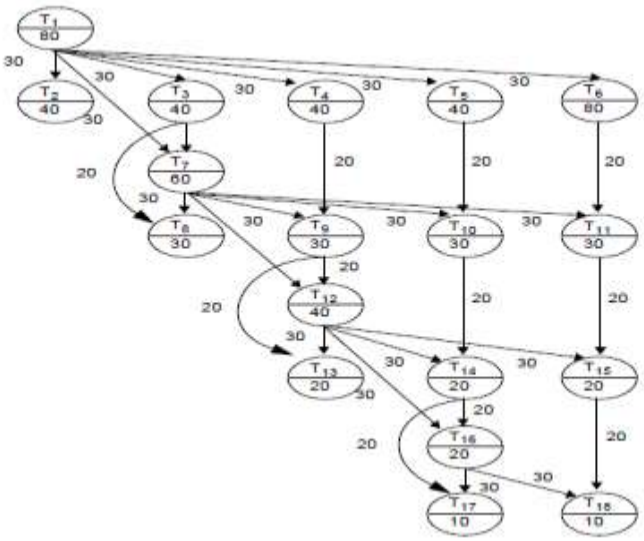

Fig A.6: Example DAG with eleven tasks

Example 7: Example taken from [11] consisting of forty tasks $T = \{T_1, T_2, T_3, \dots, T_{40}\}$ to be scheduled on a multiprocessor system with four processors $P = \{P_1, P_2, P_3, P_4\}$. The problem is depicted by DAG given in fig A.7, communication cost between tasks is considered zero so edges are not labelled.

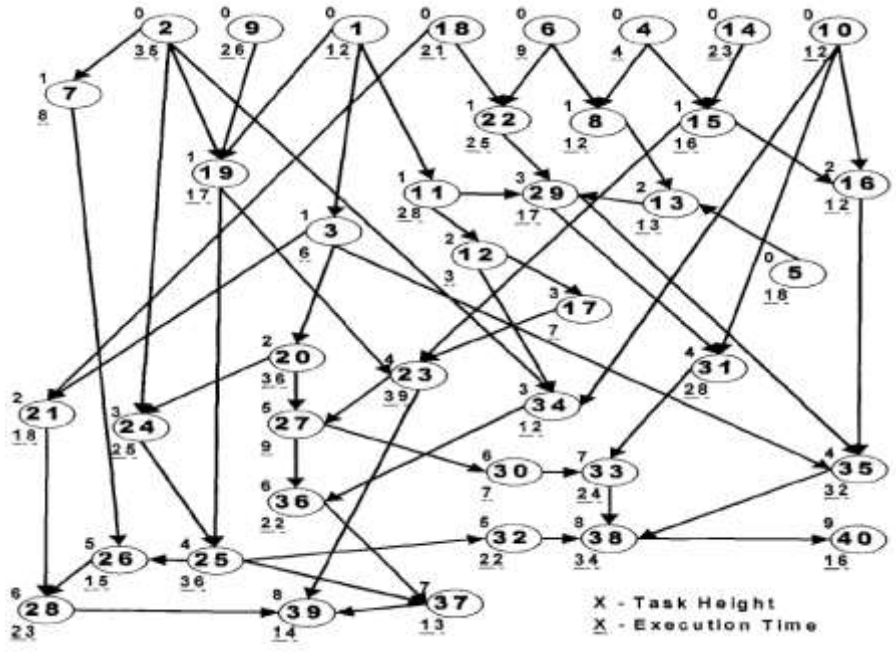

Fig A.7: Example DAG with forty tasks

A.2 HETEROGENEOUS MULTIPROCESSOR PROBLEMS FROM LITERATURE

Example 1: This is a scenario example which is a complex workflow application taken from literature [116]. It is a heterogeneous computing system of ten executable tasks, T= {T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}} denoted with numbers 1 to 10 and a set of three processors P= {P_1, P_2, P_3} connected by an arbitrary network as shown in Fig A.8. The computation cost matrix having execution time for each task on each of the processors is given Table A.1.

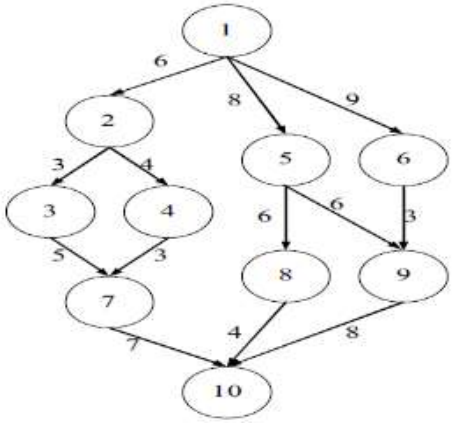

Fig A.8: DAG with 10 tasks to be scheduled on heterogeneous multiprocessor systems

Table A.1: Computation cost matrix of DAG given in Fig A.8.

Task	P_1	P_2	P_3
1	5	3	4
2	3	2	7
3	3	4	8
4	7	5	3
5	5	2	5
6	3	4	8
7	4	2	3
8	2	3	4
9	4	5	3
10	6	4	5

Example 2: this is also a heterogeneous computing system of ten executable tasks, T= {T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}} on a set of three processors P= {P_1, P_2, P_3} connected by an arbitrary network as shown in Fig A.9 taken from [138]. The communication cost matrix having execution time for each task on each of the processors is given table A.2.

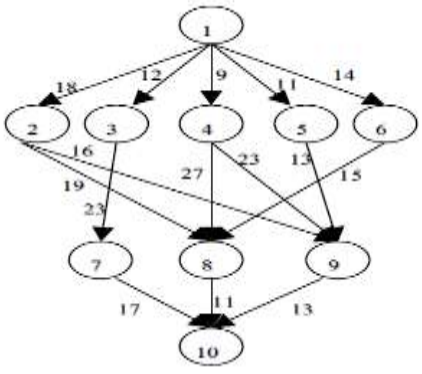

Fig A.9. DAG with 10 tasks to be scheduled on heterogeneous multiprocessor systems

Table A.2: Computation cost matrix of DAG given in Fig A.9.

Task	P_1	P_2	P_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

Example 3: It is a heterogeneous computing system of eleven executable tasks, T= {T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}} denoted with numbers 1 to 11 and a set of four processors P= {P_1, P_2, P_3, P_4} connected by an arbitrary network as shown in Fig A.10. The computation cost matrix having execution time for each task on each of the processors is given Table A.3.

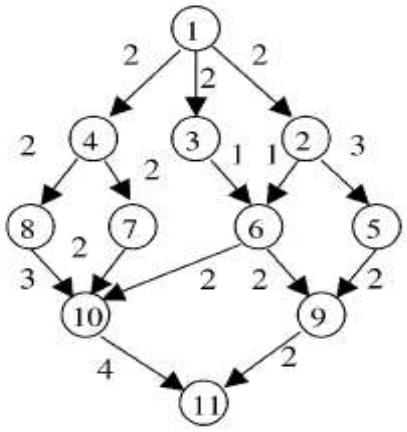

Fig A.10: DAG with 11 tasks to be scheduled on heterogeneous multiprocessor systems

Table A.3: Computation cost matrix of DAG given in Fig A.10.

Task	P_1	P_2	P_3	P_4
1	4	4	4	4
2	5	5	5	5
3	4	6	4	7
4	3	3	3	3
5	3	5	3	4
6	3	7	2	2
7	5	8	5	5
8	2	4	5	3
9	5	6	7	5
10	3	7	5	2
11	5	6	7	8

Example 4: It is a heterogeneous computing system of ten executable tasks, T= {T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}} denoted with numbers 1 to 10 and a set of two processors P= {P_1, P_2, P_3, P_4} connected by an arbitrary network as shown in Fig A.11 taken from [74]. The computation cost matrix having execution time for each task on each of the processors is given Table A.4.

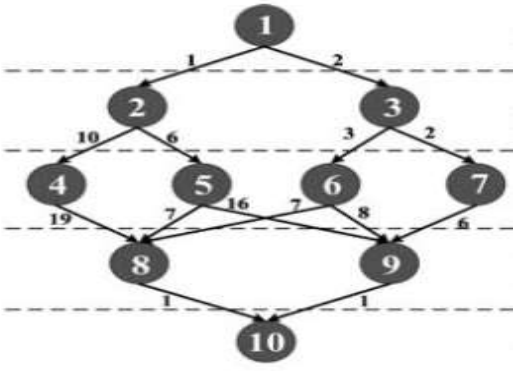

Fig A.11. DAG with 10 tasks to be scheduled on heterogeneous multiprocessor systems

Table A.4: Computation cost matrix of DAG given in Fig A.11.

Task	P_1	P_2
0	4	6
1	15	22.5
2	4	6
3	13	19.5
4	10	15
5	7	10.5
6	8	12
7	4	6
8	12	18
9	6	9
10	9	13.5

Example 5: It is a heterogeneous computing system of eleven executable tasks, T= {T_0, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}} denoted with numbers 0 to 10 and a set of two processors P= {P_1, P_2} connected by an arbitrary network as shown in Fig A.12 taken from [19]. The computation cost matrix having execution time for each task on each of the processors is given Table A.5.

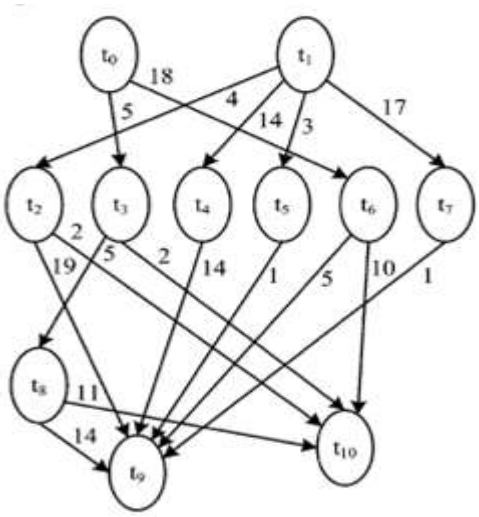

Fig A.12: DAG with 11 tasks to be scheduled on heterogeneous multiprocessor systems

Table A.5: Computation cost matrix of DAG given in Fig A.12.

Task	P_1	P_2
1	2	2
2	1	5
3	2	2
4	3	3
5	7	5
6	9	5
7	5	12
8	6	3
9	3	2
10	2	2

APPENDIX B: DATASETS USED IN PRESENT STUDY

DATASET 1 [105]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	16	11	81
2	24	12	86
3	27	13	90
4	33	14	93
5	41	15	96
6	49	16	98
7	54	17	99
8	58	18	100
9	69	19	100
10	75	20	100

DATASET 2 [69]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	28	10	125
2	29	11	139
3	29	12	152
4	29	13	164
5	29	14	164
6	37	15	165
7	63	16	168
8	92	17	170
9	116	18	176

DATASET 3 [69]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	90	10	190
2	107	11	192
3	126	12	192
4	145	13	192
5	171	14	192
6	188	15	203
7	189	16	203
8	190	17	204
9	190		

DATASET 4 [69]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	9	8	63
2	14	9	70
3	21	10	75
4	28	11	76
5	53	12	76
6	56	13	77
7	58		

DATASET 5 [118]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	15	12	149
2	35	13	157
3	60	14	173
4	74	15	179
5	94	16	182
6	102	17	184
7	114	18	185
8	134	19	187
9	139	20	191
10	141	21	192
11	148		

DATASET 6 [75]					
TEST TIME	FAILU-RES	TEST TIME	FAILU-RES	TEST TIME	FAILU-RES
1	5	38	324	75	469
2	10	39	331	76	469
3	15	40	346	77	470
4	20	41	367	78	472
5	26	42	375	79	472
6	34	43	381	80	473
7	36	44	401	81	473
8	43	45	411	82	473
9	47	46	414	83	473
10	49	47	417	84	473
11	80	48	425	85	473
12	84	49	430	86	473
13	108	50	431	87	475
14	157	51	433	88	475
15	171	52	435	89	475
16	183	53	437	90	475
17	191	54	444	91	475
18	200	55	446	92	475
19	204	56	446	93	475
20	211	57	448	94	475
21	217	58	451	95	475
22	226	59	453	96	476
23	230	60	460	97	476
24	234	61	463	98	476
25	236	62	463	99	476
26	240	63	464	100	477
27	243	64	464	101	477
28	252	65	465	102	477
29	254	66	465	103	478
30	259	67	465	104	478
31	263	68	466	105	478
32	264	69	467	106	479
33	268	70	467	107	479
34	271	71	467	108	479
35	277	72	468	109	480
36	293	73	469	110	480
37	309	74	469	111	481

DATASET 7 [11]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	13	11	95
2	18	12	100
3	26	13	104
4	34	14	110
5	40	15	112
6	48	16	114
7	61	17	117
8	75	18	118
9	84	19	120
10	89		

DATASET 8 [74]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	6	7	48
2	9	8	54
3	13	9	57
4	20	10	59
5	28	11	60
6	40	12	61

DATASET 9 [74]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	1	11	32
2	3	12	32
3	8	13	36
4	9	14	38
5	11	15	39
6	16	16	39
7	19	17	41
8	25	18	42
9	27	19	42
10	29		

DATASET 10 [14]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	27	14	111
2	43	15	116
3	54	16	122
4	64	17	122
5	75	18	127
6	82	19	128
7	84	20	129
8	89	21	131
9	92	22	132
10	93	23	134
11	97	24	135
12	104	25	136
13	106		

DATASET 11 [16]			
TEST TIME	FAILURES	TEST TIME	FAILURES
1	7	19	888
2	29	20	978
3	61	21	1024
4	108	22	1081
5	134	23	1110
6	159	24	1150
7	175	25	1166
8	223	26	1184
9	259	27	1221
10	312	28	1236
11	369	29	1244
12	408	30	1272
13	479	31	1278
14	559	32	1283
15	624	33	1286
16	681	34	1289
17	771	35	1301
18	831		