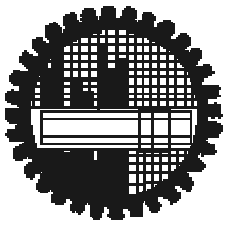
ROBUST AND REACTIVE SCHEDULING FOR MULTI-OBJECTIVE FLEXIBLE JOB SHOP PROBLEMS WITH UNPREDICTED ARRIVAL OF NEW JOBS

BIJOY DRIPTA BARUA CHOWDHURY



DEPARTMENT OF INDUSTRIAL AND PRODUCTION ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING & TECHNOLOGY
DHAKA-1000, BANGLADESH

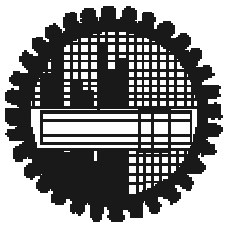
June, 2015

ROBUST AND REACTIVE SCHEDULING FOR MULTI-OBJECTIVE FLEXIBLE JOB SHOP PROBLEMS WITH UNPREDICTED ARRIVAL OF NEW JOBS

BY

BIJOY DRIPTA BARUA CHOWDHURY

A thesis submitted to the Department of Industrial & Production Engineering, Bangladesh University of Engineering & technology, in partial fulfillment of the requirements for the degree of Master of Science in Industrial & Production Engineering



DEPARTMENT OF INDUSTRIAL AND PRODUCTION ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING & TECHNOLOGY
DHAKA-1000, BANGLADESH

June, 2015

CERTIFICATE OF APPROVAL

The thesis titled "**Robust and Reactive Scheduling for Multi-Objective Flexible Job Shop Problems with Unpredicted Arrival of New Jobs"** submitted by Bijoy Dripta Barua Chowdhury**,** Student no: 0413082017 P has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Industrial & Production Engineering on June 21, 2015.

BOARD OF EXAMINERS

_____ Dr. Ferdous Sarwar Assistant Professor Department of IPE, BUET	Chairman Supervisor
_____ Head Department of IPE, BUET	Member (Ex-Officio)
_____ Dr. M. Ahsan Akhtar Hasin Professor Department of IPE, BUET	Member
_____ Dr. Shuva Ghose Assistant Professor Department of IPE, BUET	Member
_____ Dr. Mohammad Iqbal Professor Department of IPE, SUST	Member (External)

CANDIDATE'S DECLARATION

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

_____

Bijoy Dripta Barua Chowdhury

To my family

ACKNOWLEDGEMENT

At the very beginning the author expresses his sincere gratitude and profound indebtness to his thesis supervisor Dr. Ferdous Sarwar, Assistant Professor, Department of Industrial & Production Engineering, BUET, Dhaka-1000, under whose continuous supervision this thesis was carried out. His affectionate guidance, valuable suggestions and inspirations throughout this work made this study possible.

The author also expresses his gratitude to Mr. Kayser Abdullah, Lecturer, Department of Computer Science and Engineering for his assistance in coding the constraint equation of the mathematical model proposed in this thesis.

Finally, the author would like to thank all of his colleagues and friends for their co-operation and motivation to complete the thesis timely. And the author would also like to extend his thanks to his parents whose continuous inspiration, sacrifice and support encouraged him to complete the thesis successfully.

ABSTRACT

Scheduling in production systems concludes the proper coordination of activities in order to increase productivity and reduce operational costs. In dynamic manufacturing environments, scheduling solutions based on the classical objectives such as makespan will not be sufficient. In fact, because of random disruptions that may occur in the system, additional criteria that have capability to counter such disruptions should be considered. To maintain system performance effective, rescheduling is often used to counteract the effects of random disruptions.

In practical production environments, the scheduling process starts with determining an initial schedule. Then, when a disruption arises, the initial schedule should be revised in order to keep its feasibility and performance quality. The type of scheduling that is actually carried out in shops is known as real schedule. As it is clear, real schedule can be different from the initial schedule. This difference depends on the level of failure and disruption and also the changes of the setting. There are two policies to achieve a high level of system performance for the real schedule after occurring of any disruption. These strategies are entitled *reactive* scheduling and *proactive* scheduling.

The "*reactive approach*" does not consider the uncertainty when an initial schedule is determined. However, when a random event occurs, it modifies the initial schedule and performs the necessary reaction to obtain better result. This reaction can be in the form of modification and improvement of the initial schedule or the formulation of a totally-new schedule. On the other hand, the "*proactive approach*" considers the stochastic and unexpected events to create the initial schedule. In this approach, in addition to classical criteria such as makespan and tardiness, performance measures such as robustness and stability is also considered to establish a schedule. Optimization of stability is concerned with the deviation of the modified schedule relative to the initial schedule. Optimization of robustness is concerned with the different in terms of objective function (performance criteria) between initial and modified schedules. An integrated proactive–reactive approach can also be considered to generate better and practical results.

In this thesis, a two-step proactive–reactive method is presented for flexible job shop scheduling to achieve a more stable and robust solution. In the first step, it is attempted to generate an initially robust schedule by using robust optimization approach. The initial robust schedule handles the uncertain processing times. In the second step, when a random disruption occurs (which is the arrival of an unpredicted new job), an appropriate reaction is adopted to determine the best modified schedule

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

N_j	$set\ of\ all\ operations\ of\ job\ j, i \in N_j$
N_{jt}	$set\ of\ incomplete\ operations\ of\ job\ j\ at\ time\ t$
J	$set\ of\ all\ jobs, j \in J$
J_t	$set\ of\ incomplete\ jobs\ at\ time\ t, j \in J_t is\ subset\ of\ J$
K	$set\ of\ all\ machines\ k \in K$
$M_{j,i}$, $(M_{l,h})$	$set\ of\ machines\ allowed\ to\ process\ operation\ i\ (h)\ of\ job\ j\ (l)$
Ω	$set\ of\ all\ scenarios$
i , h	$indices\ for\ operation\ of\ job$
j, l	$indices\ for\ job$
k	$indices\ for\ machine$
λ , λ'	$indices\ for\ scenarios$
t	$time\ of\ the\ arrival\ of\ the\ unpredicted\ job$
n_j	$total\ number\ of\ operations\ of\ job\ j$
n	$total\ number\ of\ jobs$
m	$total\ no\ of\ machines$
$P_{j,i,k}^{\lambda}$	$processing\ time\ of\ operation\ i\ of\ job\ j\ in\ machine\ k\ in\ scenario\ \lambda$
p^{λ}	$probability\ of\ occurring\ scenario\ \lambda$
M	$a\ sufficiently\ large\ number$
a_k	$time\ at\ which\ resource\ k\ is\ available\ for\ starting\ the\ next\ operation$
$\alpha, (\beta), \{\gamma\}s$	$objective\ weight\ of\ makespan, (stability), \{robustness\}$
$C_{j,i}^{\lambda}$	$completion\ time\ of\ operation\ i\ of\ job\ j\ in\ scenario\ \lambda$
$C_{\ max}$	$Makespan\ of\ all\ jobs\ or\ maximum\ completion\ time$
$X_{j,i,k}$	$\begin{cases} 1, if\ operation\ i\ of\ job\ j\ is\ allocated\ to\ resource\ k \\ 0, otherwise \end{cases}$
$X_{j,i,l,h}$	$\begin{cases} 1, if\ operation\ i\ of\ job\ j\ is\ processed\ after\ operation\ h\ of\ job\ l \\ 0, otherwise \end{cases}$
$e_{j,i,k}$	$\begin{cases} 1, if\ operation\ i\ of\ job\ j\ is\ allowed\ to\ be\ processed\ in\ machine\ k \\ 0, otherwise \end{cases}$

ABBREVIATIONS

FJSP	Flexible Job Shop Problem
MIP	Mixed-Integer programming
MINLP	Mixed-Integer nonlinear programming
VLSI	Very large scale integration
MOGA	Multi-objective genetic algorithm

CHAPTER I

INTRODUCTION

Scheduling is a decision-making process that is used on a regular basis in many manufacturing and service industries. It deals with the allocation of resources to tasks over a given time periods and its goal is to optimize one or more objectives.

The resources and tasks in an organization can take many different forms. The resources may be machines in a workshop, runways at an airport, crews at a construction site, processing units in a computing environment, and so on. The tasks maybe operations in a production process, take-offs and landings at an airport, stages in a construction project, executions of computer programs, and so on. Each task may have a certain priority level, an earliest possible starting time and a due date. The objectives can also take different forms. One objective may be minimization of completion time of the last task and another may be minimization of the number of tasks completed after their respective due dates.

Scheduling as a decision-making process plays an important role in most manufacturing and production systems as well as in most information processing environments. It is also important in transportation and distribution settings and in other types of service industries. Therefore scheduling problems have always drawn the attention of the researchers due to its complexity to solve and applicability in numerous areas.

1.1 Rationale of the study:

Thirst for increased productivity in the modern business world has spurred manufacturing practitioners to seek every single opportunity for cost reduction and profit generation [1]. Over the last six decades, effective production scheduling mechanisms have been recognized to be increasing productivity and machine utilization. Algorithmic and scientific production scheduling came into existence once the first production scheduling heuristic technique was proposed by [2]. After that, a great deal of efforts has been spent on developing optimal production schedule, and countless papers have already published in the scholarly journals.

Amongst all scheduling problems, Job Shop Scheduling is considered as one of the hardest problem because of its nature which even becomes much harder to solve with the introduction of unexpected events like machine failure, urgent job arrival, job cancellation, due date change etc. In the real manufacturing environment, scheduling is an ongoing reactive process where the presences of these unexpected events are inevitable. The recent comprehensive survey on dynamic job shop scheduling problems [3] summarizes the principles of several dynamic scheduling techniques. In the stochastic and dynamic manufacturing environments, scheduling solutions based on the classical objectives such as makespan is not sufficient. In fact, because of random disruptions that may occur in the system, additional criteria that have the capability to counter stochastic disruptions should be considered. To maintain system performance effectively, rescheduling is often used to counteract the effects of random disruptions. However, to minimize the effect of rescheduling proactive approach is adopted to make a robust schedule which is insensitive to such disruptions.

Most scheduling problems including FJSP have been considered as NP-hard. Hence, heuristic and meta-heuristic approaches have received much attention in the literature in addition to optimizing several performance measures [4-6]. In the recent literature, two new criteria have drawn the attention of researchers for their consideration: robustness and stability. Daniels and Kouvelis generate a robust job sequence for a single machine under processing time variability such that the degradation of the performance measure under the worst possible scenario is minimized [7]. In [8, 9] stability was enhanced by inserting additional idle times while generating schedules. The work at [10] is also on generating stable schedules considering the total tardiness as the performance measure. Jensen tried to improve the robustness and flexibility of the job shop schedules when minimizing maximum tardiness, summed tardiness, total flow time, and makespan measures [11]. Some robustness and stability measures were discussed at [12]. These measures were further used at [13] for optimizing the schedule robustness and stability. A robust and stable scheduling of a single machine with random machine breakdowns was presented on [14] which is further extended in the study at [15] in the FJSP using a two-stage hybrid genetic algorithm.

Though some research works have already been done on these two criterions, there is still a lot of scopes to improve the scheduling methodology coupled with introducing new factors to make the schedule more adaptive in the real manufacturing environment. Therefore, a new multi-objective robust and flexible job shop schedule in addition to considering unexpected arrival of jobs and processing time variability at different scenarios is developed.

1.2 Objectives with Specific Aims and Possible Outcomes:

The specific objectives of this research are

i. To develop a multi-objective reactive flexible job shop scheduling model to minimize the makespan and maximize the robustness and stability.

ii. To integrate scenario based processing time variation.

iii. To incorporate a newly arrived unpredicted job into the system by generating a new schedule after the arrival.

iv. To solve the proposed optimization model with the help of a suitable algorithm and programming software.

The possible outcome of this research are,

i. Validated mathematical model of Flexible Job Shop.
ii. Gantt chart of scheduled job
iii. Rescheduled jobs after the arrival of new one.

1.3Outline of Methodology:

This research work is theoretical in nature. A mathematical FJSP model is developed considering some factors significantly affecting the scheduling decisions such as sequence of operation, makespan, robustness and stability of the overall system. The model is composed of some mathematical equations used to determine the numerical values of some decision variables (sequence of operations at different machines, completion time of each operation and job). The research methodology is outlined as follows:

i. The three objective functions are developed for minimizing the makespan and maximizing the robustness and stability.
ii. The probability of occurring a certain scenario to incorporate the processing time variability is assumed and the processing time at different scenarios are considered available from the historical data.
iii. The different necessary constraints are developed to ensure that the proposed model is a bounded one.
iv. Branch and Cut and Genetic algorithm are selected to solve the proposed multi-objective optimization model.
v. Finally, the model will be illustrated and validated with several numerical examples.

CHAPTER II

LITERATURE REVIEW

The job-shop scheduling problem (JSP) has been well studied in the manufacturing systems field during the past few decades. The classic JSP, which is a combinatorial optimization problem, is strongly NP-hard. An extension of the JSP, the flexible job-shop scheduling problem (FJSP) has received considerable attention in the field. The FJSP consists of two sub-problems, including machine assignment and operation sequence. The FJSP is concerned with finding the most efficient assignment and operation sequence of n jobs in m machines. The output of a FJSP is a Gantt Chart which specifies the sequence of operations at different machines. The mathematical modeling of FJSP is complex, because there exist several criteria that must be taken into consideration when formulating and solving the model. In case of multi-objective FJSP model, there may be some criteria which are conflicting, perhaps non-commensurate. This imposes pressure upon the researchers to implement an appropriate and realistic mathematical modeling of a FJSP.

Some research papers have been studied to understand the background of the study. Many factors have been considered to develop the previous FJSP models. Different methodologies have been followed to develop and solve the models. In this section, some research papers have been studied to understand the factors and methodologies considered by researchers.

The FJSP is a more complex version of the JSP and is also strongly NP-hard [16]. Bruker and Schlie [17] were among the first to address the FJSP and proposed a polynomial algorithm to the problems with two jobs. Most existing research addressed this problem with the assumption that the parameters are known and deterministic. However, in most real-world environments, scheduling is an ongoing reactive process where the presence of a variety of unexpected disruptions is usually inevitable, and continually forces reconsideration and revision of pre-established schedules. Many of the approaches developed to solve the problem of static scheduling are often impractical in real-world environments, and the near-optimal schedules with respect to the estimated data may become obsolete when they are released to the shop floor. Examples of such real-time events include machine failures, arrival of urgent jobs,

due date changes, etc. Mac Carthy and Liu [18] addressed the nature of the gap between the scheduling theory and scheduling practice, the failure of classical scheduling theory to respond to the needs of practical environments, and recent trends in scheduling research which attempt to make it more relevant and applicable. Shukla and Chen [19], in their comprehensive survey on intelligent real-time control in flexible manufacturing systems, stated that comparison of theory and scheduling practice showed very little correspondence between the two. Cowling and Johansson [20] addressed an important gap between scheduling theory and practice, and stated that scheduling models and algorithms are unable to make use of real-time information.

As mentioned previously, most scheduling problems including FJSP have been considered as NP-hard [16]. Hence, heuristic and meta-heuristic approaches have received much attention in literature. Numerous papers addressed stochastic single machine with uncertain jobs processing times. Such as the work of Daniels and Kouvelis in [7] where they formalized the robust scheduling concept for scheduling situations with uncertain or variable processing times. They illustrated the development of solution approaches for a robust scheduling problem by considering a single-machine environment where the performance criterion of interest is the total flow time over all jobs. They defined two measures of schedule robustness, formulated the robust scheduling problem, established its complexity, described properties of the optimal schedule, and finally presented exact and heuristic solution procedures. They also reported extensive computational results to demonstrate the efficiency and effectiveness of the proposed solution procedures.

Kouvelis and Daniels extended their work for two-machine flow shop environment in which the processing times of jobs were uncertain and the performance measure of interest was system makespan. They presented a measure of schedule robustness that explicitly considers the risk of poor system performance over all potential realizations of job processing times. Two alternative frameworks for structuring processing time uncertainty were discussed by them. For each case, they defined the robust scheduling problem, established problem complexity, discussed properties of robust schedules, and developed exact and heuristic solution approaches. Computational results

indicated that robust, schedules provide effective hedges against processing time uncertainty while maintaining excellent expected makespan performance.

Michel L. Pinedo, who is a very familiar researcher in the field of studying scheduling problems have conducted numerous researches on various scheduling problems. One of his most famous book, “Scheduling Theory, Algorithms and Systems” [21] he has included almost all types of scheduling problems, discussed about their complexities and some solution methodologies. In a work previously done by him [22], a stochastic scheduling model was considered where all parameters like processing time, release dates, due dates etc. were independent random variables. They were studied the computational complexities of determining optimal job shop policies for the stochastic scheduling model and illustrated it by some examples. The most important outcome of this research was researchers showed that some optimal policies can be determined by polynomial time algorithms.

Möhring, Radermacher, and Weiss [23] introduced the finite class of set strategies for stochastic scheduling problems. They showed that the set strategies provide satisfactory stability behavior compared to stable classes of strategies such as Evolution Strategies (ES) and Modular Evolution Strategies (MES) strategies and list-scheduling strategies such as Longest Expected Processing Time (LEPT), Shortest Expected Processing Time (SEPT) and other more complicated priority-type strategies.

Montemanni and Roberto[24] considered a version of the total flow time single machine scheduling problem where uncertainty about processing times is taken into account. They adopted the interval data model, where finite intervals of (equally possible) values for the completion time of each job are given.. Namely an interval of equally possible processing times is considered for each job, and optimization is carried out according to a robustness criterion. This model was originally proposed by Daniels and Kouvelis[25]. They proposed the first mixed integer linear programming formulation for the resulting optimization problem and explained how some known preprocessing rules can be translated into valid inequalities for this formulation.

A robust scheduling method was proposed to solve uncertain scheduling problems by Wu [26] where an uncertain scheduling problem is modeled by a set of workflow

models, and then a scheduling scheme (solution) of the problem was evaluated by workflow simulations executed with the workflow models in the set. They also presented a multi-objective immune algorithm to find Pareto optimal robust scheduling schemes that have good performance for each model in the set. The two optimization objectives for scheduling schemes were the indices of the optimality and robustness of the scheduling results. An antibody represented the resource allocation scheme, and the methods of antibody coding and decoding was designed to deal with resource conflicts during workflow simulations. Experimental tests show that the proposed method can generate a robust scheduling scheme that is insensitive to uncertain scheduling environments.

Xia [27] considered due date assignment and sequencing for multiple jobs in a single machine shop where the processing time of each job was assumed to be uncertain and was characterized by a mean and a variance with no knowledge of the entire distribution. To minimize a linear combination of three penalties: penalty on job earliness, penalty on job tardiness, and penalty associated with long due date assignment a heuristic procedure was developed to find job sequence and due date assignment. Numerical experiments indicated that the performance of the procedure is stable and robust to job processing time distributions. In addition, the performance improved when the means and variances of job processing times are uncorrelated or negatively correlated or when the penalty of a long due date assignment is significant.

Both Al-Turki [28] and Cai and Tu[29] considered sequencing n jobs on a single machine. The objective in [14] was to minimize an expected weighted combination of due dates, completion times, earliness, and tardiness penalties. The determination of optimal distinct due dates or optimal common due dates for a given schedule was investigated. The scheduling problem for a fixed common due date was considered when random completion times arise from machine breakdowns. The optimality of a v-shaped about (a point) T sequence was established in the work when the number of machine breakdowns follows either a Poisson or a geometric distribution and the duration of a breakdown has an exponential distribution. On the other hand in [15], the processing time of each job was considered as a random variable which follows an arbitrary distribution with a known mean and a known variance. The machine is subject to stochastic breakdowns and the objective was to minimize the expected sum

of squared deviations of job completion times from the due date. Two versions of the problem were addressed. In the first one the due date was a given constant, whereas in the second one the due date was a decision variable. In each case, a general form of the deterministic equivalent of the stochastic scheduling problem is obtained when the counting process related to the machine uptime distribution was a generalized Poisson process. A sufficient condition was derived under which optimal sequences were V-shaped with respect to mean processing times which were same as in [14]. Other characterizations of optimal solutions were also established. Based on the optimality properties, algorithms with pseudo polynomial time complexity were proposed to solve both versions of the problem. Liu [30] also considered a single machine shops subjected to machine breakdowns. Furthermore, Sevaux and Sorensen [31] used a modified GA to find robust solution in single machine environment subjected to stochastic release dates of jobs.

Byeon [32], Kutanoglu and Wu [33, 34], and Wu [35] used decomposition heuristic to divide the classical job shop scheduling problem with uncertain processing times into a series of sub problems and iteratively update the problem parameters to analyze the effect of the processing time variation using a priori stochastic information.

In [32] a weighted tardiness job-shop scheduling problems (JSP) was decomposed into a series of sub-problems by solving a variant of the generalized assignment problem using a graph decomposition technique which termed as "VAP." Given a specified assignment cost, VAP assigned operations to mutually exclusive and exhaustive subsets, identifying a partially specified schedule. Compared to a conventional, completely specified schedule, this partial schedule was more robust to shop disturbances, and therefore more useful for planning and control. Indeed it was showed that the proposed approach provides a means for extending traditional scheduling capabilities to a much wider spectrum of shop conditions and production scenarios. Like [32], to improve scheduling robustness under processing time variation, a two-stage scheme was proposed in [19] that preprocesses the scheduling data to create a skeleton of a schedule and then completes it over time through dynamic adaptation. Preprocessing starts at the beginning of the planning period (at the time of scheduling) when a priori information becomes available on processing time uncertainty. The job shop scheduling problem was decomposed problem into

network-structured sub-problems using Lagrangian relaxation. For each sub-problem, stochastic constraint was introduced in the model that captures the processing time uncertainty. The stochastic information was incorporated in such a way that it retains their efficient network structure. A sub-gradient search algorithm was used to improve the lower and upper bounds iteratively obtained from the Lagrangian relaxed problem, which produce a partial sequence of critical operations. This so-called Lagrangian ranking defined a preprocessed schedule where the complete scheduling was determined dynamically over time, adapting to changing shop conditions. A similar category problem was solved in [35] by using decomposition technique where it identifies and resolves a "crucial subset" of scheduling decisions through the use of a branch-and-bound algorithm.

Shafaei and Brunn [36, 37] used the rolling time approach to investigate the robustness of schedules in dynamic and stochastic environment. In [22] a cost-based performance measure was used to evaluate the scheduling rules. The simulation results, under various conditions in a balanced and unbalanced shop were presented and the effects of the rescheduling interval and operational factors including shop load conditions and a bottleneck on the robustness of the schedule were studied. From the results the key factors that influence the robustness of a scheduling system were identified and, consequently, general guidelines for creating robust schedules were proposed. In [23] a comprehensive simulation study was conducted to investigate the effects of the planning-i.e. job releasing and routing-and the scheduling functions in creating a robust schedule. A mathematical model using the integer programming technique in addition with a heuristic algorithm were used to demonstrate the solution It was shown that, in terms of shop load balance level and job delivery time, the proposed system performs better than a benchmark loading strategy on the basis of minimum processing cost.

Cowling [38] used a previously proposed multiagent architecture with two measures of stability and utility to produce a robust predictive/reactive schedule. In contrast to earlier approaches, the multi-agent architecture proposed consists of a set of heterogeneous agents which integrate and optimize a range of scheduling objectives related to different processes, and can adapt to changes in the environment while still achieving overall system goals. Each agent embodied its own scheduling model and

realized its local predictive-reactive schedule taking into account local objectives, real-time information and information received from other agents. In order to find a globally good schedule, the cooperation of the agent was done which was able to effectively react to real-time disruptions, and to optimize the original production goals whilst minimizing disruption carried by unexpected events occurring in real-time.

Policella [39, 40] studied two-stage approach to generate a robust flexible partial order schedule for the Resource-Constrained Project Scheduling problem with minimum and maximum time lags. The problem of transforming a resource feasible, fixed-times schedule into a partial order schedule (POS) to enhance its robustness and stability properties was considered. A POS retains temporal flexibility whenever problem constraints allow it and can often absorb unexpected deviation from predictive assumptions. The work specifically focused on procedures for generating Chaining Form POSs, wherein activities competing for the same resources are linked into precedence chains. The interesting property of the Chaining Form POS implemented was that "makespan preserving" with respect to its originating fixed-times schedule. Therefore, issues of maximizing schedule quality and maximizing schedule robustness can be addressed sequentially in a two-step scheduling procedure. Moreover, two heuristics were defined to make the use of a structural property of chaining form POSs to bias chaining decisions and experimental results on a resource-constrained project scheduling benchmark were presented to confirm the effectiveness this approach.

Leon [41] proposed a slack-time based robustness measures to analyze the effects of machine breakdowns and processing-time variability on the quality of the classical job shop schedules. The most promising robustness measure is found to be

$$Z_{r=1} = MS_{min} - RD3(s)$$

where, MS_{min} is the makespan of schedule s, and RD3(s) is the average operation slack in schedule s.

Lawrence and Sewell [42] studied the performance of simple dispatching heuristics versus algorithmic solution techniques in job shops subjected to uncertain processing times. A similar study was done by Sabuncuoglu and Karabuk [43], which showed that dispatching rules are more robust to interruptions than the optimum seeking

offline scheduling algorithms. Mehta and Uzsoy [44] presented an algorithm based on disjunctive graph representation to integrate random breakdowns of machines and minimizing their effect by inserting idle time into the predictive schedule of a job shop to absorb the impact of breakdowns. Jensen [11, 45] tried to improve the robustness and flexibility of the job shop schedules while minimizing maximum tardiness, summed tardiness, total flow-time, and makespan measures. Both studies used GA (developed in Mattfeld [46]) and considered two robustness measures, the neighborhood-based robustness measure and the lateness-based robustness measure. He defined schedule neighborhood $N_1(s)$ robustness measure, where $N_1(s)$ contains s and all feasible schedules that can be created from s by interchanging two consecutive operations on the same machine, as a weighted average of makespans of schedules in $N_1(s)$ and is given as follows

$$Z_{MS_{min}}(s) = \frac{1}{|N_1(s)|} MS_{min}(s')$$

where $MS_{min}(s')$ is the makespan of schedule s'. Laslo [47] considered the case of determining the machine booking schedule for a virtual job shop problem to maximize the economic gain from outsourced rented machines. Authors assumed that operations processing times are normally distributed, and hence proposed a heuristic based method.

Anglani[48] proposed a fuzzy mathematical model of scheduling parallel machines with sequence-dependent cost while considering uncertainties in processing times. The proposed approach requires the solution of a non-linear mixed integer programming (NLMIP), that can be formulated as an equivalent mixed integer linear programming (MILP) model. Due to its NP-hardness, the resulting MILP model in real applications could be intractable. Therefore, they proposed a solution method technique, based on the solution of an approximated model, whose dimension is remarkably reduced with respect to the original counterpart. Numerical experiments were conducted on the basis of data taken from a real application show that the average deviation of the reduced model solution over the optimum is less than 1.5%. Recently, Bouyahia [49] presented a probabilistic generalization to design robust a priori scheduling that assumes the number of jobs to be processed on parallel machines as a random variable with respect to the total weighted flow time.

Guo and Nonaka [50] studied how to reduce the effect of machine failure on a three-machine flow shop by proposing a method to evaluate initial schedules (preschedules) and a rescheduling method that is applied after machine failure.

Matsveichuk [51] proposed a two-stage scheduling decision framework to execute schedules of a two-machine flow shop in which each uncertain processing time of a job on a machine may take any value between a lower and upper bound. With an objective to minimize the makespan there were two phases in the scheduling process: offline (the schedule planning phase) and online (the schedule execution phase). The information of the lower and upper bound for each uncertain processing time was available at the beginning of the off-line phase while the local information on the realization (the actual value) of each uncertain processing time was available once the corresponding operation (of a job on a machine) was completed. In the off-line phase, a minimal set of dominant schedules were prepared by a scheduler, which was derived based on a set of sufficient conditions for schedule domination. This set of dominant schedules enabled a scheduler not only to quickly make an on-line scheduling decision but also to optimally cover all feasible realizations of the uncertain processing times. The approach proposed in the research enabled the scheduler to best execute a schedule and may end up with executing the schedule optimally in many instances according to our extensive computational experiments which are based on randomly generated data up to 1000 jobs.

Qi [52] introduced a rescheduling approach for single and parallel two-machine environment subjected to random machine unavailability and processing time variations. The approach considered in this work is different from most of the rescheduling analysis in that the cost associated with the deviation between the original and the new schedule was included in the model. The research focused on cases in which the shortest processing time (SPT) rule is optimal for the original problem considering both single and parallel two-machine environments.

Most of the research work discussed above only deals with scheduling in different type of shop environments. But the extent of its application is not only bound in only such kind of problems. The quest of finding an optimal schedule inspires the researcher to deal with problem in other areas also. Artigues [53] proposed insertion techniques for static and dynamic resource-constrained project scheduling.Surico [54]

suggested a hybrid meta-heuristic that integrates a mathematical programming, multi-objective evolutionary computation (genetic algorithm), and a problem-specific constructive heuristic that returns a number of solutions or the pareto sets (schedules), each with a cost and risk trade-offs, for the problem of Supply Network (SN) for ready-mixed concrete (RMC). Chtourou and Haouari [55] presented a two-stage algorithm to produce robust resource-constrained project scheduling subjected to unpredictable increase in processing times. Lambrechts [56] proposed a tabu search algorithm that uses a free slack-based objective function to produce robust predictive-reactive project schedules in the presence of uncertain renewable resource availabilities.

Rangsaritratsamee [57] proposed a rescheduling method based on local search genetic algorithm for a job shop with dynamically arriving jobs. Their proposed algorithm simultaneously considers the efficiency by preserving the makespan, tardiness and stability by minimizing the jobs starting time deviations. In their work, the rescheduling takes place at specific time intervals using all available jobs at the rescheduling moment. Fattahi and Fallahi [58] combined the work of Rangsaritratsamee [57] and Fattahi [59] and developed a multi-objective genetic algorithm based method to scheduling a flexible job shop with dynamically arriving jobs. A multi-objective mathematical model was developed to make a balance between efficiency and stability of the schedules.

Mahdavi [60] presented a real-time simulation-based decision support system to control the production of a stochastic flexible job shop subjected to stochastic processing times. Based on the theory of supervisory control, a controller was developed which constitutes the framework of an adaptive controller supporting the co-ordination and co-operation relations by integrating a real-time simulator and a rule-based DSS. A bilateral method for multi-performance criteria optimization was implemented to combine the gradient based method and the DSS to control dynamic state variables of SFJS concurrently.

Vinod and Sridharan [61] experimentally studied different scheduling decision rules for scheduling dynamic flexible job shop (jobs arrive intermittently) using a discrete simulation based model. They considered a partial flexible job shop system. The system consists of eight machines wherein an operation can be executed on three

different ones at different rates. Three scheduling rules were used for machine selection decision and a total of 15 scheduling rules were incorporated in the simulation model for job scheduling decisions. Moreover, six new scheduling rules for job scheduling were also developed and investigated. The performance measures evaluated in this research work are the mean flow time, standard deviation of flow time, mean tardiness, standard deviation of tardiness and percentage of tardy jobs. It was showed from the experimental simulation that, the proposed scheduling rules provide better overall performance for the various measures when compared with the existing scheduling rules. Vinod and Sridharan [62] continued the previous study by studying the interaction between due-date assignment methods and scheduling rules in a dynamic job shop system using a discrete-event simulation model. Their simulation analysis showed that due-date assignment methods and the used scheduling rules are significantly affecting the performance measures of the shop.

For a recent overview discussing aspects of scheduling with uncertainties readers are referred to Davenport and Beck [63], Aytug [64], Herroelen and Leus [65], and Mula [66], who gave detailed review of literature related to scheduling under uncertainty.

In light of the above literature, approaches used to achieve schedule robustness are classified into two categories, preservation of solution quality approach and execution-oriented quality approach. In the first, robustness is considered as the ability to preserve some level of solution quality, such as preservation of makespan in Leon [41] and Jensen [45], or preservation of tardiness and total flow-time as in Jensen [11], etc. In the later, also known as rolling time approach, robustness is achieved by producing partial schedules and the final decisions are delayed until the execution time is reached or nearly reached as in Kutanoglu and Wu [33, 34], and Policella [39]. Hence, two definitions for a robust schedule can be distinguished:

1. A schedule is considered to be robust if it has low cost relative to other schedule when facing disruption and when rightshifting is used as a rescheduling algorithm (Jensen, [45])

2. A schedule is considered to be robust if it can absorb the external events (disruptions) without loss of consistency while keeping the pace of execution

(Policella, [39]), i.e. without amplifying the effects of a change over all schedule components.

Despite of the apparent similarity between the two definitions, it can be concluded that the first definition is more suitable for offline scheduling such as predictive scheduling, whereas the second definition is falling in some category belonging to dynamic scheduling and more precisely to proactive/reactive (or predictive/reactive) scheduling and knowledge-based scheduling. Furthermore, the second definition has some resemblance with the definitions of stable and/or flexible schedule found in literature. In Cavalieri and Terzi [67] and Policella [39, 40] flexibility was defined as ‘‘the ability to respond effectively to changing circumstances’’. A more thorough definition of flexibility was given by Jensen [68] as “a schedule expected to perform well relative to other schedules, when facing disruption and when some rescheduling method using search (other than right-shift) is used”. Similarly, Goren [69] , Liu [30], and Wu [70] defined stable schedule as a schedule that has a very small deviation either in time or sequence between the predicted schedule and the realized schedule. At this point one may conclude that the two types of schedules named stable schedule and flexible schedule used in literature are actually describing the same schedule. This schedule (stable or flexible) can be related to the system (or schedule) nervousness measure, i.e. if the performance measure of the schedule nervousness is high then the stability of the schedule is low (representing an unstable manufacturing system) and vice versa. Furthermore, a schedule is called robust or stable depending on how it was designed to adapt to changes and unforeseen future events.

Furthermore, it can be observed that unlike single machine environment, two machines environment, and job shop environment, vast majority of the literature published in the area of stochastic scheduling gives less attention to the FJSP. The literature focuses either on deterministic FJSP, stochastic classical job shop scheduling problem (JSP) or dynamic FJSP. The literature on robust and stable scheduling for the FJSP under unpredicted arrival of new jobs is almost void. Therefore, the goal of this work is to improve robustness and stability of predictive schedule for the FJSP subjected to unpredicted arrival of new jobs. This research work introduces a new methodology that measures the variability of two schedules. The proposed methodology is based on a genetic algorithm and branch and bound

algorithm. Moreover, the current work relates the robustness of a schedule to its degree of makespan degradation under disruptions and considers it to be stable when its sum of the absolute deviations of operation completion times from the realized schedule is small.

CHAPTER III

MODEL FORMULATION

3.1 Problem Identification:

A basic assumption during the Job Shop Scheduling is that the processing times are deterministic and the situations during processing are stable. But in most of the cases processing time is not deterministic rather it is stochastic. Moreover, situations at the job shop are not stable which are significantly affected by different factors and more prone to disruption. So in real scenario, a job shop schedule will be more adaptive to the real scenario if it can anticipate such disruption. In the literature review it has been observed that most of the authors addressed machine breakdown as a major source of disruption. This is true that machine breakdown occurs frequently during operation. However, it can be minimized by properly scheduling the maintenance activities. Some works have already been discussed in the previous chapters where the authors considered the maintenance activities. Very few papers addressed unpredicted arrival of new job as a source of disruption. But it has a significant effect on job shop scheduling. It is not like one can schedule a set of jobs, start processing and after completion start another set of jobs. Rather jobs continuously arrive in the job shop and need to be scheduled in the real time. On the other hand, it must be done in such a way that after the insertion of new job the existing schedule will not be affected significantly by this disruption. Therefore, generating a job shop schedule which is robust, stable and reactive to the disruption plays a significant role in flexible job shop scheduling.

3.2 Problem Definition:

The purpose of this thesis work is to extend the previous research in developing the flexible job shop scheduling model by incorporating the unexpected arrival of new jobs. This study introduces a two stage scheduling method. In the first stage an initial sequence should be determined. In fact, it is initially assumed that there is no random disruption in the system and an initial solution for scheduling considering the problem with uncertain processing times will be generated. The robust optimization approach is used to generate a robust initial schedule. Actually to reduce the effect of

uncertainty on the processing times which is a random disruption in the future, first a job shop model is formulated using the robust optimization approach as an attempt to produce robust initial solutions. This is the proactive first stage. After that the initially robust schedule is determined, it is assumed that a new job arrives during the execution of the initial schedule. No predictive information exists for this job. In the dynamic environment of this system, this unpredicted arrival is an issue related to real time scheduling. In fact, the unpredicted arrival of a new job is considered as an unexpected disruption. Considering the new job, the proposed method adopts an appropriate reactive action in order to determine its operations position at different machines in the initial sequence. If all operations of this job are processed at the end, the arrangement of schedule does not change. However, if the operations processed at a position other than the last, the arrangement of previous operations at different machines will change. In fact, finding a proper position for the processing of the new operations plays an important role in the optimization of the job shop scheduling problem. Although changing the system's previous sequence may produce a better solution, it may also reduce the stability and cause disturbance in the system. To adopt a reactive response, a new objective function as a measure based on a classical objective and performance measures is defined. This measure helps planners to choose the most appropriate reaction to counter the effect of arrival of new jobs.

3.2.1 Proactive Scheduling step:

In the proactive scheduling step the problem with uncertain processing times that are estimated with scenarios is just considered. A robust optimization approach is used to formulate the flexible shop problem to reduce the effects of fluctuations of the processing times in the future. In the first steps proactively a more robust solution as an initial schedule is generated. After that the model is solved by using metaheuristics.

3.2.1.1 Assumptions of the study:

I. Job j has n_j operations that must be processed according to the predefined sequence.

II. The operations $i \epsilon N_j = [71\ldots, nj]$ of job $j \epsilon J$ are non preemptive, once started the operation must be completed

III. The execution of operation i of job j requires a machine selected from a subset M_{ij} subset of K of available machines
IV. The transportation time between the machines are neglected
V. Processing time at different scenarios are deterministic and include setup, operations, transportation and inspection
VI. Any kind of disruptions accept arrival of unexpected new jobs are not considered such as machine breakdown, order cancellation etc.
VII. All machines are always available in the entire time horizon
VIII. Jobs are independent and no priorities are assigned
IX. Each machine can only process one operation at a time
X. Each operation can be processed by one machine at a time
XI. All jobs are inspected priori i.e., no defective part is considered
XII. Probability of occurring a scenario is known priori

3.2.1.2 Mathematical Modeling:

In this research, a new mathematical model for robust flexible job shop scheduling is proposed. The model is developed to determine the optimal sequence of operations at different machines. For robust optimization, a model is formulated to minimize the makespan of the total schedule along with the variability of makespan at different scenarios. Necessary constraints are developed to ensure the model is bounded one.

Before proceeding to the mathematical model, some sets, parameters and variables of the model are introduced in the following:

Sets:

$N_j = set\ of\ all\ operations\ of\ job\ j, i \in N_j$

$N_{jt} = set\ of\ incomplete\ operations\ of\ job\ j\ at\ time\ t$

$J = set\ of\ all\ jobs, j \in J$

$J_t = set\ of\ incomplete\ jobs\ at\ time\ t, j \in J_t is\ subset\ of\ J$

$K = set\ of\ all\ machines\ k \in K$

$M_{j,i}, (M_{l,h}) = set\ of\ machines\ allowed\ to\ process\ operation\ i\ (h)\ of\ job\ j\ (l)$

$\Omega = set\ of\ all\ scenarios$

Indices:

$i, h = indices\ for\ operation\ of\ job$

$j, l = indices\ for\ job$

$k = indices\ for\ machine$

$\lambda, \lambda' = indices\ for\ scenarios$

Parameters

$t = time\ of\ the\ arrival\ of\ the\ unpredicted\ job$

$n_j = total\ number\ of\ operations\ of\ job\ j$

$n = total\ number\ of\ jobs$

$m = total\ no\ of\ machines$

$P_{j,i,k}^{\lambda} = processing\ time\ of\ operation\ i\ of\ job\ j\ in\ machine\ k\ in\ scenario\ \lambda$

$p^{\lambda} = probability\ of\ occurring\ scenario\ \lambda$

$M = a\ sufficiently\ large\ number$

$a_k = time\ at\ which\ resource\ k\ is\ available\ for\ starting\ the\ next\ operation$

$\alpha, (\beta), \{\gamma\}s = objective\ weight\ of\ makespan, (stability), \{robustness\}$

Variables:

$C_{j,i}^{\lambda} = completion\ time\ of\ operation\ i\ of\ job\ j\ in\ scenario\ \lambda$

$C_{max} = Makespan\ of\ all\ jobs\ or\ maximum\ completion\ time$

$$X_{j,i,k} = \begin{cases} 1, if\ operation\ i\ of\ job\ j\ is\ allocated\ to\ resource\ k \\ 0, otherwise \end{cases}$$

$$X_{j,i,l,h} = \begin{cases} 1, if\ operation\ i\ of\ job\ j\ is\ processed\ after\ operation\ h\ of\ job\ l \\ 0, otherwise \end{cases}$$

$$e_{j,i,k} = \begin{cases} 1, if\ operation\ i\ of\ job\ j\ is\ allowed\ \ to\ be\ processed\ in\ machine\ k \\ 0, otherwise \end{cases}$$

Now using this parameters and variables the developed flexible job shop scheduling model is presented as follows

***3.2.1.2.1 Objective function for robust scheduling*:**

A robust optimization approach is adopted to minimize the makespan of the total system as an objective function. The goal of the robust optimization approach is to get a set of solutions for the problem so that they remain robust despite changes that may occur in the real values of data and input parameters. In robust optimization, the uncertain parameters are described by the discrete scenarios or a continuous range. Therefore, variability of makespan between different scenarios is minimized. So that, the model can anticipates the changes in future with a very little effect on the existing schedule. The goal of this optimization method is obtaining an optimal solution, which is insensitive to almost all the samples of the uncertain parameters. The objective function described above can be written as follows where the first part is considered to minimize the makespan and the second part is considered to minimize the variability of makespan in different situations.

$$Min, \quad \sum_{\lambda \in \Omega} {}^{\lambda}p C_{max}^{\lambda} + \delta \sum_{\lambda \in \Omega} {}^{\lambda}p \left| C_{max}^{\lambda} - \sum_{\lambda \in \Omega} {}^{\lambda}p C_{max}^{\lambda} \right| \qquad (31)$$

3.2.1.2.2 Constraints for assigning and sequencing of operations to available processing positions:

In the assumption it has already been stated that the execution of operation i of job j requires a machine selected from a subset M_{ji} subset of K of available machines. If the job shop is full flexible then M_{ji} includes all available machines. To ensure that all operations are investigated for processing on all available machines and eventually

are imperatively assigned to one of them it is inevitable to develop a set of constraints. These set of constraints can be written as follows.

$$\sum_{k=1}^{m} X_{j,i,k} = 1 \qquad \forall\, i, j \tag{32}$$

For all i and j combinations all machines will be checked first. As $X_{j,i,k}$ is a binary decision variable and for a given i and j combination summation for all k is 1, only one variable can have the value 1. So these set of constraints satisfy the assumption made earlier that, each operation can be processed by one machine at a time. In the flexible job shop scheduling these constraints provides the routing flexibility

3.2.1.2.3 Machine eligibility constraints:

In flexible job shop scheduling there are two kinds of machine flexibility. One is total flexible and the other one is partial flexible. In the first case all machines are allowed to do all the operation. On the other hand, in partial flexible system only a set of machines are allowed to do an operation. So it is necessary to check in both cases whether the machine is eligible to process an operation or not and subsequently assign and operation to the eligible one. This gives the rise of machine eligibility constraints. The constraint set can be written as follows,

$$X_{j,i,k} \leq e_{j,i,k} \qquad \forall\, i, j, k \tag{33}$$

Where $e_{j,i,k}$ is a binary decision variable which takes value 1 if machine k is allowed to process operation i of job j otherwise zero. As only one $X_{j,i,k}$ can have value 1, it is ensured that it will allowed to be processed in an eligible machine. Therefore, this constraint set ensures the feasibility of the machine assignments which are investigated for any of the operations in previous constraint set.

3.2.1.2.4 Technical/ logical precedence constraints among operations of a part:

In a job shop all jobs must be processed in a predefined sequence of operation. So it is necessary to develop some constraints to avoid simultaneous assignment of multiple operations of a job. The following set of constraint represents the logical/natural precedence constraint among the operations of a job. It simply means that so long as

the previous operation of a job has not been completed on any of the machines in the shop floor the succeeding operation is not processed.

$$C_{j,i}^{\lambda} \geq C_{j,i-1}^{\lambda} + \sum_{k=1}^{m} X_{j,i,k} P_{j,i,k}^{\lambda} \qquad \forall\, i, j, \lambda \tag{34}$$

$$C_{j,1}^{\lambda} \geq \sum_{k=1}^{m} X_{j,1,k} P_{j,1,k}^{\lambda} \qquad \forall\, j, \lambda \tag{35}$$

Here, $C_{j,i}^{\lambda}$ is the completion time of operation i of job j in scenario λ and $C_{j,i-1}^{\lambda}$ is the completion time of the previous operation as well. So the difference between the completion times of the two consecutive operations should be at least equal to the processing time of the first operation between them. The difference may be greater depending on the position of the operation in the schedule. This rule is established by the set of constraints (3.4). The first operation has no precedence constraints. So the completion time must be at least greater than or equal to the processing time of the first operation at the machine in which it will be assigned.

3.2.1.2.5 Machine availability constraints:

It is assumed that all machines at available at time zero which is the case in the first stage of optimization. But after time t while an unpredicted job will come at shop floor machines may not be available at that time. There may be some operation still carried on by few machines. As preemption is not allowed, an operation must be completed once it's started which give the rise of machine availability constraints. It can be written as follows.

$$C_{j,i}^{\lambda} - X_{j,i,k} P_{j,i,k}^{\lambda} \geq a_k \qquad \forall\, i, j, k, \lambda \tag{36}$$

Here $a_k = 0$ for the first stage of optimization but it will not be the case in the second stage of operation. In the second stage $a_k \geq t$

3.2.1.2.6 Machine non-interference constraints:

Though constraint set has already been developed to avoid simultaneous assignment of operation of the same job, it may possible that operations of multiple jobs are

allocated to the same machine at the same time. So constraint set must be established to avoid simultaneous assignment of operations of different jobs concurrently at the same machine. The constraint set can be written as follows.

$$C_{j,i}^{\lambda} \geq C_{l,h}^{\lambda} + P_{j,i,k}^{\lambda} - M\left(3 - X_{j,i,l,h} - X_{j,i,k} - X_{l,h,k}\right) \ \forall\, j \leq n, l \leq n(j,i) \neq (l,h), k, \lambda \quad (37)$$

$$C_{l,h}^{\lambda} \geq C_{j,i}^{\lambda} + P_{l,h,k}^{\lambda} - \left(X_{j,i,l,h} + 2 - X_{j,i,k} - X_{l,h,k}\right) \quad \forall\, j \leq n, l \leq n(j,i) \neq (l,h), k, \lambda \quad (38)$$

Here Constraint sets (3.7) and (3.8) referred to as Either-Or constraints simultaneously ensure the following: 1) an operation cannot be at the same time both the predecessor and the successor of another operation, and 2) satisfaction of non-interference constraints (precedence constraint among operations of different jobs); i.e., for operations of different jobs that are eligible to be processed on the same machine. In other words, if two operations of two different jobs do not share the same subset of machines, consideration of the operational precedence between them is not done and both constraints become redundant. Hence, two operations $O_{j,i}$ and $O_{l,h}$ can only be sequenced when both their binary integer variables assignment $X_{j,i,k}$ and $X_{l,h,k}$ take value of 1; otherwise, they bear no relationship with one another on machine k. Once machine k was established as eligible machine to process $O_{j,i}$ and $O_{l,h}$ ($X_{j,i,k} = 1$ and $X_{l,h,k} = 1$) the precedence relationship between these two operations should be decided by the sequencing binary decision variable which is $X_{j,i,l,h}$. In any circumstance other than stated above, both constraint sets (3.7) and (3.8) become redundant constraints. If $O_{j,i}$ is processed after $O_{l,h}$ on machine k, the sequencing binary decision variable takes value 1 and therefore constraint set (3.8) become active. Otherwise, this constraint show that $C_{l,h}$ is just greater than a large negative number, which is naturally true. The sequencing binary variable ($X_{j,i,l,h}$) takes value 0 if $O_{j,i}$ is not processed after $O_{l,h}$. Since two operations in sequencing decision have no more than two states with respect to each other (predecessor or successor), if $O_{j,i}$ does not succeed $O_{l,h}$, it has to precede it, in which case constraint set (3.7) become active and constraint set (3.8) become evident inequality equations.

3.2.1.2.7 Constraints for capturing the value of objective function:

$$C_{max}^{\lambda} \geq C_{j,n_j}^{\lambda} \qquad \forall\, j, \lambda \qquad (39)$$

Constraint set (3.9) keeps track of make-span and compute it.

3.2.1.2.8 Non-negativity constraints:

$$C_{j,i}^{\lambda} \geq 0 \qquad (310)$$

Constraint set (3.10) shows the non-negativity nature of the MILP'S continuous variables.

3.2.1.2.9 Constraints for demonstrating the nature of the decision variables:

$$X_{j,i,k}\,,\ X_{j,i,l,h}\,,e_{j,i,k} \in \{0,1\} \qquad \forall\, i,j,h,l,k \qquad (311)$$

Constraint set (3.10) demonstrates the binary nature of decision variables.

So taking objective function and all the constraints in consideration, the mathematical model for the first stage is as follows:

Objective function:

$$Min, \quad \sum_{\lambda \in \Omega} {}^{\lambda}p C_{max}^{\lambda} + \delta \sum_{\lambda \in \Omega} {}^{\lambda}p \left| C_{max}^{\lambda} - \sum_{\lambda \in \Omega} {}^{\lambda}p C_{max}^{\lambda} \right| \qquad (31)$$

Subject to,

$$\sum_{k=1}^{m} X_{j,i,k} = 1 \qquad \forall\, i,j \qquad (32)$$

$$X_{j,i,k} \leq e_{j,i,k} \qquad \forall\, i,j,k \qquad (33)$$

$$C_{j,i}^{\lambda} \geq C_{j,i-1}^{\lambda} + \sum_{k=1}^{m} X_{j,i,k} P_{j,i,k}^{\lambda} \qquad \forall\, i,j,\lambda \qquad (34)$$

$$C_{j,1}^{\lambda} \geq \sum_{k=1}^{m} X_{j,1,k} P_{j,1,k}^{\lambda} \qquad \forall\, j,\lambda \qquad (35)$$

$$C_{j,i}^{\lambda} - X_{j,i,k} P_{j,i,k}^{\lambda} \geq a_k \qquad \forall\, i,j,k,\lambda \tag{36}$$

$$C_{j,i}^{\lambda} \geq C_{l,h}^{\lambda} + P_{j,i,k}^{\lambda} - M\left(3 - X_{j,i,l,h} - X_{j,i,k} - X_{l,h,k}\right) \;\forall\, j \leq n, l \leq n (j,i) \neq (l,h), k, \lambda \tag{37}$$

$$C_{l,h}^{\lambda} \geq C_{j,i}^{\lambda} + P_{l,h,k}^{\lambda} - M\left(X_{j,i,l,h} + 2 - X_{j,i,k} - X_{l,h,k}\right) \;\forall\, j \leq n, l \leq n (j,i) \neq (l,h), k, \lambda \tag{38}$$

$$C_{max}^{\lambda} \geq C_{j,n_j}^{\lambda} \qquad \forall\, j,\lambda \tag{39}$$

$$C_{j,i}^{\lambda} \geq 0 \tag{310}$$

$$X_{j,i,k}\,,\; X_{j,i,l,h}\,,\; e_{j,i,k} \in \{0,1\} \qquad \forall\, i,j,h,l,k \tag{311}$$

3.2.1.3 Reactive scheduling step

In the first stage which is called the proactive stage an initial schedule will be generated. After the execution of this schedule, if any unpredicted job will arrive at the shop the initial schedule may be infeasible. So that a second stage of optimization is necessary. This stage is called the reactive stage as it will be executed after the arrival of new job. The objective of this stage is to generate a new schedule with minimal change in the initial one. The mathematical model is almost similar to the first stage accept the objective function and some changes in the sets that will be considered for the development of the constraints. The mathematical model of the reactive stage is as follows:

Objective function:

$$Min, \quad \alpha\, C_{max}^{\lambda'} + \beta \sum_{i \in N_{j_t}} \sum_{j \in J_t} \left| C_{j,i}^{\lambda} - C_{j,i}^{\lambda'} \right| + \gamma \left| C_{max}^{\lambda} - C_{max}^{\lambda'} \right| \tag{312}$$

Subject to,

$$\sum_{k=1}^{m} X_{i,j,k} = 1 \qquad i \in N_{j_t}, j \in J_t \tag{32a}$$

$$X_{j,i,k} \leq e_{i,j,k} \qquad i \in N_{j_t}, j \in J_t \tag{33a}$$

$$C_{j,i}^{\lambda'} \geq C_{j,i-1}^{\lambda'} + \sum_{k=1}^{m} X_{j,i,k} P_{j,i,k}^{\lambda'} \qquad i \in N_{j_t}, j \in J_t, k \in K \qquad (34a)$$

$$C_{j,1}^{\lambda'} \geq \sum_{k=1}^{m} X_{j,1,k} P_{j,1,k}^{\lambda'} \qquad i \in N_{j_t}, j \in J_t, k \in K, \lambda \in \Omega \qquad (35a)$$

$$C_{j,i}^{\lambda'} - X_{j,i,k} P_{j,i,k}^{\lambda'} \geq a_k \qquad i \in N_{j_t}, j \in J_t, k \in K \qquad (36a)$$

$$C_{j,i}^{\lambda'} \geq C_{l,h}^{\lambda'} + P_{j,i,k}^{\lambda'} - M(3 - X_{j,i,l,h} - X_{j,i,k} - X_{l,h,k}) \qquad (37a)$$

$$i,h \in N_{j_t}, j \in J_t, (j,i) \neq (l,h)\ k \in K$$

$$C_{l,h}^{\lambda'} \geq C_{j,i}^{\lambda'} + P_{l,h,k}^{\lambda'} - M(X_{j,i,l,h} + 2 - X_{j,i,k} - X_{l,h,k}) \qquad (38a)$$

$$i,h \in N_{j_t}, j \in J_t, (j,i) \neq (l,h)\ k \in K$$

$$C_{max}^{\lambda'} \geq C_{j,n_j}^{\lambda'} \qquad \forall j \qquad (39a)$$

$$C_{j,i}^{\lambda'} \geq 0 \qquad (310a)$$

$$X_{j,i,k}, X_{j,i,l,h}, e_{j,i,k} \in \{0,1\} \qquad i,h \in N_{j_t}; j,l \in J_t; k \in K \qquad (311a)$$

Here the objective function consists of three parts. The first part contains the main objective which is minimization of makespan of all jobs. The second part is for stability where the absolute deviation of the completion time of all operation between initial schedule and the new schedule is minimized. The third part is for robustness where the absolute deviation of the makespan between two stages is minimized. Though the third portion of the objective function make the model robust but it may not be stable without the second one. Minimizing the deviation of the completion time restrict the generation of new schedule in such way so that the change of position of operation will be minimal. Thus makes the model reactive, robust and stable.

In the constraints there are few changes made in this stage. First, the set of jobs; only remaining jobs are considered for schedule in the next stage (i.e.$i \in N_{j_t}$). Second, the

set of operations; though only remaining jobs are considered, all operations may not need to be scheduled again because some operations may be already completed or may be in processing while the new job arrives (i.e. $j \in J_t$).

CHAPTER IV

FLEXIBLE JOB SHOP SCHEDULING USING BRANCH AND CUT METHOD

Branch and Bound is the most popular and successful computational approach to integer programming problems today. Rather than being a specific algorithm, branch and bound is a general principle that allows the user to fine tune the procedure and adjust it to the problem under consideration. The branch and bound principle was first suggested by Land and Doig [72], followed by Dakin [73] and Balas's [74] additive algorithm for zero-one problems. Some more recent and comprehensive books are those by Lee [75], Mitchell [76], Rafael and Gerhard [77] and Androulakis [78]. In the next section a detail description of Branch and Bound and Branch and Cut algorithm is given

4.1 Branch and Bound Algorithm:

Branch and bound is a global optimization technique which is an exhaustive search process. To reduce the search space for achieving the result within reasonable timeframe some techniques are applied which improve the process of finding the optimal solution. One of these techniques is to relax a mathematical optimization problem by dropping some of its constraints, so that the resulting problem has a larger feasible region and a solution that cannot be worse than that of the original problem. Another technique is to add constraints to a problem, a process that reduces the size of the feasible region, so that the optimal objective value deteriorates. Branch and bound methods can also be viewed as cutting plane methods with an added feature that divides the feasible region into smaller parts which are then dealt with separately. Each successive cut results in a linear programming problem that is represented by a node in a graph, where the arcs correspond to the cuts. The graph is developed and explored as the cuts are successively applied, with the initial node n_1 corresponding to the relaxation of the problem without any additional constraints. By construction, the graph will be a tree, and if we regard its arcs as directed corresponding to the order in which the cuts are applied, we have arborescence with the initial node n_1 as its root. As the process is directed from n_1, to nodes representing problems with additional

cuts, the proper term would be "branch and bound arborescence." However, the common practice and use is the term "branch and bound tree." In a branch and bound tree, the initial node n_1, is usually referred to as the *top node.*

4.1.1 Search Strategies:

Since each node of the solution tree represents a linear programming problem that must be solved in the process, it is obviously important to keep the size of the tree as small as possible. This is accomplished by stopping at a node whose associated problem has an integral solution, is infeasible, or is fathomed (value dominated). A good branch and bound method will bring about such favorable situations as early as possible in the search. If each path is pursued until integrality or infeasibility was reached, the strategy is called a *depth-first search* strategy. The *depth* (or *level)* of a node is the length of the path leading to it from the top node. In general, a depth first strategy will pursue each node, using some branching strategy, until integrality, infeasibility, or fathoming has occurred, and then backtrack until the first node is encountered, which can be further pursued. This strategy will bring a high level (deep in the tree) early in the search. By contrast, a *breadth-first search* strategy explores all nodes at one level before proceeding to the next. The breadth-first search strategy was much better than the depth-first search strategy in the sense that the tree is much smaller. Figure 4.1 illustrates these two strategies.

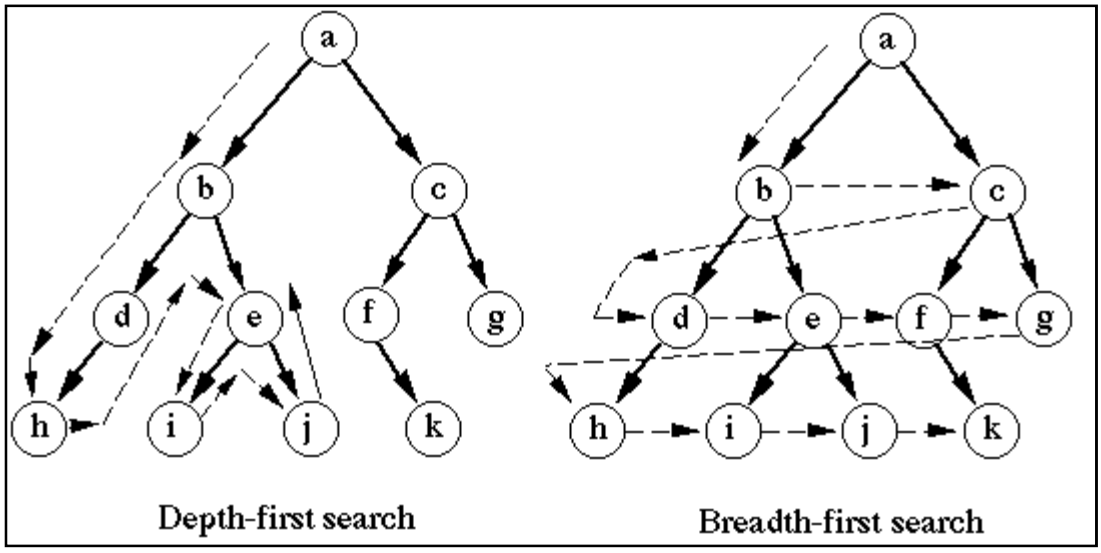


Figure 4.1: Depth-first search vs. Breadth-first search strategy

In practice, however, it appears that depth-first strategies are more advantageous than breadth-first. One reason for this is that the number of nodes grows exponentially as the level increases, and since feasible (integer) solutions are typically found deeper into the tree, depth-first search tends to find integer solutions sooner than breadth-first search. Another appealing feature of depth-first search is related to the way in which the linear programming problems at each node are solved. Assuming that the simplex method is used for solving the problems, an immediate successor of a node has the same set of constraints as its predecessor, plus the one additional constraint generated in the branching. The construction of a branch and bound tree is guided by two decisions during each time a branching is done,

- which node to pursue next *(node selection strategy) ,* and
- which variable to branch on *(branch selection strategy).*

These two types of strategies are discussed below.

4.1.2 Node Selection:

A number of strategies exist for selecting the next node to branch from. For simplicity of the exposition, it is assumed that the original problem has a maximization objective. If so far in the search one or more nodes have been encountered for which the solution is integer, the one with the highest objective function value (ties are broken arbitrarily) is chosen which is called the *incumbent node* or solution. The collection of nodes that have no branches leading out of them forms the set S. Initially, only the top node is in S.

The nodes in the set S fall into four categories.

1. The solution at a node $n_i \in S$is integer, in which case we will not branch further from this node. If the solution is better than that of the existing incumbent(s) $n_k$then n_i, replaces n_k; if it is as good as n_k it becomes another incumbent.
2. The node $n_i \in S$ represents a problem that has no feasible solution, making branching on it pointless, since any successor problem will also lack feasible solutions.

3. The node n_i is fathomed, i.e., its non-integer solution is no better than the incumbent solution. Again, its potential successors have poorer solutions than the incumbent and are therefore not of interest.
4. The node $n_i$has a non-integer solution with an objective function value that is better than that of the incumbent, so that branching from this node might lead to an integer optimal solution. Such a node is called *live* or *active.*

In summary, out of the four possible node types--integer, infeasible, fathomed, and live, only branch from live nodes will be done. Defining $L \subset S$ as the set of live nodes, the node selection problem addresses the choice of a node $n_i \in L$.

The two node selection strategies have already described previously: the breadth-first strategy where all live nodes at a given level are considered before nodes on lower levels are examined, and the depth-first strategy where the next node to be considered is a live successor of the latest node that was explored. Since backtracking is needed if the node that was explored last is not live, the strategy is better described as "depth-first with backtracking"; or *last in, first out* (LIFO), borrowing a concept from inventory management. Although a depth-first strategy can be expected to perform better than a breadth-first strategy on average, both of these strategies would be outperformed, again on average, by strategies that make better use of the information gathered during the construction of the tree. One such strategy is the *best-bound-first* strategy that selects the live node with the largest z-value. Formally, the best-bound-first strategy selects the node n_k such that $n_k = z_k = \max\{z_l : n_l \in L\}$

Another node selection strategy involves the estimate $\hat{z}_l$ of the integer optimal solution corresponding to node n_l. Such an estimate can be computed based on the expected *degradation* expressing the deterioration of z_l by requiring the solution point at node n_l to be integral. More specifically, let the solution at node n_i include$\tilde{x}_j = \lfloor \tilde{x}_j \rfloor + f_j$ with $f_j \neq 0$Using some user selected coefficients $p_j^-$and p_j^+ we estimate the decrease in the objective function of $p_j^- f_j$ for branching left at node n_i and of $p_j^+(1 - f_j)$ if branching right. The coefficients p_j^- and p_j^+ can either be user-specified or estimated, e.g., by using dual information at node n_i or information from previous branchings on x_j. A *best-estimate* search strategy selects the node n_k such that$\hat{z}_k = \max\{\hat{z}_l : n_l \in L\}$. Denoting by $\underline{z}_{IP}$ the objective function

value of the incumbent solution, the *quick-improvement* strategy attempts to quickly improve on the incumbent solution by selecting the node n_k such that $k = \arg max\{(z_l - \underline{z}_{IP})/(z_l - \hat{z}_l)\}$. Detail discussion and further details on node selection are available in Nemhauser and Wolsey [79].

4.1.3 Branch Selection:

Once a node $n_k \in L$ has been selected for further exploration, the next decision concerns the choice of variable to be branched on. Clearly, this variable, while required to be integer, must currently have a non-integer value. One simple strategy is branching the non-integer variable with lowest index, an obviously arbitrary rule. Another strategy is to branch on the non-integer variable x_k with the "most fractional" value. Formally, let $\tilde{\boldsymbol{x}} = [\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \dots, \tilde{x}_n]$ be the solution at the chosen live node, and let $\tilde{x}_j = \lfloor \tilde{x}_j \rfloor + f_j$, so that f_j denotes the *fractional* part of x_j. The *most fractional* strategy would branch on the variable whose present value is farthest from the nearest integer or, equivalently, has the value closest to $1/2$, i.e., a variable x_k with $k = \arg\min_{j=1,\dots,n}\{|f_j - 1/2|\}$. Unfortunately, experience has failed to identify robust methods for branch selection, and in practice user-specified priorities are employed. More involved methods that employ penalties and use more elaborate computations regarding the penalty coefficients p_j^- and p_j^+; have not turned out to improve the overall efficiency of the search if the additional computational effort it takes to apply them is taken into account.

4.1.4 A General Branch and Bound Procedure:

To formulate a general branch and bound algorithm an all-integer or mixed-integer programming problem P_{IP} with a maximization objective is considered; its linear programming relaxation is called P_{LP}. It is assumed that specific node and branch selection strategies have been chosen. The algorithm is initialized with node n_1 that includes the optimal linear programming relaxation $\overline{\boldsymbol{x}}_{\boldsymbol{LP}}$ with objective value $\overline{\boldsymbol{z}}_{\boldsymbol{IP}}$. Given that $\overline{\boldsymbol{x}}_{\boldsymbol{LP}}$ does not satisfy all of the integrality conditions (otherwise $\overline{\boldsymbol{x}}_{\boldsymbol{LP}} = \overline{\boldsymbol{x}}_{\boldsymbol{IP}}$= i IP is optimal for the (mixed) integer programming problem as well), set $S := L := \{n_1\}, t := 1$, and $z := -\infty$

A General Branch and Bound Algorithm

Step 1: Is $L = \emptyset$? If yes: If $\underline{z} > -\infty$, then the solution $\bar{\mathbf{x}}_{IP} = \underline{\mathbf{x}}$ with objective value $\bar{z}_{LP} = \underline{z}$ is optimal, otherwise the integer programming problem has no feasible solution.

If no: Go to Step 2.

Step 2: Choose some node $n_i \in L$ (according to some prescribed criterion) with solution $\mathbf{x}^i$ and objective value z_i.

Step 3: Branch on some variable x_j (according to some prescribed criterion) to the nodes n_{t+1} (with $x_j \leq \lfloor x_j \rfloor$) and n_{t+2} (with $x_j \geq \lceil x_j \rceil$). set $L := L \setminus \{n_i\}$, and $S := S \cup \{n_{t+1}, n_{t+2}\} \setminus \{n_i\}$. Solve the linear programming problems at n_{t+1} and n_{t+2}. The results are solutions $\mathbf{x}^{t+1}$ and $\mathbf{x}^{t+2}$ with objective values z_{t+1} and z_{t+2}, respectively.

Step 4: Is $\mathbf{x}^{t+1}$ feasible? If yes: Go to Step 5.

If no: Go to Step 7.

Step 5: Does $\mathbf{x}^{t+1}$ satisfy the integrality requirements of the original problem?

If yes: Go to Step 6.

If no: Set $L := L \cup \{n_{t+1}\}$ and go to Step 7.

Step 6: Is $z_{t+1} > \underline{z}$? If yes: Set $\underline{\mathbf{x}} := \mathbf{x}^{t+1}$, $\underline{z} := z_{t+1}$, and go to Step 7.

If no: Go to Step 7.

Step 7: Repeat Steps 4-6 with $t + 2$ instead of $t + 1$; then set $t := t + 2$ and go to Step 1.

Figure 4.2: Branch and Bound algorithm, (Eiselt [80])

The key to the algorithm is the updating procedure of the sets S and *L*. In each step when the procedure branches from some node n_i to two nodes n_{t+1} and n_{t+2} the set of end nodes S is updated to include the new nodes n_{t+1} and n_{t+2} and to exclude n_i. From the set of live nodes *L,* the node from which the branching takes place is deleted in Step 3, and node $n_{t+1}(or\ n_{t+2})$is added to the set in Step 5 only if its solution is feasible (Step 4) but not yet integer (Step 5).

There is an interesting analogy between cutting plane algorithms and branch and bound methods. One may view the branching constraints as vertical or horizontal cutting planes designed to cut off areas of the feasible region that do not contain any integer points. It is also possible to mix in regular cutting planes with the branching at the nodes of a branch and bound tree. This approach is called *branch and cut,* and is

typically used for zero-one problems as well as problems with special structures. The idea is to find valid inequalities for the original problem, which are violated at some nodes in a branch and bound tree. These valid inequalities are then added at these nodes in a cutting plane fashion, thus generating new nodes from which branching can be done as usual. This could be accomplished by introducing an additional step between the existing Steps 2 and 3 in the general branch and bound algorithm above. In the next section branch and cut algorithm is discussed in detail.

4.2 Branch and Cut:

Branch- and- cut methods are very successful techniques f or solving a wide variety of integer programming problems, and they can provide a guarantee of optimality. Many combinatorial optimization problems can be formulated as mixed integer linear programming problems. They can then be solved by branch- and- cut methods, which are exact algorithm s consisting of a combination of a cutting plane method with a branch-and-bound algorithm. These methods work by solving a sequence of linear programming relaxations of the integer programming problem. Cutting plane methods improve the relaxation of the problem to more closely approximate the integer programming problem, and branch-and-bound algorithms proceed by a sophisticated divide and conquer approach to solve problems.

Cutting plane algorithms for general integer programming problems were first proposed by Gomory [81]. Unfortunately, the cutting planes proposed by Gomory did not appear to be very strong, leading to slow convergence of these algorithms, so the algorithms were neglected for many years. The development of polyhedral theory and the consequent introduction of strong, problem specific cutting planes led to a resurgence of cutting plane methods in the 1980's, and cutting plane methods are now the method of choice for a wide variety of problems. Perhaps the best known branch-and-cut algorithms are those that have been used to solve the traveling salesman problem (TSP). This approach is able to solve and prove optimality of far larger instances than other methods. Two papers that describe some of this research and also serve as good introductions to the area of branch-and-cut algorithms are Grotschel and Holland [82]; Padberg and Rinaldi [83]. Branch-and-cut methods have also been used to solve other combinatorial optimization problems, again through the exploitation of strong cutting planes arising from polyhedral theory. Problems attacked recently with

cutting plane or branch-and-cut methods include the linear ordering problem, maximum cut problems, scheduling problems, network design problems, packing problems, the maximum satisfiability problem, biological and medical applications, and finding maximum planar sub-graphs.

Though it is difficult to solve the flexible job shop problem using exact algorithm, some research work has already been done in this field also. Branch and cut method was used by several researcher to solve the FJSP because the computational effort is less than the actual branch and bound method. For instance, Stecco [84] was showed a comparison between three different formulations of a production scheduling problem with sequence-dependent and time-dependent setup times on a single machine by using branch and cut algorithm. Gupta [85] was minimized the total elapsed time for nx3 flow shop by considering the effect of breakdown interval and the transportation time using Branch and Bound technique

It is usually not possible to efficiently solve a general integer programming problem using just a cutting plane approach, and it is therefore necessary to also branch, resulting in a branch-and-cut approach. A pure branch-and bound approach can be sped up considerably by the employment of a cutting plane scheme, either just at the top of the tree, or at every node of the tree, because the cutting planes lead to a considerable reduction in the size of the tree. For general problems, the specialized facets used when solving a specific combinatorial optimization problem are not available. Useful families of general inequalities include cuts based on knapsack problems (Crowder [86]), Gomory cutting planes (Gomory [81] ; Balas [87]), and lift and project cutting planes (Balas [88]). Cutting planes and polyhedral theory are discussed in more detail later.

Nemhauser and Wolsey [79] and Wolsey [89] provide excellent and detailed descriptions of cutting plane algorithms and the other material in this entry, as well as other aspects of integer programming. Schrijver [90] is an excellent source of additional material. One aspect of a branch-and-cut approach that should not be overlooked is that it can be used to provide bounds. In particular, if the problem is minimization, one can't be able to prove optimality; a lower bound on the optimal value can be deduced from the algorithm, which can be used to provide a guarantee on the distance from optimality. Therefore, for large and/ or hard problem s, branch

and-cut can be used in conjunction with heuristics or meta-heuristics to obtain a good (possibly optimal) solution and also to indicate how far from optimality this solution may be.

Let's consider the following mixed integer linear programming problem

$$\begin{aligned} &\min\ c^T x \\ &\text{subject to } Ax \leq b \qquad (ILP) \qquad (4.1) \\ &x \geq 0 \\ &x_i \qquad \text{integer, } i = 1,,p. \end{aligned}$$

as standard form, where x and c are n-vectors, b is an m-vector, and A is an m × n matrix. The first p variables are restricted to be integer, and the remainder may be fractional. If $p = n$ then this is an integer programming problem. If a variable is restricted to take the values 0 or 1 then it is a binary variable. If all variables are binary then the problem is a binary program. There is no loss of generality with restricting attention to such a format. A branch-and-cut algorithm is outlined in Figure 4.3.

1. *Initialization*: Denote the initial integer programming problem by ILP^0 and set the active nodes to be $L = \{ILP^0\}$. Set the upper bound to be $\bar{z} = +\infty$. Set $\underline{z}_l = -\infty$ for the one problem $l \in L$.

2. *Termination*: If $L = \emptyset$, then the solution x^* which yielded the incumbent objective value $\bar{z}$ is optimal. If no such x^* exists (i.e., $\bar{z} = +\infty$) then ILP is infeasible.

3. *Problem selection*: Select and delete a problem ILP^l from L.

4. *Relaxation:* Solve the linear programming relaxation of ILP^l. If the relaxation is infeasible, set $\underline{z}_l = +\infty$ and go to Step 6. Let $\underline{z}_l$ denote the optimal objective value of the relaxation if it is finite and let x^{lR} be an optimal solution; otherwise set $\underline{z}_l = -\infty$.

5. *Add cutting planes*: If desired, search for cutting planes that are violated by x^{lR}; if any are found, add them to the relaxation and return to Step 4.

6. *Fathoming and Pruning*:

   (a) If $\underline{z}_l \geq \bar{z}$ go to Step 2.

   (b) If $\underline{z}_l < \bar{z}$ and x^{lR} is integral feasible, update $\bar{z} = \underline{z}_l$, delete from L all problems with $\underline{z}_l \geq \bar{z}$, and go to Step 2.

7. *Partitioning*: Let $\{S^{lj}\}_{j=1}^{j=k}$ be a partition of the constraint set S^l of problem ILP^l. Add problems $\{\text{ILP}^{lj}\}_{j=1}^{j=k}$ to L, where ILP^{lj} is ILP^l with feasible region restricted to S^{lj} and $\underline{z}_{lj}$ for $j = 1, \ldots, k$ is set to the value of $\underline{z}_l$ for the parent problem l. Go to Step 2.

Figure 4.3: Branch and Cut Algorithm, (Mitchell [91])

Notice that L is the set of active nodes in the branch-and-cut tree. The value of the best known feasible point for (ILP) is $\bar{z}$, which provides an upper bound on the optimal value of (ILP). Further, $\underline{z}_l$ is a lower bound on the optimal value of the current sub-problem under consideration. The value of the LP relaxation of the sub-problem can be used to update $\underline{z}_l$. In some situations, a very large number of violated cutting planes are found in Step 5, in which case it is common to sort the cutting planes somehow (perhaps by violation), and add just a subset. The sub-problems formed in Step 7 are called child sub-problems, with the previous problem ILP^l being the parent sub-problem. Usually, the partitioning takes the form of a variable disjunction: $x_i \leq a$ versus $x_i \geq a + 1$ for some variable x_i and integer a, as in the example. Other choices are possible, and they are discussed more in the branch-and-bound section. The relaxations can be solved using any method for linear

programming problems. Typically, the initial relaxation is solved using the simplex method. Subsequent relaxations are solved using the dual simplex method, since the dual solution for the relaxation of the parent sub-problem is still feasible in the relaxation of the child sub-problem. Further, when cutting planes are added in Step 5, the current is still dual feasible, so again the modified relaxation can be solved using the dual simplex method. It also possible to use an interior point method and this can be a good choice if the linear programming relaxations are large.

We say that any inequality $\pi^T x \leq \pi_0$ that is satisfied by all the feasible points of (ILP) is a valid inequality. The convex hull of the set of feasible solutions to (ILP) is a polyhedron. Every valid inequality defines a face of this polyhedron, namely the set of all the points in the polyhedron that satisfy $\pi^T x = \pi_0$. A facet is a face of a polyhedron that has dimension one less than the dimension of the polyhedron, and it is necessary to have an inequality that represents each facet in order to have a complete linear inequality description of the polyhedron. If all the facets of the convex hull of the set of integer feasible points are known, then the integer problem can be solved as a linear programming problem by minimizing the objective function over this convex hull. Unfortunately, it is not easy to obtain such a description. In fact, for an NP-Complete problem, such a description must contain an exponential number of facets, unless P=NP.

Dantzig's, Gomory, Chvatal-Gomory, strong, and general cutting planes are described below.

4.2.1 Dantzig's Cutting Plane Method:

Gomory [92] and [93] were among the earliest proponents of cutting plane methods, and the first method presented here is generally known as a Dantzig cut. Suppose that the linear programming relaxation of an all-integer programming problem was solved and in the optimal solution of the relaxed problem, there exists at least one variable $x_k$with a noninteger value b_i. As $x_j \in \mathbb{N}_0 \; \forall \; j$ is required, at least one variable that is currently nonbasic must have a positive value, and since its value must be integer, at least one of the current nonbasic variables must equal a value of one or larger. This implies that the sum of variables that are currently nonbasic, must equal at least one.

Defining nbv as the set of variables that are presently nonbasic, this constraint is called a Dantzig cut and can be written as

$$\sum_{j \in nbv} {}_j x \geq 1 \qquad (42)$$

Note that relation (4.2) does indeed satisfy the conditions of a cut: in the current solution all nonbasic variables equal zero, so at present the left-hand side of the relation equals zero, clearly violating the constraint, so that (1) cuts offthe current solution. On the other hand, the derivation of the relation has ensured that none of the integer feasible points were made infeasible.

A minor modification of Dantzig's cuts is credited by Taha [94] to Bowman and Nemhauser [95]. The authors observe that the sum of nonbasic variables on the left-hand side of a Dantzig cut can be replaced by the sum of nonbasic variables with noninteger coefficients in the current tableau. Apparently, such a cut is stronger than Dantzig's original cut, but there is not much difference in practical terms as normally only a few of the coefficients in the tableau are integer anyway.

4.2.2 Gomory's Cutting Plane Methods:

Another type of cut was introduced by Gomory [92]. To facilitate the discussion, let a_{ij} and b_i denothe the left hand and right hand side of the current tabulae. By assumption, there exists at least one variable $x_k = b_i \notin \mathbb{Z}$ and we say that the $i-th$ row is a source row. Define then fractional parameters $f_{ij} = a_{ij} - \lfloor a_{ij} \rfloor$ and $f_i = b_i - \lfloor b_i \rfloor$ with $f_{ij} \in [0;1[$ and $f_i \in]0;1[$ so that

$$x_k + \sum_{j \in nbv} \lfloor a_{ij} \rfloor x_j + \sum_{j \in nbv} {}_{ij} f_j = \lfloor b_i \rfloor + f_i \qquad (43)$$

As $x_j \in \mathbb{Z} \ \forall j$ is required as well as $\sum_{j \in nbv} \lfloor a_{ij} \rfloor x_j$ and $\lfloor b_i \rfloor \in \mathbb{Z}$, we obtain

$$f_i - \sum_{j \in nbv} {}_{ij} f_j \in \mathbb{Z} \qquad (44)$$

With $f_i < 1$ and the sum in (3) nonnegative, the expression in relation (3) must be less than or wqual to zero, so that

$$-\sum_{j \in nbv} a_{ij} x_j \leq -f_i \qquad (4.5)$$

Relation (4.5) is the fractional Gomory cut. As in the case of the Dantzig cut, the current solution violates this new relation (as at present, the left-hand side equals zero and the right-hand side is strictly negative) and the derivation assures that no integer solutions were cut off. At least one dual simplex iteration is needed to find a primal feasible solution again. A very different approach was taken by Gomory [81] in designing all-integer cuts. The basic idea is to start with a tableau in which all parameters are integers, and retain that property while optimizing.

4.2.3 Chvátal-Gomory's cutting plane method:

Cutting planes can be obtained by first combining together inequalities from the current linear programming relaxation and then exploiting the fact that the variables must be integral. This process is known as integer rounding, and the cutting planes generated are known as Chvátal-Gomory cutting planes. Integer rounding was described implicitly by Gomory [81], and described explicitly by Chvátal [96].

4.2.4 Model Optimization Using Branch and cut Algorithm:

The branch and cut algorithm is an efficient method for solving optimization problem to find the global optima. To solve the mathematical model developed in this thesis C++ at Code Blocks and Mixed Integer Linear Programming(MILP) solver ((intlinprog) at MATLAB are used.

Solving the mathematical model using the Integer Linear Programming requires some modification. The basic structure of the MATLAB MILP solver is like below

$$\min_x f^T x$$

$$Subject\ to \begin{cases} x\ (intcon) are\ integers \\ Ax \leq b \\ A_{eq} x = b_{eq} \\ lb \leq x \leq ub \end{cases} \qquad (4.6)$$

Where, x is the decision variables, f^T is a vector of the coefficients of the decision variable in the objective function. A is a coefficient matrix of the decision variable in the inequality constraints and b is a vector which is the right side of all inequality constraints. Similarly A_{eq} is a coefficient matrix of the decision variable in the equality constraints and b_{eq} is a vector which is the right side of all equality constraints. lb and ub are lowar bounds and upper bounds of the decision variable respectively.

To solve the model by this solver first the model is modified as follows to develop the necessary constraint equations.

Objective function:

$$Min, \quad \sum_{\lambda \in \Omega} {}^{\lambda}pC_{max}^{\lambda} + \delta \sum_{\lambda \in \Omega} {}^{\lambda}p\left|C_{max}^{\lambda} - \sum_{\lambda \in \Omega} {}^{\lambda}pC_{max}^{\lambda}\right| \tag{47}$$

Subject to,

$$\sum_{k=1}^{m} {}_{j,i,k}X = 1 \qquad \forall\, i,j \tag{48}$$

$$X_{j,i,k} \leq {}_{j,i,k}e \qquad \forall\, i,j,k \tag{49}$$

$$-C_{j,i}^{\lambda} + C_{j,i-1}^{\lambda} + \sum_{k=1}^{m} {}_{j,i,k}X\, {}_{j,i,k}^{\lambda}P \leq 0 \qquad \forall\, i,j,\lambda \tag{410}$$

$$-C_{j,1}^{\lambda} + \sum_{k=1}^{m} {}_{j,1,k}X\, {}_{j,1,k}^{\lambda}P \leq 0 \qquad \forall\, j,\lambda \tag{411}$$

$$-C_{j,i}^{\lambda} + X_{j,i,k}\, {}_{j,i,k}^{\lambda}P \leq {}_{k}a \qquad \forall\, i,j,k,\lambda \tag{412}$$

$$-C_{j,i}^{\lambda} + C_{l,h}^{\lambda} + M\left(X_{j,i,l,h} + X_{j,i,k} + X_{l,h,k}\right) \leq -P_{j,i,k}^{\lambda} + 3M \qquad \forall\, j \leq n, l \leq n(j,i) \neq (l,h), k, \lambda \tag{413}$$

$$-C_{l,h}^{\lambda} + C_{j,i}^{\lambda} + M\left(-X_{j,i,l,h} + X_{j,i,k} + X_{l,h,k}\right) \leq -P_{j,i,k}^{\lambda} + 2M \qquad \forall\, j \leq n, l \leq n(j,i) \neq (l,h), k, \lambda \tag{414}$$

$$-C_{max}^{\lambda} + C_{j,n_j}^{\lambda} \leq 0 \qquad \forall\, j,\lambda \tag{415}$$

$$C_{j,i}^{\lambda} \geq 0 \tag{416}$$

$$X_{j,i,k}\ ,\ X_{j,i,l,h}\ ,e_{j,i,k}\ \in \{0,1\} \qquad \forall\ i,j,h,l,k \tag{417}$$

This is for the first stage of optimization. The constraint equations of the second stage are modified in the similar way. After the generation of equation f, A, b, Aeq, beq, lb and ub matrices are developed and optimization is done by using the above mentioned programming software. The results of several numerical examples are shown in chapter 6.

CHAPTER V

FLEXIBLE JOB SHOP SCHEDULING USING GENETIC ALGORITHM

Different computational methodologies and procedures have been proposed to solve flexible job shop scheduling problems, which could be classified into three catagories; (1) exact procedures, (2) heuristics and improvement procedures and (3) meta-heuristic. FJSPs are computationally difficult, in an *n* jobs m machine problem m^n combinations need to be checked. Due to the combinatorial nature of the problem, exact (optimal) algorithms have been successfully applied only to small problems (i.e. 10 jobs 10 machines), but they require high computational efforts and extensive memory capabilities. Again, despite the guarantee of global optimality by the exact algorithm, their efficiency as well as accuracy decreases as the number of jobs and machines increases. As a result, metaheuristic algorithms have got the attention in recent years to solve FJSP. This is due to their ability to generate feasible solutions in the least possible computational time.

The developed flexible job shop scheduling model is a multi-objective constrained ILP problem which falls on the class of the most difficult optimization problem. In the previous chapter, this model is optimized using a branch and cut algorithm. But, it has been observed that, it requires more computational time in reaching optimality. Therefore, in this chapter, genetic algorithm is used to solve the developed model.

GA is a well known search algorithm and has been widely used in different field of study. In job shop scheduling problems it was extensively used by the researcher. Some researchers also proposed some modification to improve the local search method. Hybridization with other algorithms are also done by some researcher. Neubauer [97] discussed about several approaches to different production scheduling using genetic algorithm. A genetic algorithm was proposed by Jawahar and Aravindan [98] to derive an optimal combination of priority dispatching rules “pdrs” (independent pdrs one each for one Work Cell “WC”), to resolve the conflict among the contending jobs.

An effective hybrid genetic algorithm with three novel features, full active schedule (FAS), a new crossover operator called the precedence operation crossover (POX) and improved generation alteration model are introduced to solve the job shop problem by Zhang and Rao in [99]. For solving large size job shop problem Wang, Xiao and Yin [100] proposed a two-stage GA which attempts to firstly find the fittest control parameters, namely, number of population, probability of crossover, and probability of mutation, for a given job shop problem with a fraction of time using the optimal computing budget allocation method, and then the fittest parameters are used in the GA for a further searching operation to find the optimal solution. A hybrid genetic algorithm was proposed by Jie and Mitsuo [101] to solve the job shop problem subject to availability constraints. Countless papers are available on the application of genetic algorithm in solving job shop problem [102-104] of different types. In the following section definition, basic idea, components of GA and a brief discussion on STS will be provided.

5.1 Genetic Algorithm:

GA is an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. During each temporal increment (called a generation), the structure in the current population are rated for their effectiveness as domain solutions, and on the basis of these evaluations, a new population of candidate solution is formed using specific genetic operators such as reproduction, crossover and mutation.

The genetic algorithm technique was first invented by Holland [105] and has been successfully applied to numerous large search space problems by Davis [106]; Forrest [107]; Goldberg [108]. It is a search algorithm based on the mechanics of the natural selection process (biological evolution). The most basic concept is that the strong tend to adapt and survive while the weak tend to die out. That is, optimization is based on evolution and the “Survival of the fittest” concept. GA has the ability to create an initial population of feasible solutions, and then recombine them in a way to guide their search to only the most promising areas of the state space. Each feasible solution is encoded as a chromosome (string) also called genotype and each chromosome is given a measure of fitness via a fitness

(evaluation or objective) function. The fitness of a chromosome determines its ability to survive and produce offspring. A finite population of chromosome is maintained. GA uses probabilistic rules to evolve a population from one generation to the next. They have a high built in degree of randomness to escape from local optima and inferior regions of the solution space. Through parallel processing on a population of randomly generated chromosomes, it speeds up the whole search procedure. There is an abundance of applications of GAs in almost any field, including an extensive use in solving the FLP. In short it is a robust search technique and produce "close" to optimal results in a "reasonable" amount of time.

5.1.1 Pseudo-code and Flow Chart for Generic GA:

The procedure of a generic GA is shown as follows:

Begin

INITIALIZE population with random candidate solutions;

EVALUATE each candidate;

Repeat until TERMINATION-CONDITION is satisfied

1. SELECT parents;
2. CROSSOVER between parents
3. MUTATE the resulting children
4. EVALUATE children;
5. SELECT individuals for next generation

The procedure of generic GA can be represented by the following Figure 5.1

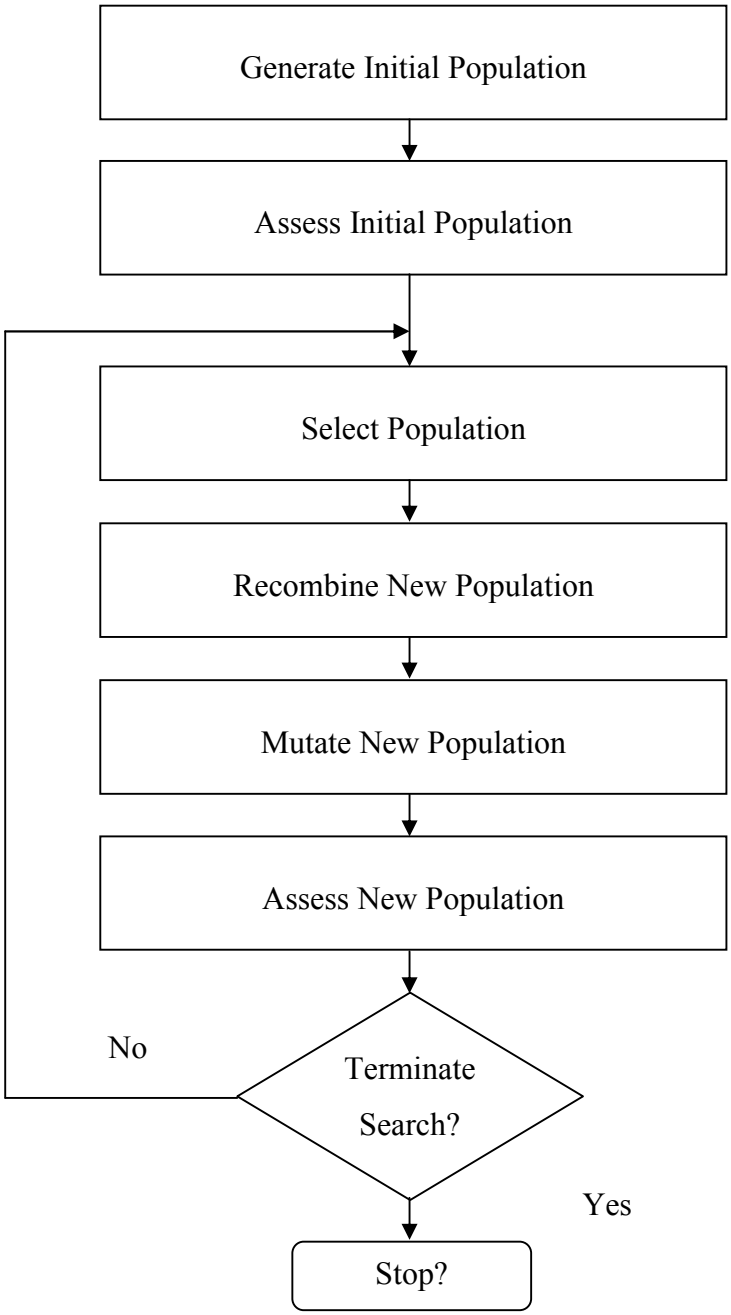


Figure 5.1: Flowchart of Genetic Algorithm

5.1.2 Basic Components of GA:

As described in Davis [106], a genetic algorithm has five components:

1. A means of encoding solutions to the problem as a chromosome.
2. A function that evaluates the "fitness" of a solution.
3. A means of obtaining an initial population of solutions.
4. Reproduction operators for the encoded solutions.
5. Appropriate settings for the genetic algorithm control parameters.

5.1.2.1 Chromosome Encoding:

The first, and perhaps the most important, step in applying a genetic algorithm to a problem is to choose a way to represent a solution to the problem as a finite-length string over a finite alphabet. These strings are referred to as chromosomes. The values on the chromosome may be arranged and interpreted as needed. They may represent Boolean values, integers, or even discretized real numbers. Complex chromosome can have combination of two or more type. Some encodings, which have been already used successfully, have been introduced here.

Binary - Binary encoding is the most common, mainly because first works about GA used this type of encoding. In binary encoding every chromosome is a string of bits, 0 or 1.

Example: Chromosome A: 101100101100

Permutation - Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem. In permutation encoding, every chromosome is a string of numbers, which represents number in a sequence.

Example: Chromosome A: 1 5 3 2 6 4 7 9 8

Value - Direct value encoding can be used in problems, where some complicated values are used. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem e.g. real numbers or chars to some complicated objects.

Example: Chromosome A: 1.2324 5.3243 0.4556 2.3293 2.4545

Chromosome B: (back), (back), (right), (forward), (left)

Tree - Tree encoding is used mainly for evolving programs or expressions, for genetic programming. In tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language.

Example:

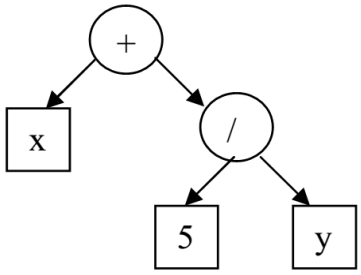


Figure 5.2: Tree encoding for equation: (+ x (/ 5 y))

The choice of how to encode solutions on a chromosome is of primary importance to the success of the genetic algorithm approach to a problem. The encoding of information on the chromosome should be right for the problem rather than specific to the problem. The encoding should be able to represent all the relevant parameters of the problem and should avoid other parameters. Using parameters that are not directly relevant will cause the genetic algorithm to be subject to changes in the problem that would not otherwise affect it, thereby making it no more useful than a specialized heuristic. Some knowledge of the search space is, of course, unavoidable according to Rawlins, [109].

5.1.2.2 Fitness Function:

A function is needed that will interpret the chromosome and produce an evaluation of the chromosome's fitness. This function must be defined over the set of possible chromosomes and is assumed to return some positive value representing the fitness. The definition of this function is crucial because it must accurately measure the desirability of the features described by the chromosome. In addition, the function must make this evaluation in a very efficient manner because of the large number of times the function will be called during the execution of the genetic algorithm. For

example, with a population of 100 chromosomes that runs for 1000 generations, there could be as many as 100,000 calls to this evaluation function during execution.

5.1.2.3 Choosing an Initial Population:

In a "pure" genetic algorithm, the initial population is chosen randomly, with the goal of selecting chromosomes from all over the search space. Whatever genetic material is in the initial population will be the only material, except for the rare changes due to mutation, available to the genetic algorithm during its search. One might employ a heuristic to choose the initial population in an attempt to introduce the "right" genetic building blocks into the population. However, this can lead to problems of premature convergence to a local optimum since genetic algorithms are "notoriously opportunistic" Grefenstette [110].

5.1.2.4 Reproduction Operators:

According to GA outline, Parent Selection Mechanism is a prerequisite for reproduction operations: Crossover and Mutation.

***Parent Selection Mechanism*:** The role of parent selection is to distinguish among individuals based on their quality to allow the better individuals to become parents of the next generation. Parent selection is probabilistic. Thus high quality individuals get a higher chance to become parents than those with low quality. Nevertheless, low quality individuals are often given a small but positive chance; otherwise the whole search could become too greedy and get stuck in a local optimum. Some of the selection methods are described below:

I. *Roulette Wheel Selection* - Parents are selected according to their fitness. The better the chromosomes, the more chances to be selected. Imagine a roulette wheel where every chromosome in a population has its place big accordingly to its fitness function. This can be simulated by following algorithm.

Roulette Wheel Selection procedure:

1. *[Sum]* Calculate sum of all chromosome's fitness in population - sum *S*.
2. *[Select]* Generate random number from interval *(0,S)* - *r*.
3. *[Loop]* Go through the population and sum fitness from *0* - sum *S*. When the sum *S* is greater than *r*, stop and return the chromosome where you are.

Step 1 is performed only once for each population.

II. *Rank Selection* - Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness *1*, second worst *2* etc. and the best will have fitness *N* (number of chromosomes in population).

III. *Steady-State Selection* - Main idea of this selection is that big part of chromosomes should survive to next generation. In every generation a few (good - with high fitness) chromosomes are selected for creating a new offspring. Then some (bad – having low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

Crossover Operator: This operator merges information from two parents' genotype into one or two offspring genotypes. Crossover is a stochastic operator: the choice of what parts of each parent are combined and the way these parts are combined depend on random drawings. The principle behind crossover is simple: by mating two individuals with different but desirable features, an offspring can be produced which combines both of these features. These are different kinds of crossover:

I. *One-point crossover* - One crossover point is selected. String from beginning of chromosome to the crossover point is copied from one parent; the rest is copied from the second parent.

Example: **11001**011+11011**111** = **11001111** & 11011011

Parent1 + Parent 2 = Child1 & Child 2

II. *Two point crossover* - Two crossover points are selected. String from beginning of chromosome to the first crossover point is copied from one

parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent.

Example: **11**001**011** + 11**011**11 = **11011111** & 11001011

Parent1 + Parent 2 = Child1 & Child 2

III. *Uniform crossover* – Genes are randomly copied from the first or from the second parent.

Example: 1**10**01**011** + **1**10**111**01 = **11011111** & 11011111 (random)

Parent1 + Parent 2 = Child1 & Child 2

Mutation Operator: A unary variation operator is called mutation. It is applied to one genotype and delivers a modified mutant. In general, mutation is supposed to cause a random unbiased change. Mutation has a theoretical role; it can guarantee that the space is connected.

Example: 1**1**001001 = 1**0**001001 (2^{nd} bit is inverted).

5.1.2.5 Genetic Algorithm Control Parameters:

There are other parameters that govern the genetic algorithm search process. Some of these are:

I. *Population size* - Determines how many chromosomes, and therefore how much genetic material, are available for use during the search. If there is too little, the search has no chance to adequately cover the space. If there is too much, the genetic algorithm wastes time evaluating chromosomes.
II. *Generations* - Specifies how many times the population will be replaced through reproduction.
III. *Crossover Rate* - Specifies the probability of crossover (mating) occurring between two chromosomes.
IV. *Mutation Rate* - Specifies the probability that a value in the chromosome of a newly created offspring will be randomly changed.

V. *Termination Condition* - GA is stochastic process and mostly there are no guarantees to reach a global optimum. Commonly used conditions for terminations are the following:
   - A solution is found that satisfies minimum criteria.
   - Fixed number of generations reached.
   - For a given number of generations, there is no improvement in fitness.

5.1.3 Constraints Handling in GA:

There are many ways to handle constraint in a GA. At the high conceptual level it can be distinguished in two cases: Indirect constraint handling and direct constraint handling. Indirect constraint handling means to incorporate them in the fitness function *f(x)* such that *f(x)* optimal implies that the constraints are satisfied. Direct constraint handling means that the constraint stays as they are and the GA is 'adapted' to enforce them. Direct and Indirect both can be used together in a single application.

5.1.3.1 Direct Constraint Handling:

Treating constraint directly implies that violating them is not reflected in the fitness function, thus there is no bias towards chromosome satisfying them. Therefore the population will become less and less feasible with respect to these constraints. This means feasibility of the chromosomes have to be monitored and maintained. The basic problem in this case is that the regular operators are blind to constraints, mutating one or crossing over two feasible chromosomes can result in infeasible offspring. Typical approaches to handle constraints directly are the following:

I. Eliminating infeasible solution
II. Repairing infeasible solution
III. Preserving feasibility by special operators
IV. Decoding, i.e., the search space.

Eliminating infeasible solution is very inefficient, and therefore hardly applicable. Repairing infeasible candidates requires a repair procedure for the chromosome. Preserving feasibility can be NP-Complete. Finally decoding can be seen as shifting to a search space that is different than the original problem formulation.

5.1.3.2 Indirect Constraint Handling:

In the case of indirect constraint handling, the optimization objectives replacing the constraints are viewed as penalties for constraint violation hence to be minimized. In general penalties are given for violated constraints. Advantages of indirect constraint handling are:

- Reproduction of the problem to simple optimization.
- Possibility of embedding user preferences by means of weights.

Disadvantages of indirect constraint handling are:

- Loss of information packing everything in a single number.
- Does not work well with sparse problem.

5.1.4 Reasons to choose GA:

Genetic algorithm is a parallel, stochastic search process. It is widely used in many applications due to the following reasons:

1. The search is highly parallel, with each population member defining many different possible search directions. Potentially, GA search could be implemented extremely efficiently on massively parallel hardware.
2. No special information about the solution surface such as gradient or local curvature need to be identified. The objective function need not to be smooth, continuous or unimodal.
3. Genetic algorithms have proved to be fairly robust under varying parameter settings and problem particulars. As long as solutions with similar encodings do not have highly variant objective function values, genetic algorithms usually find near optimal solutions.
4. Being a population-based approach, GAs are well suited to solve multi-objective optimization problems. A generic single-objective GA can be modified to find a set of multiple non-dominated solutions in a single run.

Since the optimization of multi-objective unequal-area FLPs have to do with the vast number of possible physical layouts, and with the existence of many locally optimal layouts as well as should capture the pareto front, therefore GA is used in this thesis work.

5.2 Model optimization using GA:

The MATLAB Integer GA solver is used for solving the FJSP. The basic structure of the GA solver is as follows.

$$\min_x f^T x$$

$$Subject\ to \begin{cases} x\ (intcon) are\ integers \\ Ax \leq b \\ A_{eq}x = b_{\ eq} \\ lb \leq x \leq ub \end{cases} \quad (51)$$

Here the input arguments are

Table 5.1: Input Arguments of GA	
Fitness function (fitnessfcn)	To handle the fitness function(Objective function). The fitness function accept a row vector of length of number of variables and return a scalar value.
Number of Variable (nvars)	Positive integer representing the number of variables in the problem.
A	Matrix for linear inequality constraints showed in equation set 5.1. If the problem has m linear inequality constraints and nvars variables, then A is a matrix of size m-by-nvars.
B	Vector for linear inequality constraints showed in equation set 5.1. If the problem has m linear inequality constraints and nvars variables, then b is a vector of length m.
Aeq	Matrix for linear equality constraints of the form showed in equation 5.1. If the problem has m linear equality constraints and nvars variables, then Aeq is a matrix of size m-by-nvars.
Beq	Vector for linear equality constraints of the form showed in equation set 5.1. If the problem has m linear equality constraints and nvars variables, then beq is a vector of length m.
Lb	Vector of lower bounds.

Ub	Vector of upper bounds.
Integer Variables (IntCon)	Vector of positive integers taking values from 1 to nvars. Each value in IntCon represents an x component that is integer-valued.

Though GA solver can handle both linear equality and inequality constraints at the same time for continuous variables, it can't handle the equality constraints and integer variables at the same time. It is a limitation of using GA solver. However, linear inequalities can be modified as follows to solve the problem if relaxing the integer variables is not possible.

$$\begin{cases} A_{eq}x \geq b_{eq} \\ A_{eq}x \leq b_{eq} \end{cases} \quad (52)$$

GA solver also provides numerous options to modify the algorithm. Some options are discussed below

5.2.1 Population Options:

This option let the user to specify the population parameters like, population type, population size, initial population, initial scores, initial range etc. For solving the FJSP model, the population size is set to the number of variables and the initial range is kept limited between upper bound and lower bound.

5.2.2 Fitness Scaling Options:

Fitness scaling converts the raw fitness scores that are returned by the fitness function to values in a range that is suitable for the selection function. Though several scaling functions are provided by the solver like rank, proportional, top, in this research work proportional scaling function is used. Proportional scaling makes the scaled value of an individual proportional to its raw fitness score.

5.2.3 Selection Options:

Selection options specify how the genetic algorithm chooses parents for the next generation. The options provided by the selection options of the GA solver of MATLAB are

Stochastic uniform: The default selection function, Stochastic uniform, lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. The first step is a uniform random number less than the step size.

Remainder: Remainder selection assigns parents deterministically from the integer part of each individual's scaled value and then uses roulette selection on the remaining fractional part. For example, if the scaled value of an individual is 2.3, that individual is listed twice as a parent because the integer part is 2. After parents have been assigned according to the integer parts of the scaled values, the rest of the parents are chosen stochastically. The probability that a parent is chosen in this step is proportional to the fractional part of its scaled value.

Uniform: Uniform selection chooses parents using the expectations and number of parents. Uniform selection is useful for debugging and testing, but is not a very effective search strategy.

Roulette: Roulette selection chooses parents by simulating a roulette wheel, in which the area of the section of the wheel corresponding to an individual is proportional to the individual's expectation. The algorithm uses a random number to select one of the sections with a probability equal to its area.

Tournament: Tournament selection chooses each parent by choosing Tournament size players at random and then choosing the best individual out of that set to be a parent. Tournament size must be at least 2.

5.2.4 Reproduction Options:

Reproduction options let the user to specify how the genetic algorithm creates children for the next generation.

Elite count: specifies the number of individuals that are guaranteed to survive to the next generation.

Crossover fraction: specifies the fraction of the next generation, other than elite children, that are produced by crossover.

In this work the reproduction options kept to default crossover function

5.2.5 Mutation Options:

Mutation options let the user to specify how the genetic algorithm makes small random changes in the individuals in the population to create mutation children. Mutation provides genetic diversity and enables the genetic algorithm to search a broader space. The default mutation function for unconstrained problems, Gaussian, adds a random number taken from a Gaussian distribution with mean 0 to each entry of the parent vector. Unfortunately this option can't be used for integer programming.

5.2.6 Stopping Criteria Options:

Stopping criteria determine what causes the algorithm to terminate. It can be specified by the following options:

Generations: Specifies the maximum number of iterations for the genetic algorithm to perform. The default is 100*nvar

Time limit: Specifies the maximum time in seconds the genetic algorithm runs before stopping, as measured by cputime.

Fitness limit: The algorithm stops if the best fitness value is less than or equal to the value of Fitness limit.

Stall generations: The algorithm stops if the average relative change in the best fitness function value over Stall generations is less than or equal to Function tolerance.

Stall time limit: The algorithm stops if there is no improvement in the best fitness value for an interval of time in seconds specified by Stall time limit, as measured by cputime.

Function tolerance: The algorithm stops if the average relative change in the best fitness function value over Stall generations is less than or equal to Function tolerance

The modified model for solving with the GA solver is as follows

5.2.7 Fitness function:

$$Min, \quad \sum_{\lambda \in \Omega} {}^{\lambda}pC_{max}^{\lambda} + \delta \sum_{\lambda \in \Omega} {}^{\lambda}p\left|C_{max}^{\lambda} - \sum_{\lambda \in \Omega} {}^{\lambda}pC_{max}^{\lambda}\right| \tag{53}$$

Subject to,

$$\sum_{k=1}^{m} {}_{j,i,k}X \leq 1 \qquad \forall\, i,j \tag{54}$$

$$-\sum_{k=1}^{m} {}_{j,i,k}X \leq -1 \qquad \forall\, i,j \tag{55}$$

$$X_{j,i,k} \leq e_{j,i,k} \qquad \forall\, i,j,k \tag{55}$$

$$-C_{j,i}^{\lambda} + C_{j,i-1}^{\lambda} + \sum_{k=1}^{m} {}_{j,i,k}X\, P_{j,i,k}^{\lambda} \leq 0 \qquad \forall\, i,j,\lambda \tag{56}$$

$$-C_{j,1}^{\lambda} + \sum_{k=1}^{m} {}_{j,1,k}X\, P_{j,1,k}^{\lambda} \leq 0 \qquad \forall\, j,\lambda \tag{57}$$

$$-C_{j,i}^{\lambda} + X_{j,i,k} P_{j,i,k}^{\lambda} \leq a_k \qquad \forall\, i,j,k,\lambda \tag{58}$$

$$-C_{j,i}^{\lambda} + C_{l,h}^{\lambda} + M\left(X_{j,i,l,h} + X_{j,i,k} + X_{l,h,k}\right) \leq -P_{j,i,k}^{\lambda} + 3M \qquad \forall\, j \leq n, l \leq n(j,i) \neq (l,h), k, \lambda \tag{59}$$

$$-C_{l,h}^{\lambda} + C_{j,i}^{\lambda} + M\left(-X_{j,i,l,h} + X_{j,i,k} + X_{l,h,k}\right) \leq -P_{j,i,k}^{\lambda} + 2M \qquad \forall\, j \leq n, l \leq n(j,i) \neq (l,h), k, \lambda \tag{510}$$

$$-C_{max}^{\lambda} + C_{j,n_j}^{\lambda} \leq 0 \qquad \forall\, j,\lambda \tag{511}$$

$$C_{j,i}^{\lambda} \geq 0 \tag{512}$$

$$X_{j,i,k}\,,\ X_{j,i,l,h}\,,\ e_{j,i,k} \in \{0,1\} \qquad \forall\, i,j,h,l,k \tag{513}$$

CHAPTER VI

RESULTS AND DISCUSSIONS

In this thesis work a two stage multi-objective flexible job shop scheduling model has been developed considering the prime objective to minimize the makespan of the overall system. The developed constrained multi-objective integer linear model has been optimized to determine the optimal arrangement of the operation of different jobs at different machines so that maximum completion time, variability of completion time of each operation and variability of makespan at different stages will be minimized. This model is illustrated with two numerical examples and then optimized using a branch and cut algorithm and a genetic algorithm.

6.1 Numerical Example:

In this case a four job five machine (4 x 5) problem is considered. The input data for the model are collected from Zhang [111]. The model requires the following input parameters for the first stage of optimization.

I. Number of jobs
II. Number of machines
III. Number of operation of each job
IV. Sequence of operation of each job
V. Processing time of each operation at different machines

In the second stage some other inputs will be needed. These are

I. Arrival time of the unpredicted job
II. Processing time of unpredicted job at different machines
III. Operations already completed or in process
IV. Time when each machine will be available for second stage operations

6.1.1 Optimization using Branch and Cut Algorithm*:*

To apply the integer linear programming the model is modified as described in chapter IV. The optimization using Branch and Cut Algorithm involve the following sequential steps

I. Reading the input data
II. Variable generation according to the model in terms of x
III. Developing the coefficient matrices for all equations
IV. Developing separate matrices for $A, b, A_{eq}, b_{eq}, lb, ub, intcon, f$
V. Defining the objective function
VI. Running optimization
VII. Reading output result
VIII. Decoding the result to generate separate Gantt chart at different scenarios both for job and machine vs. the time to make it easily readable to the user.

6.1.1.1 First stage optimization-Developing a Robust Schedule:

The input data for 4 x 5 job shop is as follows,

Number of jobs = 4

Number of machines = 5

Processing time of each operation at different machine are shown below,

Table 6.1: Processing time of different operations at scenario 1

Operation (O_{ji})	Machine				
	M-1	M-2	M-3	M-4	M-5
11	2	5	4	1	2
12	5	4	5	7	5
13	4	5	5	4	5
21	2	5	4	7	8
22	5	6	9	8	5
23	4	5	4	54	5
31	9	8	6	7	9
32	6	1	2	5	4
33	2	5	4	2	4
34	4	5	2	1	5
41	1	5	2	4	12
42	5	1	2	1	2
51	0	0	0	0	0
52	0	0	0	0	0
53	0	0	0	0	0

Here the processing time can be in hours or in days. The fifth job is a dummy job which is considered in the first stage to reduce the computational complexity at the second stage. For this example it is assumed that one unpredicted job may arrive after the initialization of the operation. One may consider two jobs. In that case two dummy jobs can be created. This job has no effect on the initial schedule as processing time of all operations at different machines are zero. These are the input data for the first scenario. In this thesis there are two scenarios are considered. The input data for second scenario are shown below.

Table 6.2: Processing time of different operations at scenario 2

Operation (O_{ji})	Machine				
	M-1	M-2	M-3	M-4	M-5
11	1	4	6	9	3
12	4	1	1	3	4
13	3	2	5	1	5
21	2	10	4	5	9
22	4	8	7	1	9
23	6	11	2	7	5
31	8	5	8	9	4
32	9	3	6	1	2
33	7	1	8	5	4
34	7	3	12	1	6
41	5	10	6	4	9
42	4	2	3	8	7
51	0	0	0	0	0
52	0	0	0	0	0
53	0	0	0	0	0

In the first stage two schedules will be generated. The decision regarding which schedule to execute depends on the scenario occurs at the real time. So the job shop has the flexibility to operate in different scenarios.

The forms of input and output of an optimization is important to make it usable by others for real application as one of the objective of this thesis work is to make the schedule adaptable to the real manufacturing or application. In this work, emphasis is given on generating output and taking inputs.

The flow chart of the optimization using the programming software is shown below

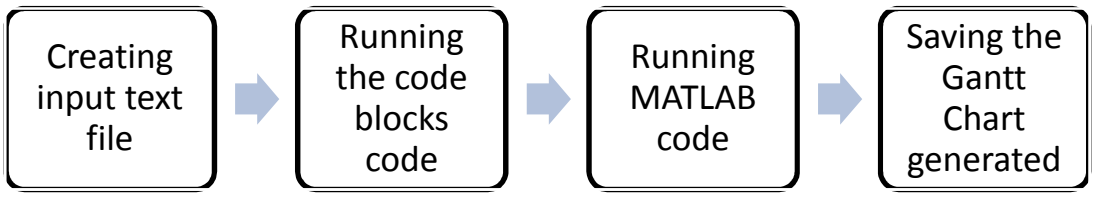


Figure 6.1: Flow chart of optimization steps

Here the input text consist the number of jobs, number of machines, number of operation of each job, processing time matrix at different machines and the big M Value.

One of the input text file is as follows.

```
5          <Number of machines>

5          <Number of Jobs>

3          <Number of operations for the first job>
2 5 4 1 2 <Processing time of Operation 1 of job 1 at 5 machines>
5 4 5 7 5
4 5 5 4 5

3
2 5 4 7 8
5 6 9 8 5
4 5 4 54 5

4
9 8 6 7 9
6 1 2 5 4
2 5 4 2 4
4 5 2 1 5

2
1 5 2 4 12
5 1 2 1 2

3
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

100
```

This is for variable and equation generation. After running C++ code the output log file is as follows

```
total number of machine 5
total number of job 5
1th job has 3 operations
2 5 4 1 2
5 4 5 7 5
4 5 5 4 5

2th job has 3 operations
2 5 4 7 8
5 6 9 8 5
4 5 4 54 5

3th job has 4 operations
9 8 6 7 9
6 1 2 5 4
2 5 4 2 4
4 5 2 1 5

4th job has 2 operations
1 5 2 4 12
5 1 2 1 2

5th job has 3 operations
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

value of Big M 100
assignment variables start x1
assignment variables end at x75

sequencing variables start x76
sequencing variables end at x269

completion variables start at x270
completion variables end at x284

max completion time variable is x285
```

Another input text file is required for developing Gantt Chart which contains the operation ID and processing time of the corresponding operations

```
1 1 2 5 4 1 2
1 2 5 4 5 7 5
1 3 4 5 5 4 5
2 1 2 5 4 7 8
2 2 5 6 9 8 5
2 3 4 5 4 54 5
3 1 9 8 6 7 9
3 2 6 1 2 5 4
3 3 2 5 4 2 4
3 4 4 5 2 1 5
4 1 1 5 2 4 12
4 2 5 1 2 1 2
5 1 0 0 0 0 0
5 2 0 0 0 0 0
5 3 0 0 0 0 0
```

where the first two numbers of each row denote the job number and operation number respectively.

6.1.1.1.1 Output of stage 1:

After creating the following input the optimization using MATLAB is initialized. The Gantt Charts and plot of the objective function are shown in the figures. The output file attached in the appendix. Here the probability of the two scenarios is assumed 0.6 and 0.4 respectively. The output of the optimization is as follows,

Scenario-1:

Table 6.3: Results of scenario 1 at stage 1

Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	4	4	1	1
1	2	7	2	4	6
1	3	13	3	5	11
2	1	16	1	2	2
2	2	25	5	5	7
2	3	26	1	4	11
3	1	33	3	6	6
3	2	37	2	1	7
3	3	44	4	2	9
3	4	49	4	1	11
4	1	51	1	1	3
4	2	57	2	1	8
5	1	65	5	0	0
5	2	68	3	0	0
5	3	75	5	0	0

Scenario-2:

Table 6.4: Results of scenario 2 at stage 1					
Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	1	1	1	1
1	2	8	3	1	2
1	3	14	4	1	7
2	1	16	1	2	3
2	2	24	4	1	6
2	3	28	3	2	8
3	1	35	5	4	4
3	2	39	4	1	5
3	3	42	2	1	6
3	4	49	4	1	8
4	1	54	4	4	4
4	2	56	1	4	8
5	1	65	5	0	0
5	2	68	3	0	2
5	3	72	2	0	7

So the makespan for scenario 1 and 2 are 11days and 8 days respectively and the objective function for the first stage is 11.24 days. So it is evident from the output that for completing all jobs at least 11.24 days will be required. The objective function value at two different scenario are plotted against the number of nodes which are shown in figure 6.2 and 6.3 respectively

From the Gantt chart output shown in figure 6.4-6.7, it is clear that a non-overlapping schedule is generated for the execution. The significance of generating two Gantt chart, one is job vs. time and other is machine vs. time is to identify the sequence of operations at different machine and to keep track of machine sequence of operations of each job. For instance let consider figure 6.4 and 6.5. In figure 6.4 the operation sequence are shown at different machines. So from this operator one can visualize the operation sequence at different machines easily. For example, in machine 2 operation no 12, 32 and 42 will be done sequentially. But, if someone is interested to know the machine sequence for a job it will be a hectic job to find the same color slots at different machines to identify the machine sequence. That's why the second type of Gantt chart is needed which shows the machine sequence for different jobs.

Therefore, observing these two diagrams one can easily execute the schedule generated.

For this optimization 5 Gomory Cuts and 1 Zero-half cut are applied for the first scenario. For scenario 2, 1 Gomory cut, 3 strong CG cuts and 1 zero-half cut are applied. Due to this cut generation the optimization time is significantly low. The cputime for this optimization is 559.13 seconds.

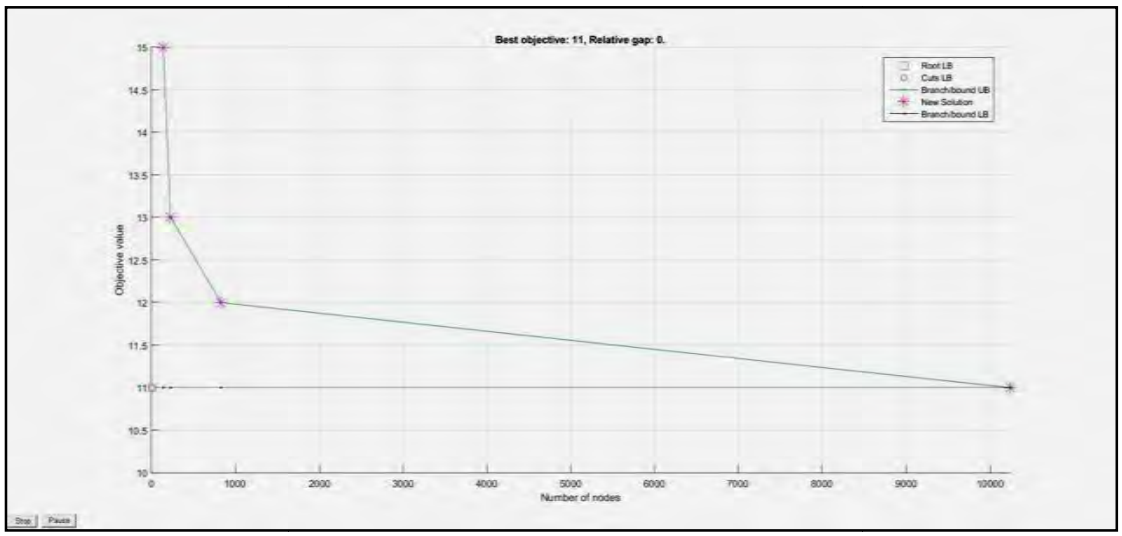


Figure 6.2: Objective versus Number of nodes at scenario 1of stage 1

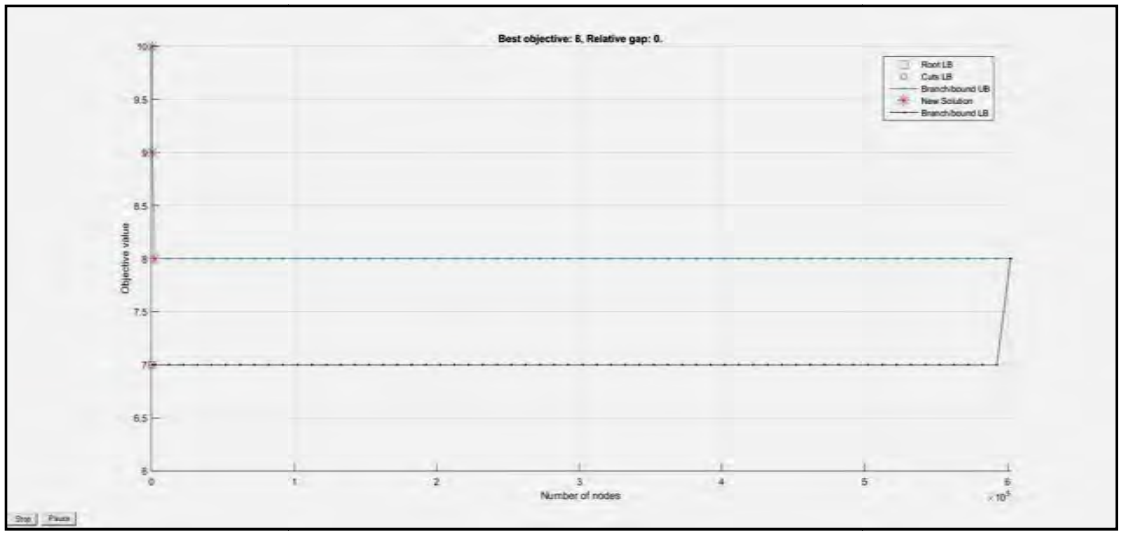


Figure 6.3: Objective versus Number of nodes at scenario 2of stage 1

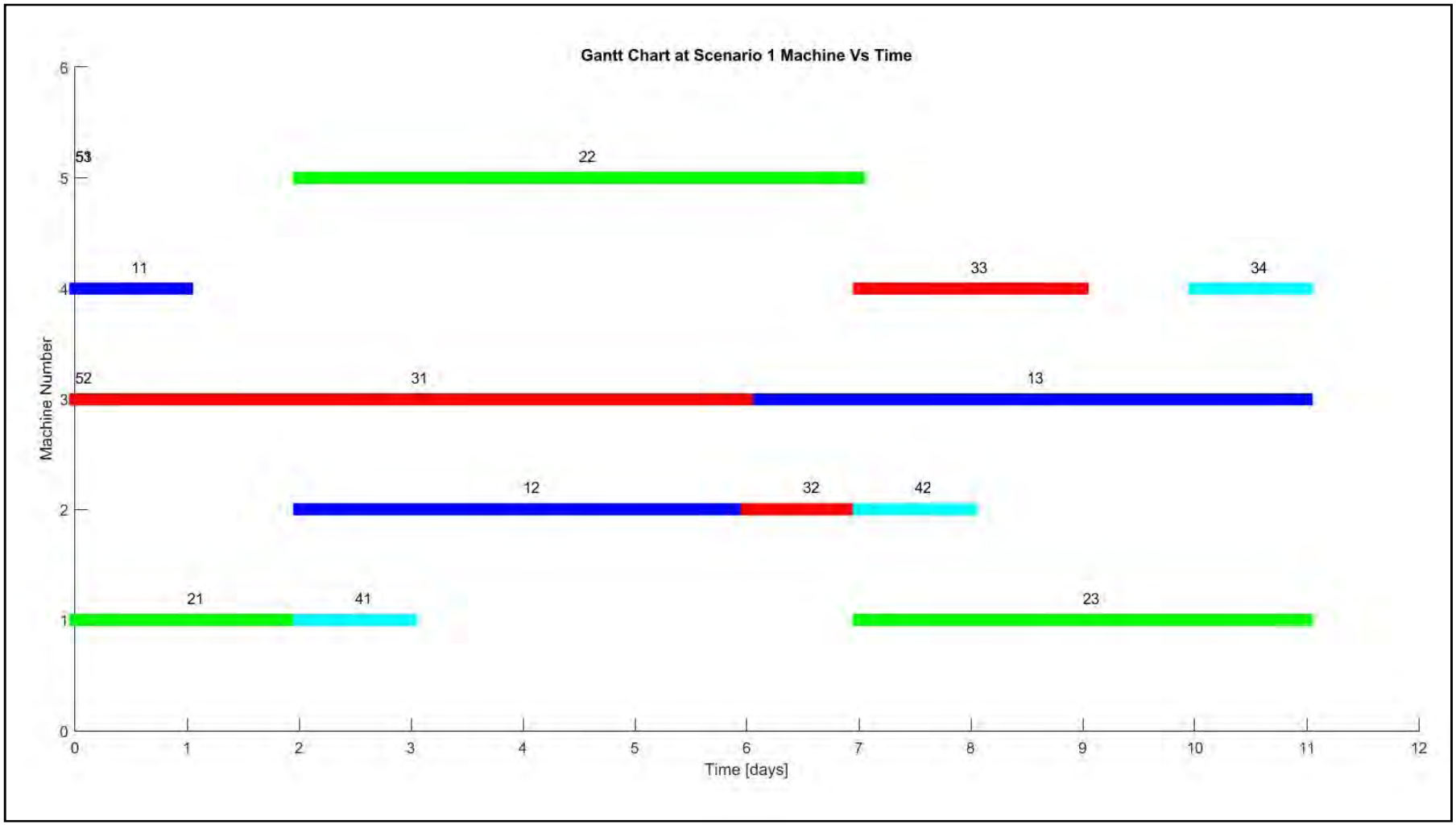


Figure 6.4: Gantt chart at scenario 1 of stage 1-Machine vs. Time

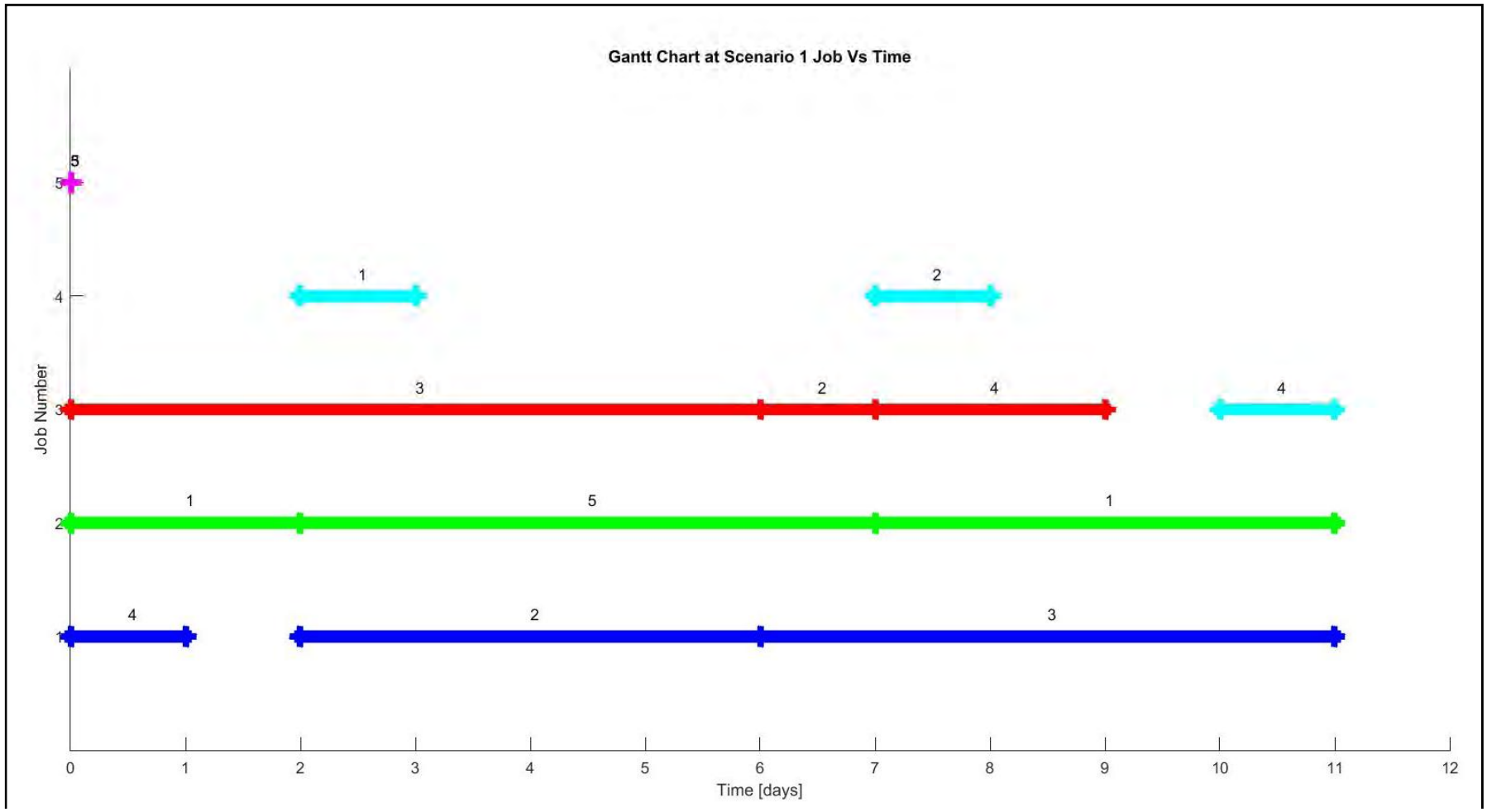


Figure 6.5: Gantt chart at scenario 1 of stage 1-Job vs. Time

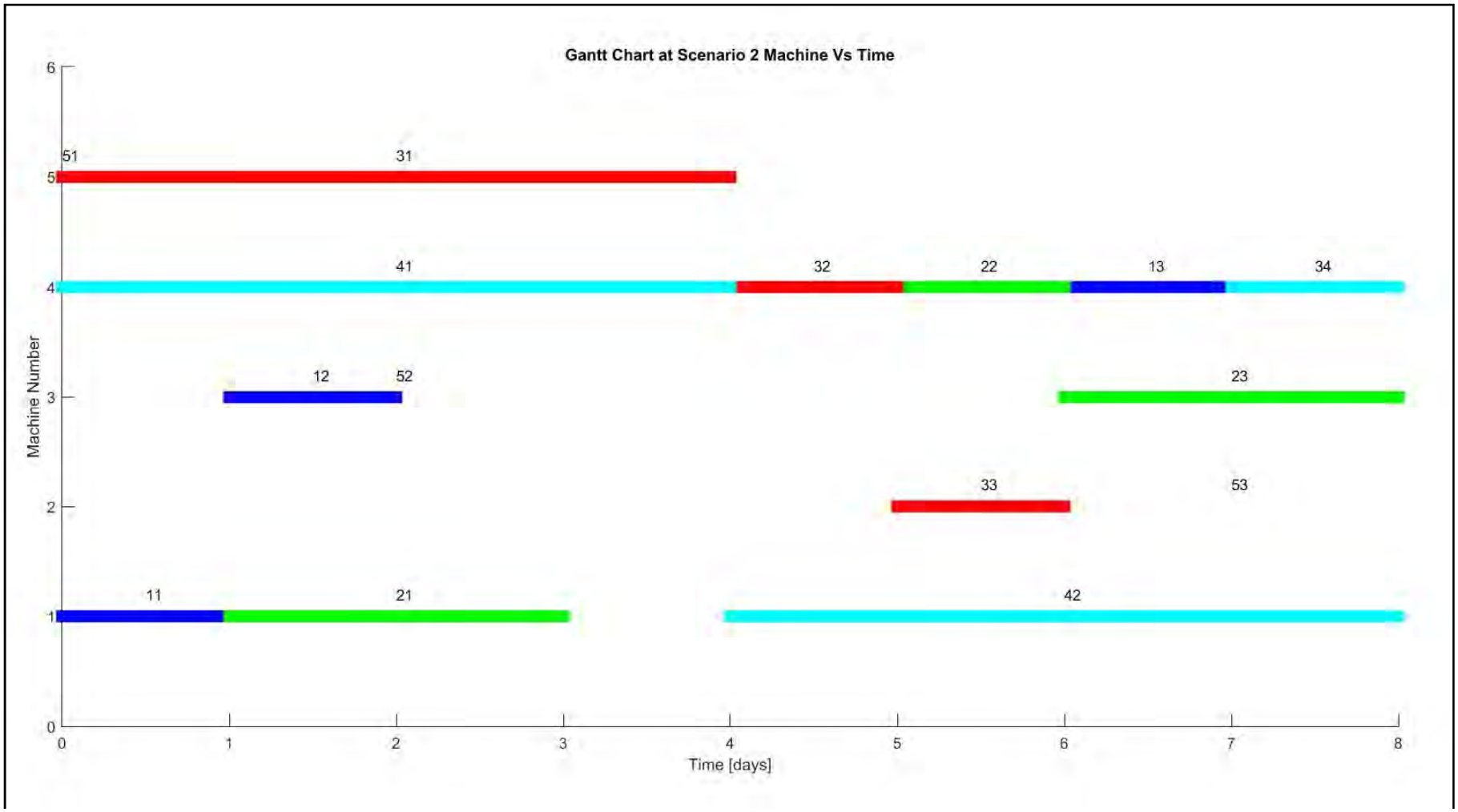


Figure 6.6: Gantt chart at scenario 2 of stage 1-Machine vs. Time

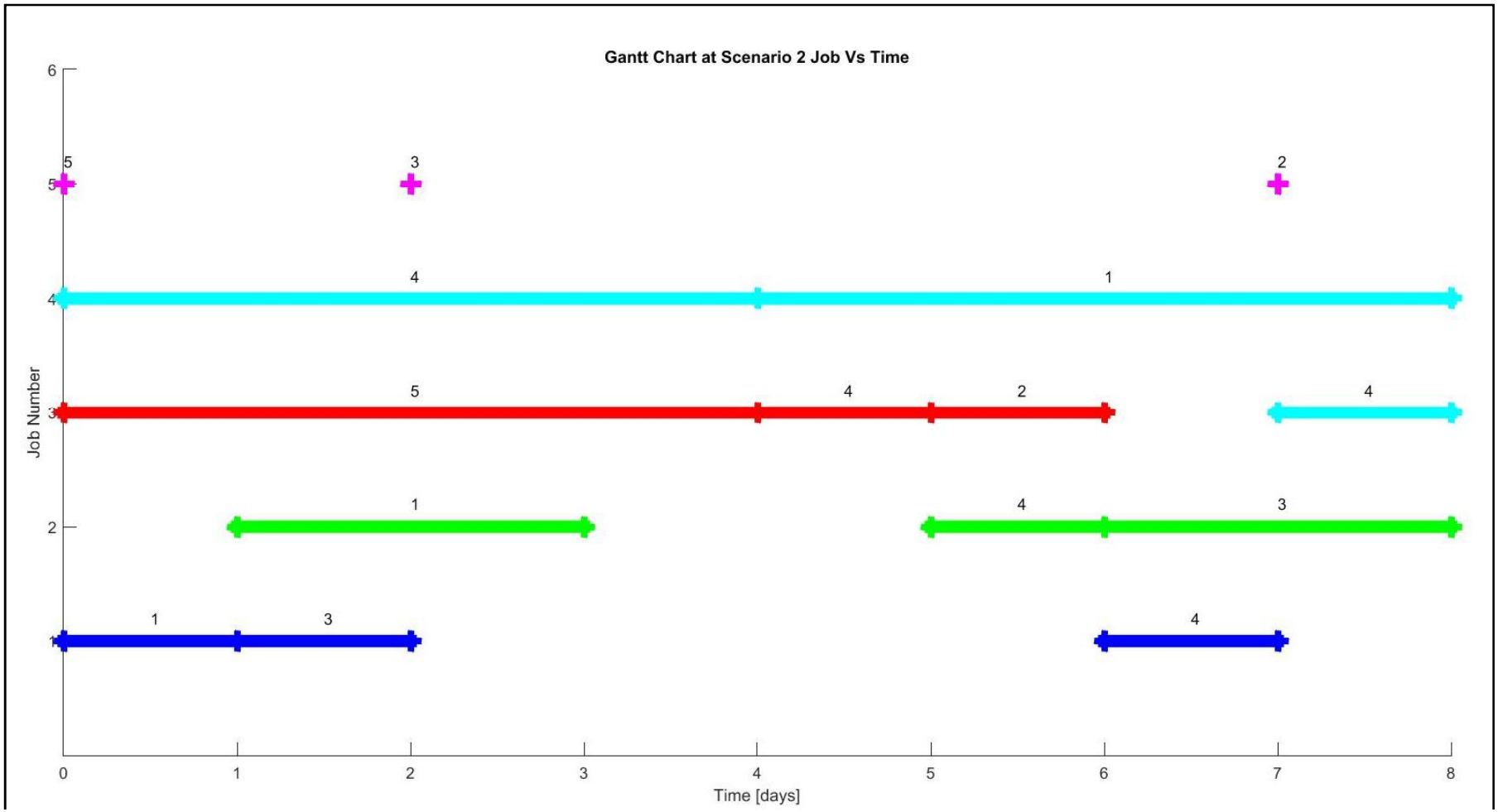


Figure 6.7: Gantt chart at scenario 2 of stage 1-Job vs. Time

6.1.1.2 Second stage optimization-Reactive Scheduling:

In real manufacturing environment jobs continuously arrive at shop floor and need to be scheduled at real time which give the rise of reactive scheduling. In this stage an unpredicted job is assumed which arrive at shop floor at time *t* when already the initial schedule is executed. Let's assume a job arrives at shop floor at time *t=3* which consists of three operations. After 3 days some operations has already completed and some operations are still in process in few machines. For rescheduling these completed jobs and work in process jobs must be excluded from the list of operations because preemption is not allowed. After 3 days at scenario 1, operations $O_{11}, O_{21}, O_{4\ 1}$are already completed and operations O_{12}, O_{22}, O_{31} are in process. It is convenient to assign processing time zero for this operation. So that in optimization equations variable ID will remain same. After doing so the new input data for optimization is like below

Table 6.5: Processing time of different operations at scenario 1 in stage 2

Operation (O_{ji})	Machine				
	M-1	M-2	M-3	M-4	M-5
11	0	0	0	0	0
12	0	0	0	0	0
13	4	5	5	4	5
21	0	0	0	0	0
22	0	0	0	0	0
23	4	5	4	54	5
31	0	0	0	0	0
32	6	1	2	5	4
33	2	5	4	2	4
34	4	5	2	1	5
41	0	0	0	0	0
42	5	1	2	1	2
51	5	7	11	3	2
52	8	3	10	7	5
53	6	2	13	5	4

As some jobs are still in process it is evident that all machines are not available at *t=3* which is also visible from the machine vs. time Gantt chart of scenario 1. Machine availability time, a_k for different machines obtained from the output of first stage are

Table 6.6: Machine availability time

M-1	M-2	M-3	M-4	M-5
3	6	6	3	7

These steps are called *preprocessing.* Similar preprocessing is done for scenario 2

6.1.1.2.1 Output of stage 2:

The output of stage 2 are given below

Scenario 1

Table 6.7: Results of scenario 1 at stage 2

Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	3	3	0	7
1	2	6	1	0	7
1	3	15	5	5	12
2	1	16	1	0	7
2	2	24	4	0	7
2	3	26	1	4	12
3	1	33	3	0	7
3	2	38	3	2	9
3	3	44	4	2	11
3	4	49	4	1	12
4	1	53	3	0	7
4	2	59	4	1	8
5	1	64	4	3	7
5	2	67	2	3	10
5	3	72	2	2	12

Scenario 2

Table 6.8: Results of scenario 2 at stage 2

Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	3	3	0	4
1	2	6	1	0	4
1	3	14	4	1	8
2	1	16	1	0	4
2	2	24	4	1	6
2	3	30	5	5	11
3	1	33	3	0	4
3	2	39	4	1	5
3	3	42	2	1	6
3	4	49	4	1	7
4	1	53	3	0	4
4	2	56	1	4	8
5	1	65	5	2	6
5	2	67	2	3	9
5	3	72	2	2	11

The objective function values are

Table 6.9: Objective function values		
Objective	Scenario-1	Scnario-2
Makespan	12	11
Variability of completions times of the operations	53	28
Variability of the makespan	1	3

The objective function plot and rescheduled Gantt charts are shown below

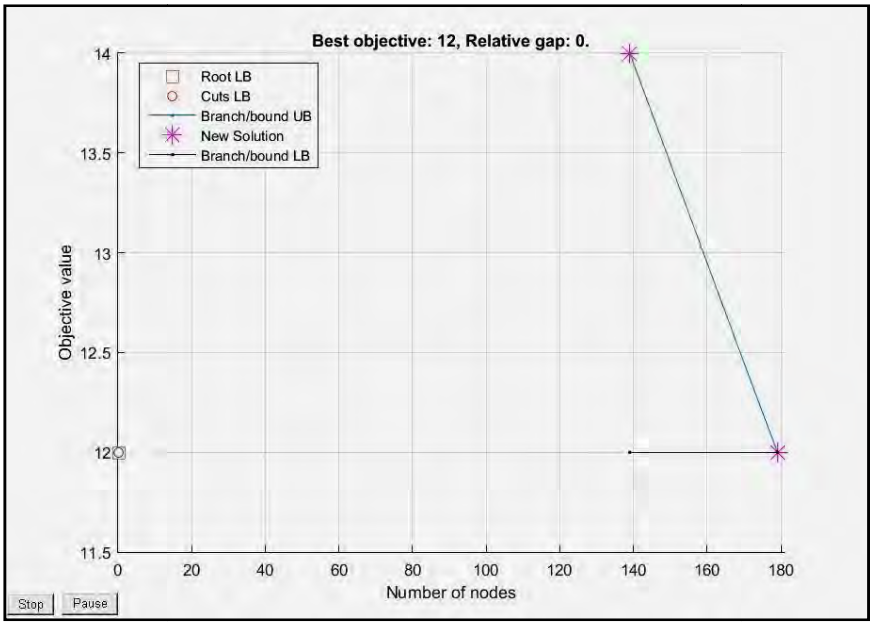


Figure 6.8: Objective versus Number of nodes at scenario 1of stage 2

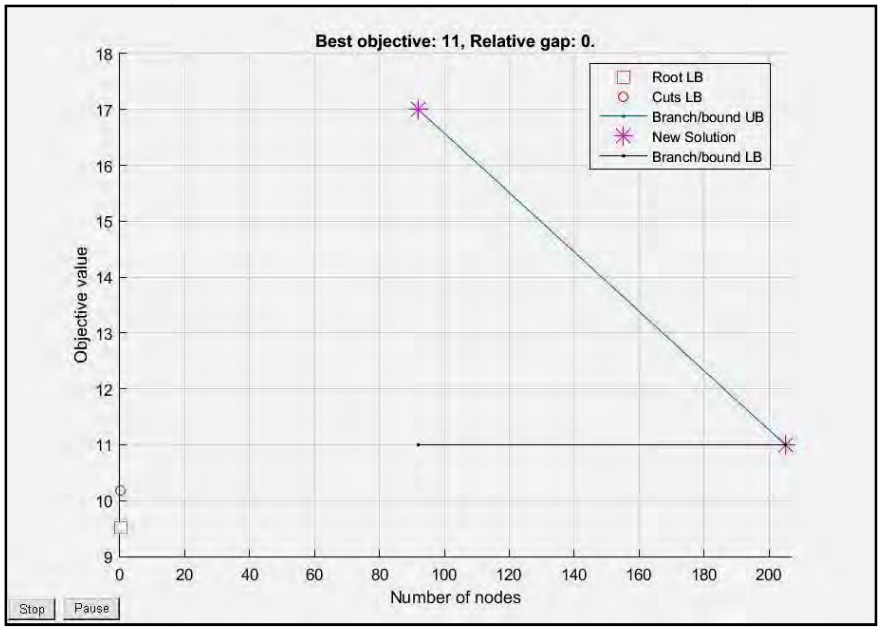


Figure 6.9: Objective versus Number of nodes at scenario 2of stage 2

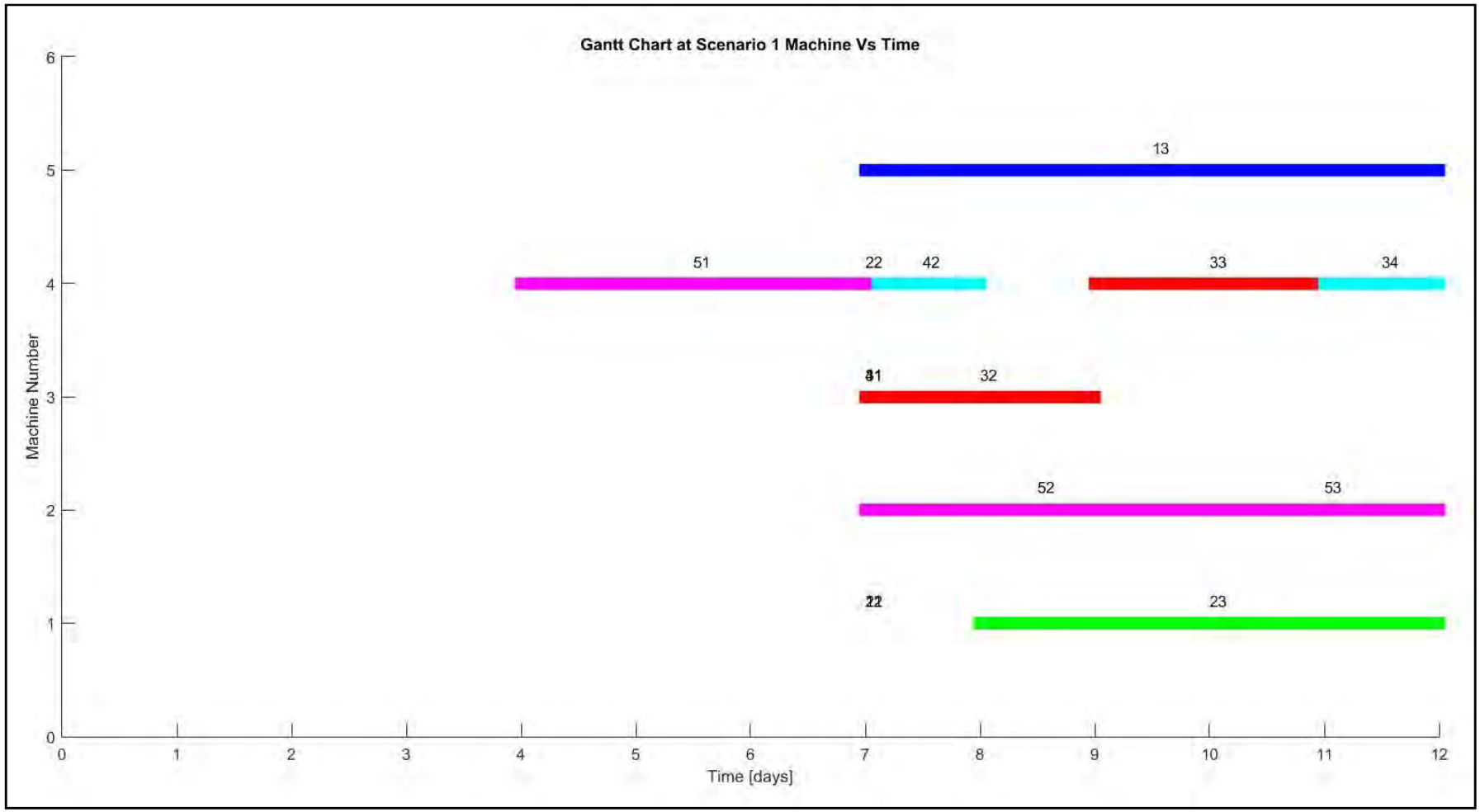


Figure 6.10: Gantt chart at scenario 1 of stage 2-Machine vs. Time

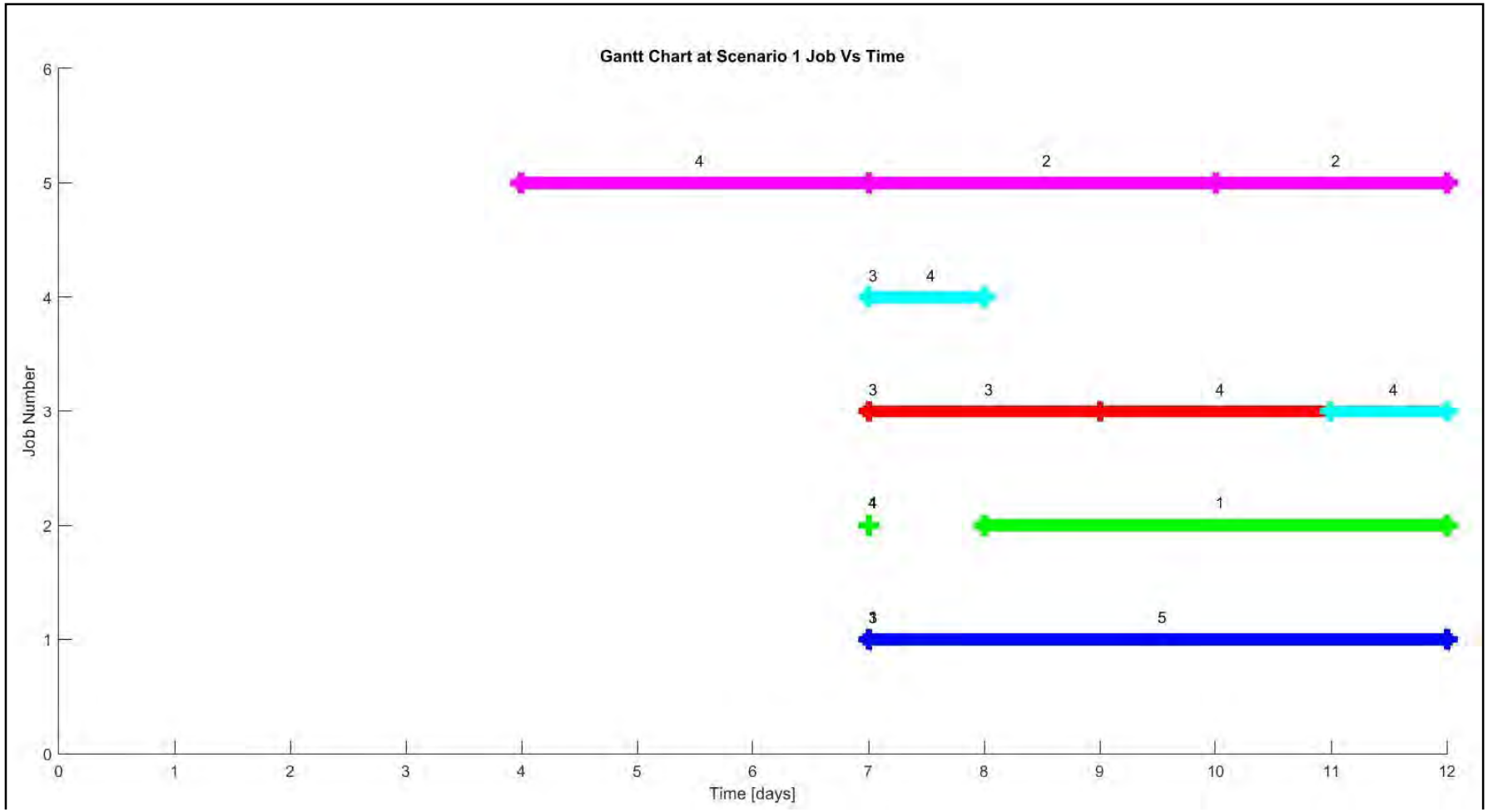


Figure 6.11: Gantt chart at scenario 1 of stage 2-Job vs. Time

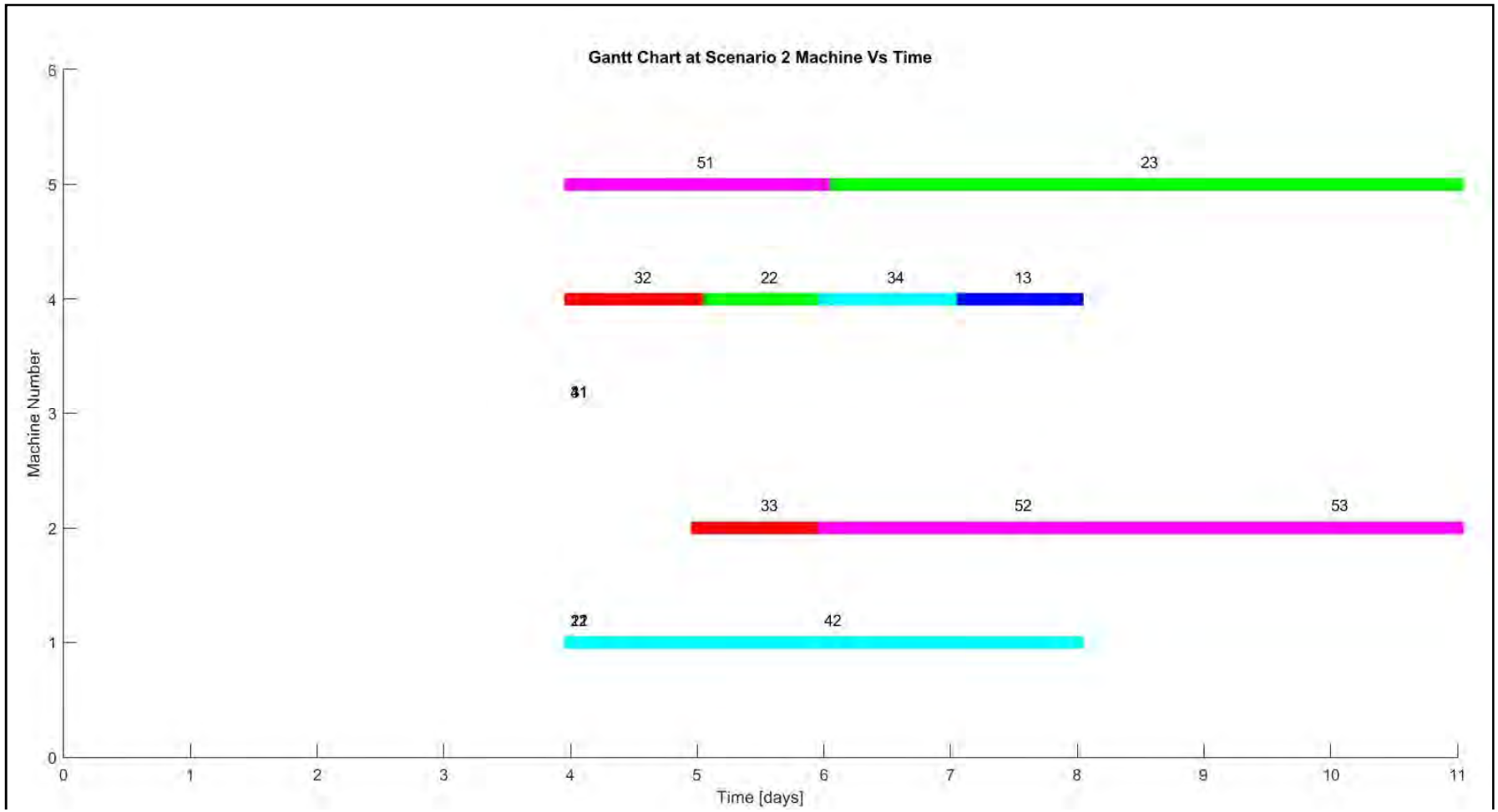


Figure 6.12: Gantt chart at scenario 2 of stage 2-Machine vs. Time

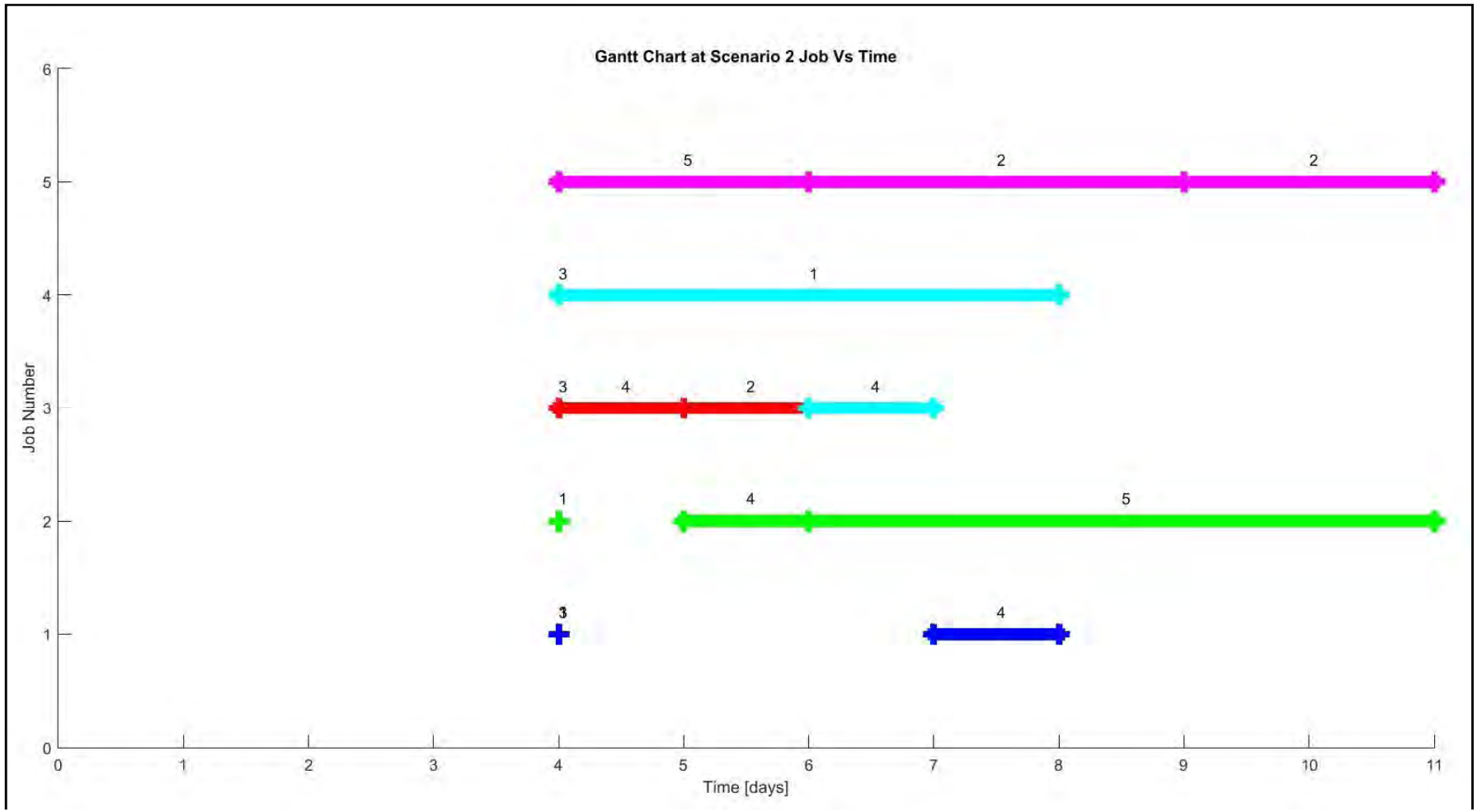


Figure 6.13: Gantt chart at scenario 2 of stage 2-Job vs. Time

From the two stages of observation it can be concluded that the model is a validated one. The objective functions plot shows the convergence to the minimum value. Close observation of the output data and Gantt charts reveals that there are only few changes occurred in the machine assignment and positioning the operations in stage 2. So the proposed mathematical model is stable and robust as well.

6.1.2 Optimization using Genetic Algorithm*:*

The application of genetic algorithm involves similar steps as in Branch and Cut algorithm accepts few changes. The steps of genetic algorithm was described in detail in chapter V

I. Reading the input data
II. Variable generation according to the model in terms of x
III. Modification of equality constraints as described in chapter V
IV. Developing the coefficient matrices for all equations
V. Developing separate matrices for $A, b, lb, ub, intcon$
VI. Defining the fitness function
VII. Setting the genetic algorithm parameters like number of populations, number of generation, stopping condition, crossover and mutation options etc.
VIII. Running optimization
IX. Reading output result
X. Decoding the result to generate separate Gantt chart at different scenarios both for job and machine vs. the time to make it easily readable to the user.

The inputs are similar as in Branch and Cut algorithm. Here the outputs are shown only.

6.1.2.1 First stage optimization-Developing a Robust Schedule (GA):

Population: 285

Generation: 300

Crossover probability: 0.8

Fitness Selection Function: Proportional

6.1.2.1.1 Output of stage 1(GA):

Scenario-1:

Table 6.10: Results of scenario 1 at stage 1(GA)					
Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	1	1	2	4
1	2	7	2	4	8
1	3	11	1	4	14
2	1	16	1	2	5
2	2	25	5	5	10
2	3	26	1	4	11
3	1	33	3	6	6
3	2	37	2	1	8
3	3	44	4	2	11
3	4	49	4	1	14
4	1	53	3	2	7
4	2	60	5	2	12
5	1	64	4	0	6
5	2	70	5	0	10
5	3	74	4	0	13

Scenario-2:

Table 6.11: Results of scenario 1 at stage 1(GA)					
Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	1	1	1	1
1	2	8	3	1	3
1	3	14	4	1	11
2	1	16	1	2	3
2	2	24	4	1	10
2	3	28	3	2	12
3	1	35	5	4	9
3	2	39	4	1	11
3	3	42	2	1	12
3	4	47	2	3	15
4	1	51	1	5	6
4	2	57	2	2	10
5	1	65	5	0	0
5	2	68	3	0	11
5	3	74	4	0	11

The objective function plot and Gantt charts at different scenarios are shown below.

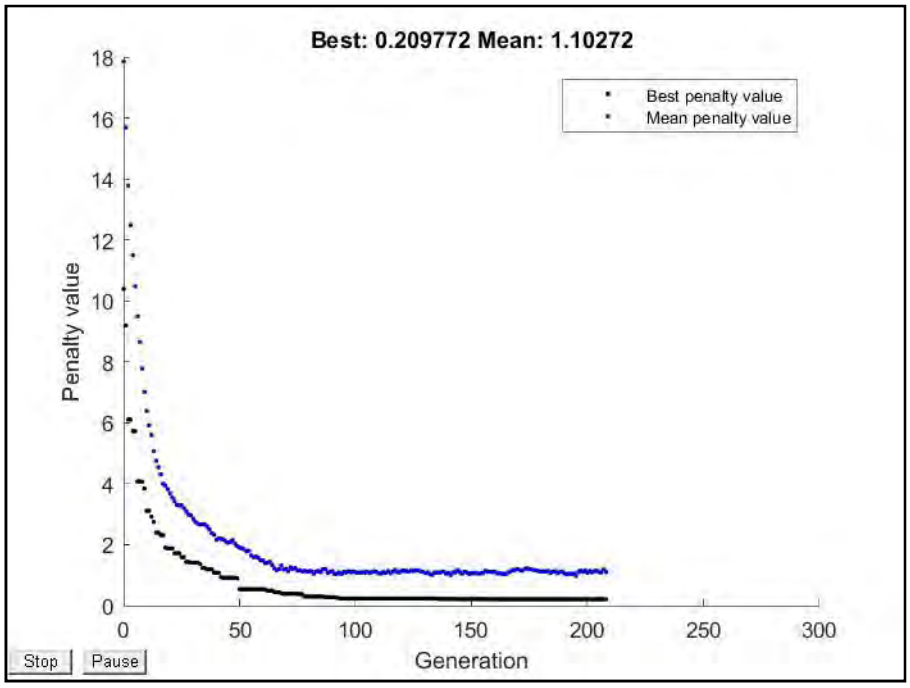


Figure 6.14: Objective versus Number of nodes at scenario 1of stage 1(GA)

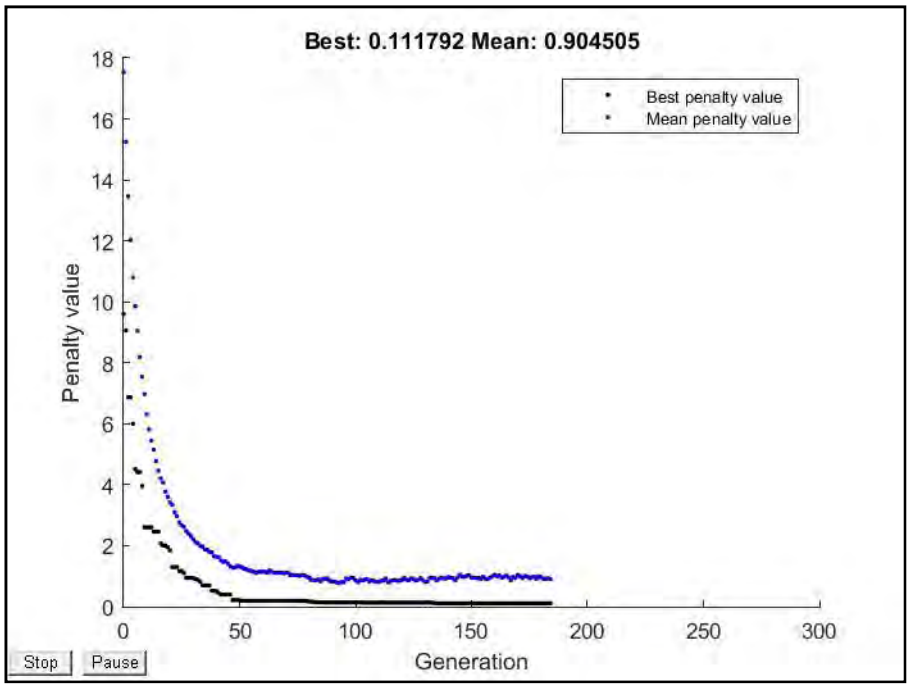


Figure 6.15: Objective versus Number of nodes at scenario 2of stage 1(GA)

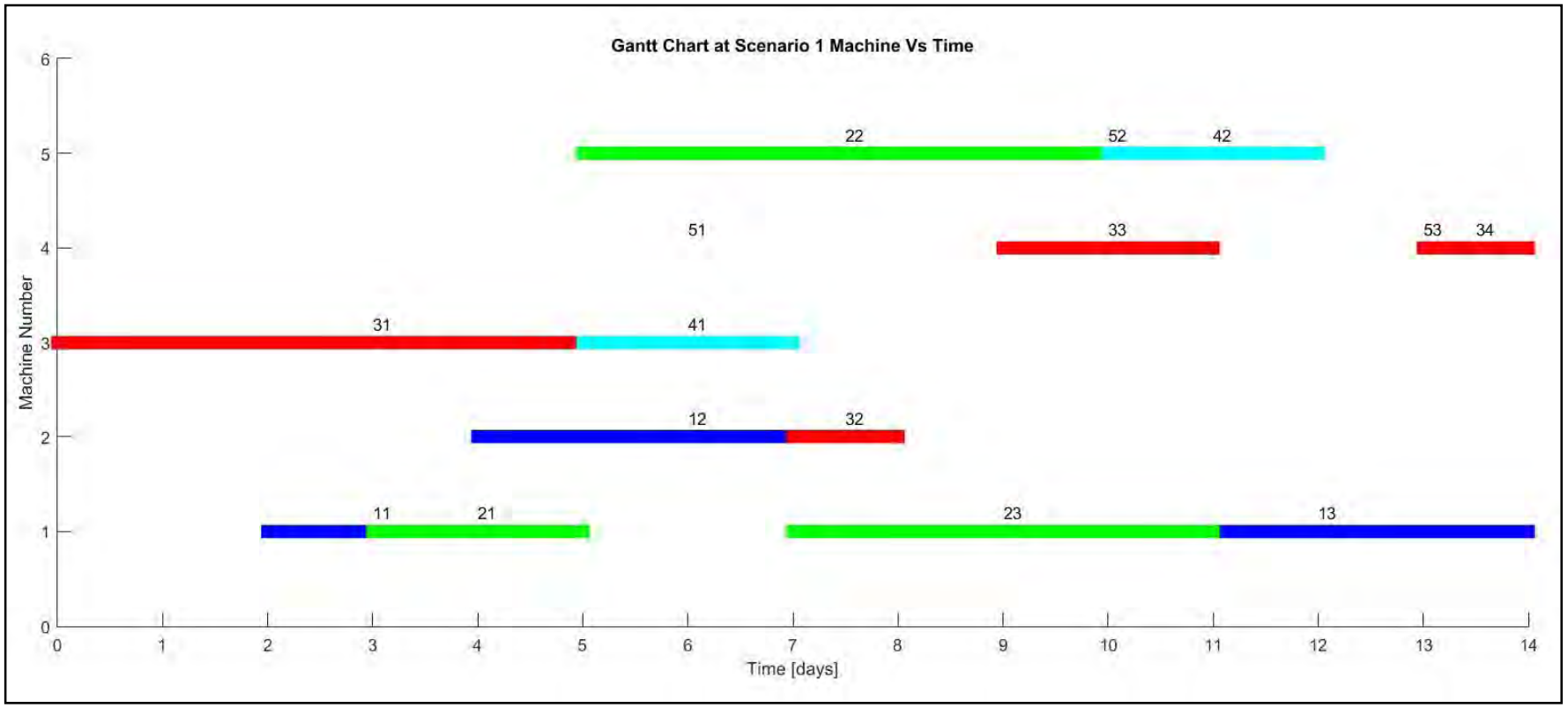


Figure 6.16: Gantt chart at scenario 1 of stage 1-Machine vs. Time (GA)

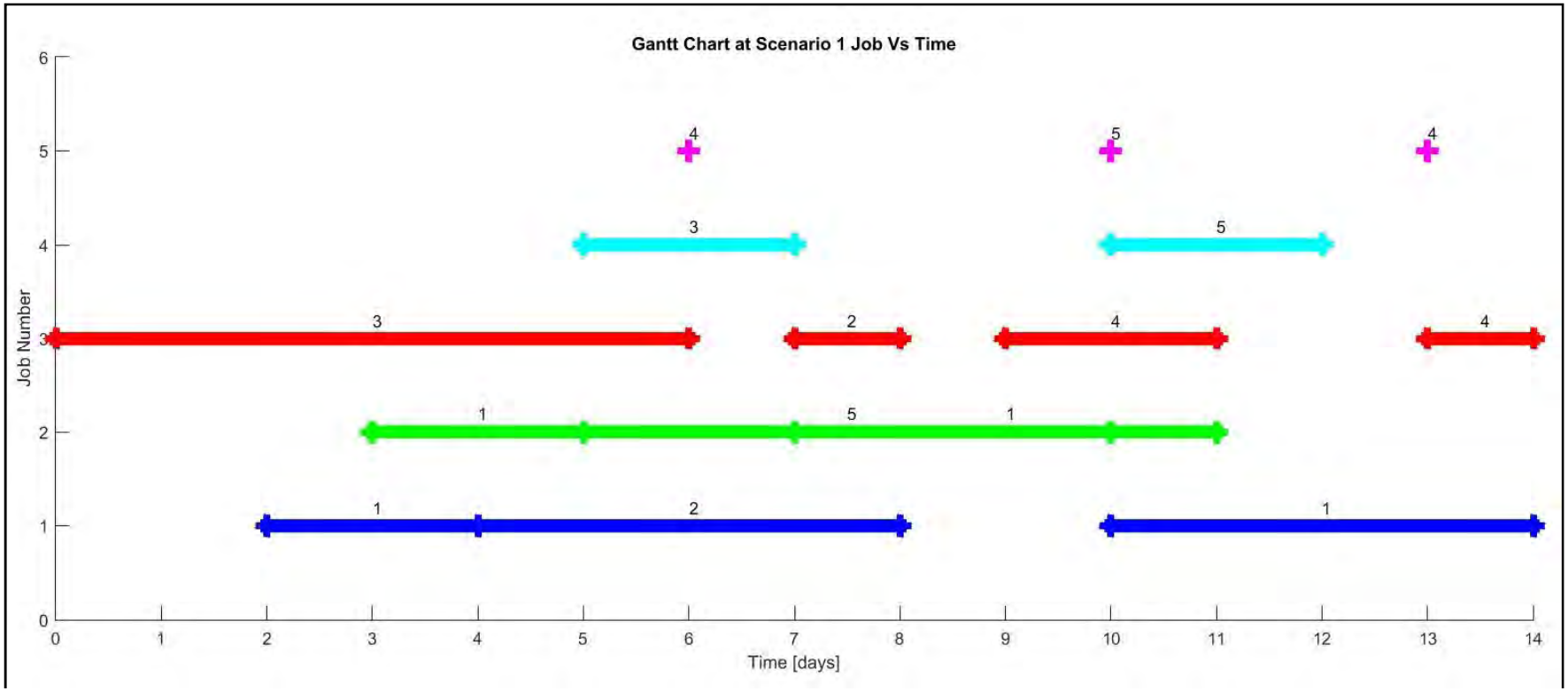


Figure 6.17: Gantt chart at scenario 1 of stage 1-Job vs. Time (GA)

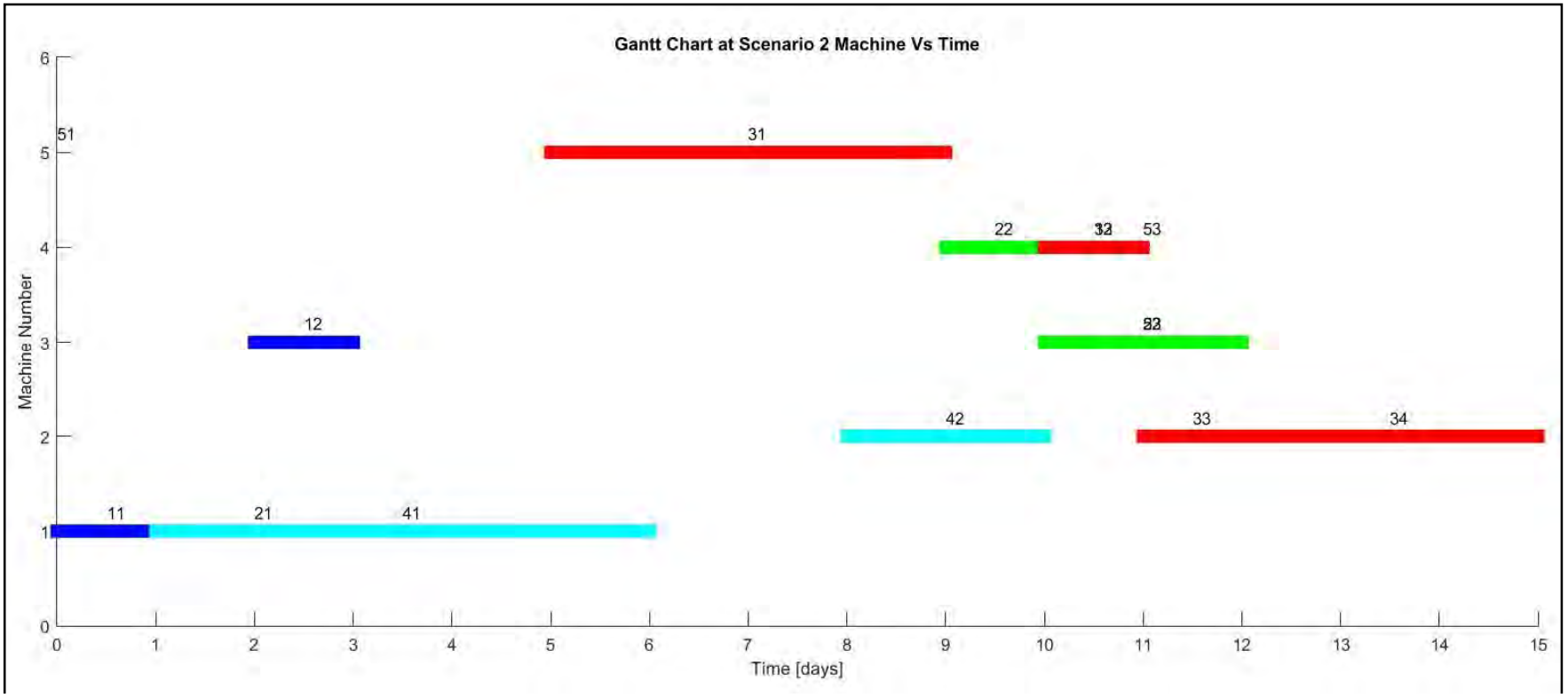


Figure 6.18: Gantt chart at scenario 2 of stage 1-Machine vs. Time (GA)

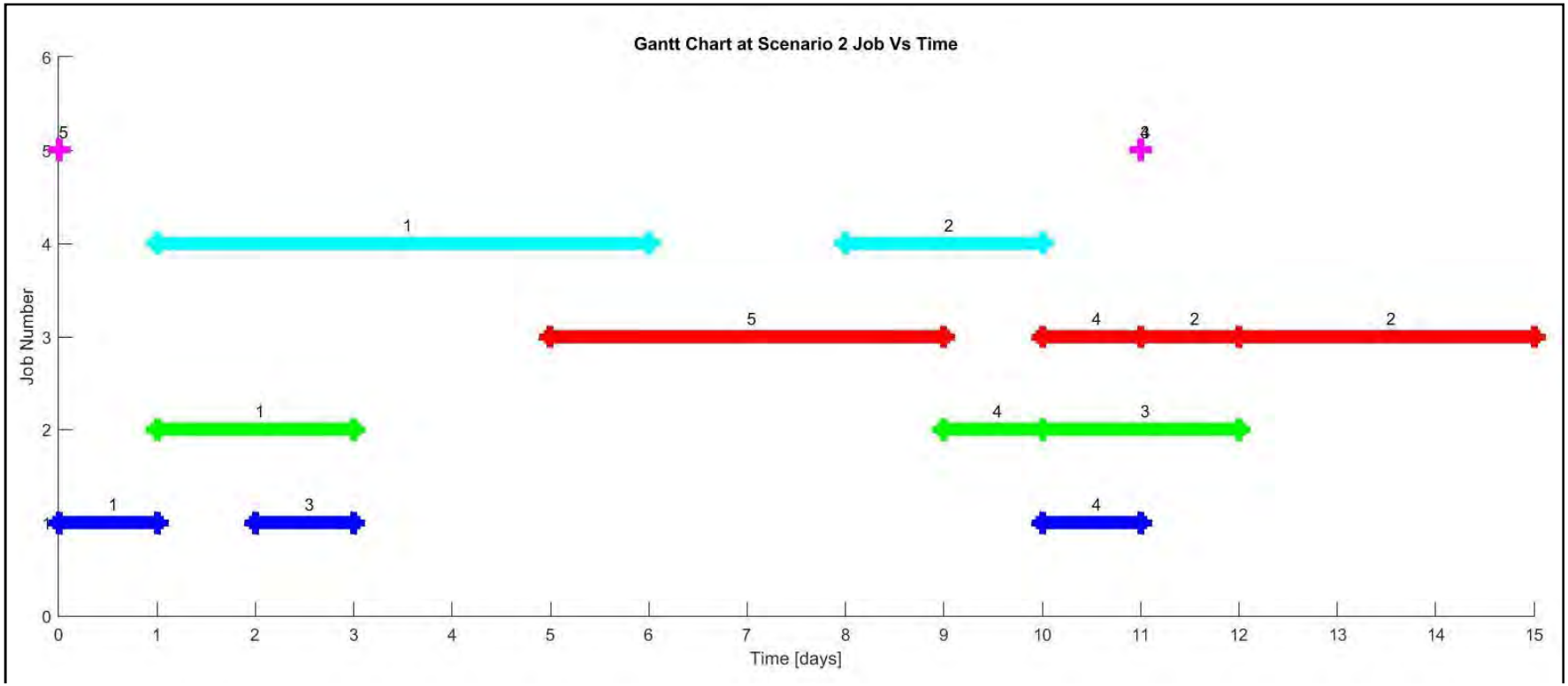


Figure 6.19: Gantt chart at scenario 2 of stage 1-Job vs. Time (GA)

6.1.2.2 Second stage optimization-Reactive Scheduling(GA):

The preprocessing of data for second stage of optimization is similar as in section 6.1.1.2

6.1.2.2.1 Output of stage 2(GA):

Scenario-1:

Table 6.12: Results of scenario 1 at stage 1(GA)

Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	1	1	0	6
1	2	6	1	5	14
1	3	15	5	5	27
2	1	17	2	5	8
2	2	21	1	5	21
2	3	28	3	4	28
3	1	34	4	0	7
3	2	37	2	1	11
3	3	41	1	2	17
3	4	49	4	1	30
4	1	53	3	2	9
4	2	59	4	1	29
5	1	65	5	2	5
5	2	67	2	3	11
5	3	72	2	2	16

Scenario-2:

Table 6.13: Results of scenario 1 at stage 1(GA)

Job No.	Operation No.	Decision variable no.	Machine assigned	Processing time (days)	Completion time (days)
1	1	4	4	0	7
1	2	8	3	1	18
1	3	14	4	1	23
2	1	20	5	0	7
2	2	23	3	7	18
2	3	28	3	2	28
3	1	32	2	5	10
3	2	37	2	3	17
3	3	42	2	1	22
3	4	47	2	3	28
4	1	52	2	0	6
4	2	57	2	2	21

5	1	61	1	5	9
5	2	69	4	7	22
5	3	75	5	4	30

Table 6.14: Objective function values (GA)		
Objective	Scenario-1	Scnario-2
Makespan	30	30
Variability of completions times of the operations	102	141
Variability of the makespan	16	15

The objective function plot and Gantt charts at different scenarios are shown below

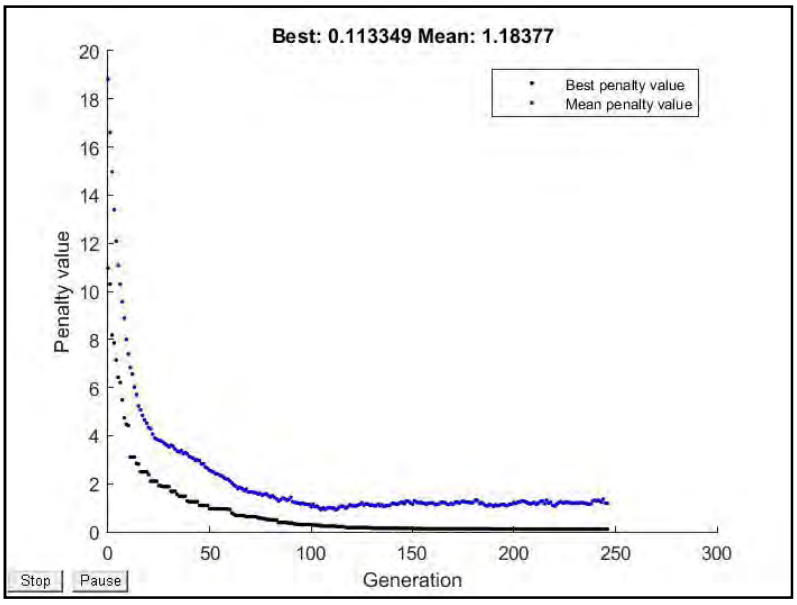


Figure 6.20: Objective versus Number of nodes at scenario 1of stage 2(GA)

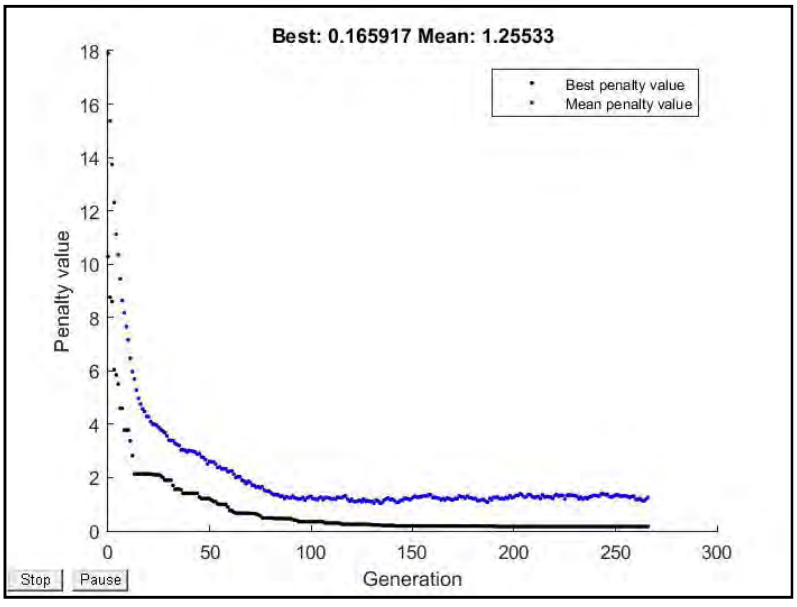


Figure 6.21: Objective versus Number of nodes at scenario 2of stage 2(GA)

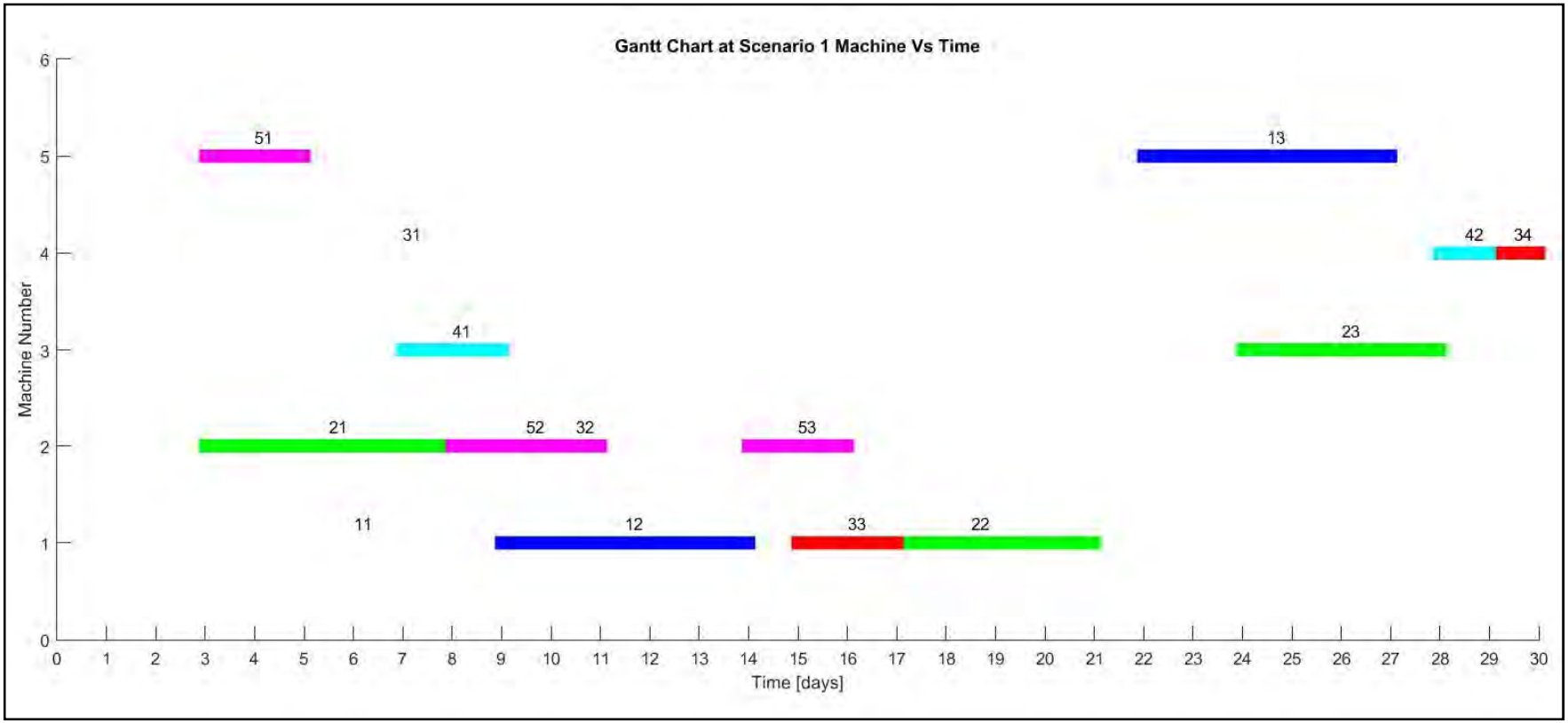


Figure 6.22: Gantt chart at scenario 1 of stage 2-Machine vs. Time (GA)

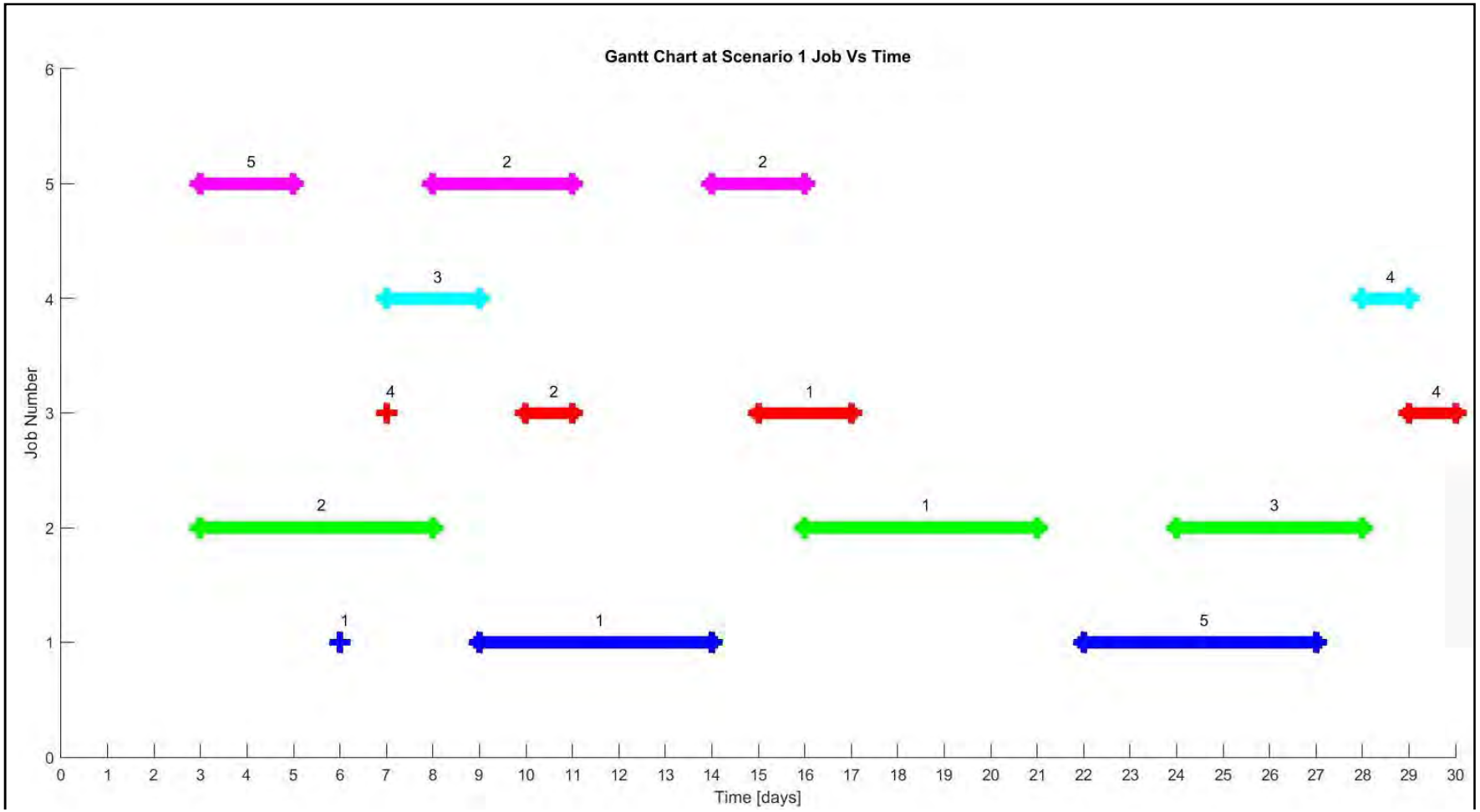


Figure 6.23: Gantt chart at scenario 1 of stage 2-Job vs. Time (GA)

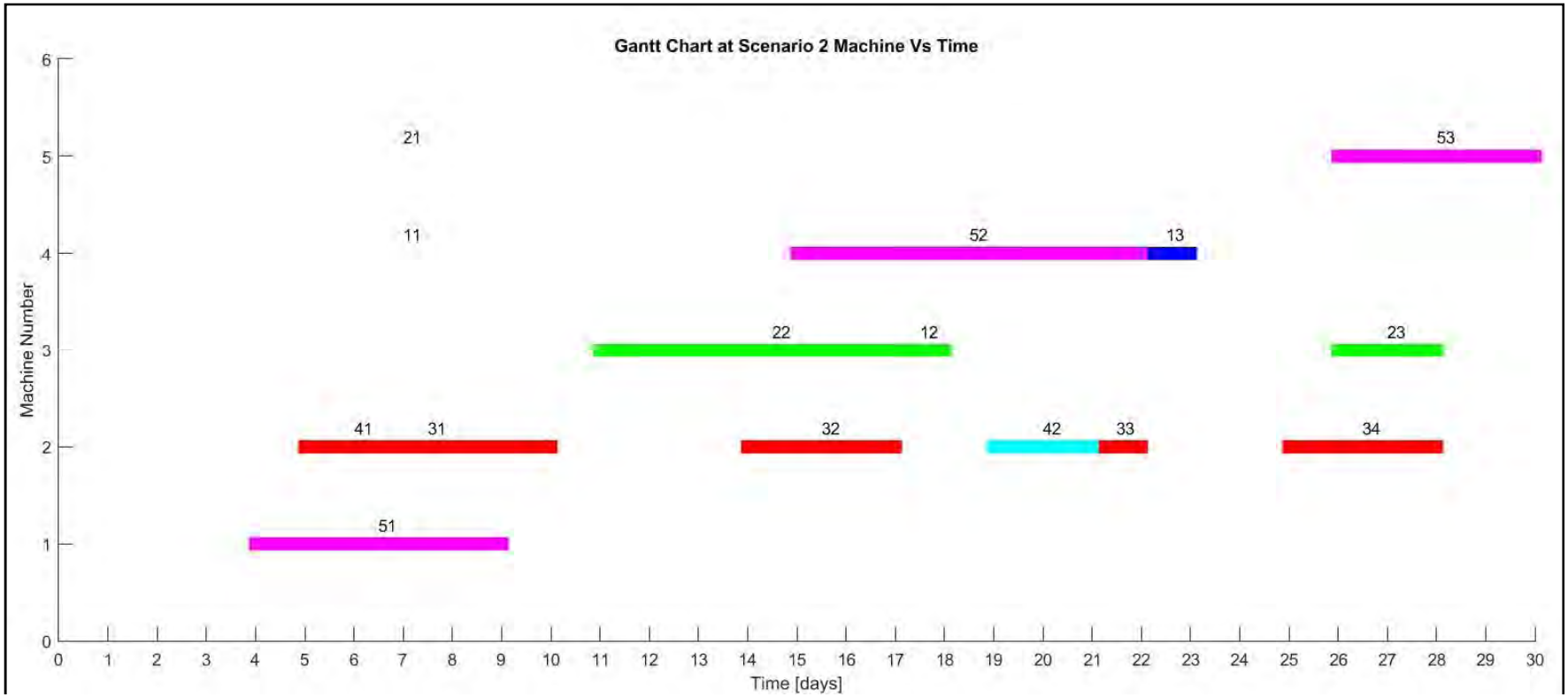


Figure 6.24: Gantt chart at scenario 2 of stage 2-Machine vs. Time (GA)

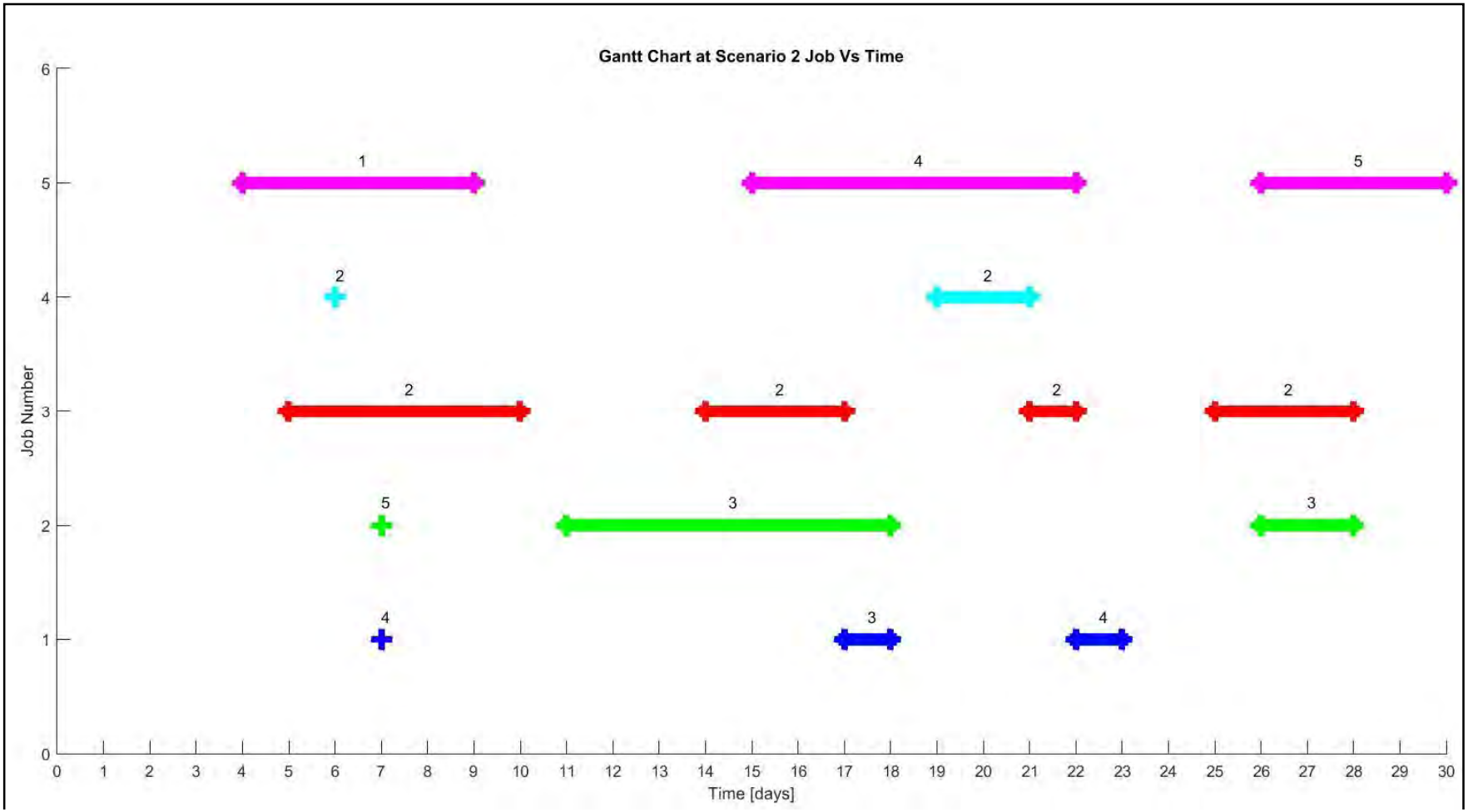


Figure 6.25: Gantt chart at scenario 2 of stage 2-Job vs. Time (GA)

Analyzing the two algorithm it can be concluded that Branch and Cut algorithm performs better in optimizing the schedule in both stages compared to Genetic Algorithm. One possible cause is the Branch and Cut algorithm search almost the entire solution space whereas Genetic Algorithm stops after achieving local optimal. But in case of problem with large number of instances Genetic Algorithm outperform the Branch and Cut method with respect to computation time (see appendix5 and 7).

CHAPTER VII

CONCLUSIONS AND FUTURE RESEARCH

7.1Conclusions

This study addresses a two stage flexible Job Shop scheduling and rescheduling problems encompassing multiple objectives and multiple machines with machine eligibilities. The multiple objectives of the scheduling problem are makespan, stability and Robustness. This problem belongs to the NP-hard problem class, which has a very high complexity resulting in very high computation time as the problem sizes are increased. The study implements the GA to determine a near optimal sequence from a collection of n jobs scheduled on a bank of identical parallel machines with machine eligibilities to minimize makespan and to maximize the schedule robustness and stability simultaneously. The near optimal sequence generated by the GA is then used as the initial schedule or input for the rescheduling problem. Moreover the problem is also solved by using branch and cut algorithm. Due to the application of the cutting plane the computational time is significantly improved. The incorporation of the unpredicted arrival of new job and interactive interface for schedule generation makes this work more adaptable to the real application. Upon the arrival of new job, the schedule will be revised and a new schedule will be generated considering the objective to minimize the variability of completion time of each job and variability of makespan in two stages. Therefore the proposed model is robust, stable and effective for the real life application.

7.2Future Research

There are some possible directions to which this research can be extended. In this thesis processing time data are deterministic under a scenario which reduces the computational effort and kept the model linear. Implementation of stochastic processing time can be done in future. Some objectives are not considered in this work like tardiness, critical machine workload etc. The work can be extended for the following objectives also. The problem is solved by MATLAB based solver which provides a built in framework for solving the problem. Improvement of algorithm like hybridization can be done to solve the in much less computational time then the

proposed methods. The machines in this problem are considered identical in terms of operation and speed. But in real scenario it may not be the case. Therefore partial flexible job shop problem can be developed. Moreover, no rescheduling penalty is considered in the model. Rescheduling may involve rerouting of jobs and new setups and fixtures. So the incorporation of rescheduling cost in the proposed model may be a possible future research.

References

[1] V. Roshanaei, M. S. Esfehani, and M. Zandieh, "Integrating non-preemptive open shops scheduling with sequence-dependent setup times using advanced metaheuristics," *Expert systems with applications,* vol. 37, pp. 259-266, 2010.

[2] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval research logistics quarterly,* vol. 1, pp. 61-68, 1954.

[3] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling,* vol. 12, pp. 417-431, 2009.

[4] H. Davoudpour and N. Azad, "Solving Multi-Objective Flexible Job Shop Scheduling Problems Using Immune Algorithm," *International Journal of Modern Science and Technology,* pp. 2325-2332, 2012.

[5] L. De Giovanni and F. Pezzella, "An improved genetic algorithm for the distributed and flexible job-shop scheduling problem," *European journal of operational research,* vol. 200, pp. 395-408, 2010.

[6] G. Vilcot and J.-C. Billaut, "A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem," *European Journal of Operational Research,* vol. 190, pp. 398-411, 2008.

[7] R. L. Daniels and P. Kouvelis, "Robust scheduling to hedge against processing time uncertainty in single-stage production," *Management Science,* vol. 41, pp. 363-376, 1995.

[8] S. V. Mehta, "Predictable scheduling of a single machine subject to breakdowns," *International Journal of Computer Integrated Manufacturing,* vol. 12, pp. 15-38, 1999.

[9] S. V. Mehta and R. M. Uzsoy, "Predictable scheduling of a job shop subject to breakdowns," *Robotics and Automation, IEEE Transactions on,* vol. 14, pp. 365-378, 1998.

[10] R. O'Donovan, R. Uzsoy, and K. N. McKay, "Predictable scheduling of a single machine with breakdowns and sensitive jobs," *International Journal of Production Research,* vol. 37, pp. 4217-4233, 1999.

[11] M. T. Jensen, "Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures," *Applied Soft Computing,* vol. 1, pp. 35-52, 2001.

[12] S. Goren and I. Sabuncuoglu, "Robustness and stability measures for scheduling: single-machine environment," *IIE Transactions,* vol. 40, pp. 66-83, 2008.

[13] S. Goren and I. Sabuncuoglu, "Optimization of schedule robustness and stability under random machine breakdowns and processing time variability," *IIE Transactions,* vol. 42, pp. 203-220, 2009.

[14] L. Liu, H.-y. Gu, and Y.-g. Xi, "Robust and stable scheduling of a single machine with random machine breakdowns," *The International Journal of Advanced Manufacturing Technology,* vol. 31, pp. 645-654, 2007.

[15] N. Al-Hinai and T. ElMekkawy, "Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm," *International Journal of Production Economics,* vol. 132, pp. 279-291, 2011.

[16] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of operations research,* vol. 1, pp. 117-129, 1976.

[17] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing,* vol. 45, pp. 369-375, 1990.

[18] B. MacCarthy and J. Liu, "Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling," *The International Journal of Production Research,* vol. 31, pp. 59-79, 1993.

[19] C. S. Shukla and F. F. Chen, "The state of the art in intelligent real-time FMS control: a comprehensive survey," *Journal of intelligent Manufacturing,* vol. 7, pp. 441-455, 1996.

[20] P. Cowling and M. Johansson, "Using real time information for effective dynamic scheduling," *European Journal of Operational Research,* vol. 139, pp. 230-244, 2002.

[21] M. L. Pinedo, *Scheduling: theory, algorithms, and systems*: Springer Science & Business Media, 2012.

[22] M. Pinedo, "On the Computational Complexity of Stochastic Scheduling Problems," in *Deterministic and Stochastic Scheduling*. vol. 84, M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan, Eds., ed: Springer Netherlands, 1982, pp. 355-365.

[23] R. H. Möhring, F. J. Radermacher, and G. Weiss, "Stochastic scheduling problems II-set strategies," *Zeitschrift für Operations Research,* vol. 29, pp. 65-104, 1985.

[24] R. Montemanni, "A mixed integer programming formulation for the total flow time single machine robust scheduling problem with interval data," *Journal of Mathematical Modelling and Algorithms,* vol. 6, pp. 287-296, 2007.

[25] P. Kouvelis, R. L. Daniels, and G. Vairaktarakis, "Robust scheduling of a two-machine flow shop with uncertain processing times," *IIE Transactions (Institute of Industrial Engineers),* vol. 32, pp. 421-432, 2000.

[26] X. Zuo, H. Mo, and J. Wu, "A robust scheduling method based on a multi-objective immune algorithm," *Information Sciences,* vol. 179, pp. 3359-3369, 2009.

[27] Y. Xia, B. Chen, and J. Yue, "Job sequencing and due date assignment in a single machine shop with uncertain processing times," *European Journal of Operational Research,* vol. 184, pp. 63-75, 2008.

[28] U. M. Al-Turki, J. Mittenthal, and M. Raghavachari, "The single-machine absolute-deviation early-tardy problem with random completion times," *Naval Research Logistics,* vol. 43, pp. 573-587, 1996.

[29] X. Cai and F. S. Tu, "Scheduling jobs with random processing times on a single machine subject to stochastic breakdowns to minimize early-tardy penalties," *Naval Research Logistics,* vol. 43, pp. 1127-1146, 1996.

[30] L. Liu, H. Y. Gu, and Y. G. Xi, "Robust and stable scheduling of a single machine with random machine breakdowns," *International Journal of Advanced Manufacturing Technology,* vol. 31, pp. 645-654, 2007.

[31] M. Sevaux and K. Sörensen, "A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates," *4OR,* vol. 2, pp. 129-147, 2004.

[32] E. S. Byeon, S. D. Wu, and R. H. Storer, "Decomposition heuristics for robust job-shop scheduling," *IEEE Transactions on Robotics and Automation,* vol. 14, pp. 303-313, 1998.

[33] E. Kutanoglu and S. D. Wu, "Improving scheduling robustness via preprocessing and dynamic adaptation," *IIE Transactions (Institute of Industrial Engineers),* vol. 36, pp. 1107-1124, 2004.

[34] E. Kutanoglu and S. D. Wu, "Improving Schedule Robustness Via Stochastic Analysis and Dynamic Adaptation," *IMSE Technical Report 98T-001,* 1998.

[35] S. D. Wu, E. S. Byeon, and R. H. Storer, "A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness," *Operations Research,* vol. 47, pp. 113-124, 1999.

[36] R. Shafaei and P. Brunn, "Workshop scheduling using practical (inaccurate) data - Part 3: A framework to integrate job releasing, routing and scheduling functions to create a robust predictive schedule," *International Journal of Production Research,* vol. 38, pp. 85-99, 2000.

[37] R. Shafaei and P. Brunn, "Workshop scheduling using practical (inaccurate) data. Part 2: An investigation of the robustness of scheduling rules in a dynamic and stochastic environment," *International Journal of Production Research,* vol. 37, pp. 4105-4117, 1999.

[38] P. I. Cowling, D. Ouelhadj, and S. Petrovic, "Dynamic scheduling of steel casting and milling using multi-agents," *Production Planning and Control,* vol. 15, pp. 178-188, 2004.

[39] N. Policella, A. Oddi, S. F. Smith, and A. Cesta, "Generating robust partial order schedules," in *Lecture Notes in Computer Science (including subseries*

Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) vol. 3258, ed, 2004, pp. 496-511.

[40] N. Policella, A. Cesta, A. Oddi, and S. F. Smith, "Schedule robustness through broader solve and robustify search for partial order schedules," *Proceedings of AI (*)IA 2005, Lecture Notes in Artificial Intelligence,* vol. 3673, pp. 160-172, 2005.

[41] V. J. Leon, S. D. Wu, and R. H. Storer, "Robustness measures and robust scheduling for job shops," *IIE Transactions,* vol. 26, pp. 32-43, 1994.

[42] S. R. Lawrence and E. C. Sewell, "Heuristic, optimal, static, and dynamic schedules when processing times are uncertain," *Journal of Operations Management,* vol. 15, pp. 71-82, 1997.

[43] I. Sabuncuoglu, "Rescheduling frequency in an FMS with uncertain processing times and unreliable machines," *Journal of Manufacturing Systems,* vol. 18, pp. 268-282, 1999.

[44] S. V. Mehta and R. M. Uzsoy, "Predictable scheduling of a job shop subject to breakdowns," *IEEE Transactions on Robotics and Automation,* vol. 14, pp. 365-378, 1998.

[45] M. T. Jensen, "Generating robust and flexible job shop schedules using genetic algorithms," *IEEE Transactions on Evolutionary Computation,* vol. 7, pp. 275-288, 2003.

[46] D. C. Mattfeld, "Evolutionary search and the job shop," *Production and Logistics,* 1996.

[47] Z. Laslo, D. Golenko-Ginzburg, and B. Keren, "Optimal booking of machines in a virtual job-shop with stochastic processing times to minimize total machine rental and job tardiness costs," *International Journal of Production Economics,* vol. 111, pp. 812-821, 2008.

[48] A. Anglani, A. Grieco, E. Guerriero, and R. Musmanno, "Robust scheduling of parallel machines with sequence-dependent set-up costs," *European Journal of Operational Research,* vol. 161, pp. 704-720, 2005.

[49] Z. Bouyahia, M. Bellalouna, P. Jaillet, and K. Ghedira, "A priori parallel machines scheduling," *Computers & Industrial Engineering,* 2009.

[50] B. Guo and Y. Nonaka, "Rescheduling and optimization of schedules considering machine failures," *International Journal of Production Economics,* vol. 60, pp. 503-513, 1999.

[51] N. M. Matsveichuk, Y. N. Sotskov, N. G. Egorova, and T. C. Lai, "Schedule execution for two-machine flow-shop with interval processing times," *Mathematical and Computer Modelling,* vol. 49, pp. 991-1011, 2009.

[52] X. Qi, J. F. Bard, and G. Yu, "Disruption management for machine scheduling: The case of SPT schedules," *International Journal of Production Economics,* vol. 103, pp. 166-184, 2006.

[53] C. Artigues, P. Michelon, and S. Reusser, "Insertion techniques for static and dynamic resource-constrained project scheduling," *European Journal of Operational Research,* vol. 149, pp. 249-267, 2003.

[54] M. Surico, U. Kaymak, D. Naso, and R. Dekker, *Hybrid Meta-Heuristic for Robust Scheduling. ERIM Report Series Reference No. ERS-2006-018-LIS,* 2006.

[55] H. Chtourou and M. Haouari, "A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling," *Computers and Industrial Engineering,* vol. 55, pp. 183-194, 2008.

[56] O. Lambrechts, E. Demeulemeester, and W. Herroelen, "A tabu search procedure for developing robust predictive project schedules," *International Journal of Production Economics,* vol. 111, pp. 493-508, 2008.

[57] R. Rangsaritratsamee, W. G. Ferrell Jr, and M. B. Kurz, "Dynamic rescheduling that simultaneously considers efficiency and stability," *Computers and Industrial Engineering,* vol. 46, pp. 1-15, 2004.

[58] P. Fattahi and A. Fallahi, "Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability," *CIRP Journal of Manufacturing Science and Technology,* vol. 2, pp. 114-123, 2010.

[59] P. Fattahi, M. Saidi Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *Journal of Intelligent Manufacturing,* vol. 18, pp. 331-342, 2007.

[60] I. Mahdavi, B. Shirazi, and M. Solimanpur, "Development of a simulation-based decision support system for controlling stochastic flexible job shop manufacturing systems," *Simulation Modelling Practice and Theory,* vol. 18, pp. 768-786, 2010.

[61] V. Vinod and R. Sridharan, "Development and analysis of scheduling decision rules for a dynamic flexible job shop production system: A simulation study," *International Journal of Business Performance Management,* vol. 11, pp. 43-71, 2009.

[62] V. Vinod and R. Sridharan, "Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system," *International Journal of Production Economics,* vol. 129, pp. 127-146, 2011.

[63] A. J. Davenport and J. C. Beck, "A survey of techniques for scheduling with uncertainty," *A Survey of Techniques for Scheduling with Uncertainty,* 2000.

[64] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: A review and some future

directions," *European Journal of Operational Research,* vol. 161, pp. 86-110, 2005.

[65] W. Herroelen and R. Leus, "Project scheduling under uncertainty: Survey and research potentials," *European Journal of Operational Research,* vol. 165, pp. 289-306, 2005.

[66] J. Mula, R. Poler, J. Garcia-Sabater, and F. C. Lario, "Models for production planning under uncertainty: A review," *International journal of production economics,* vol. 103, pp. 271-285, 2006.

[67] S. Cavalieri and S. Terzi, "Proposal of a performance measurement system for the evaluation of scheduling solutions," *International Journal of Manufacturing Technology and Management,* vol. 8, pp. 248-263, 2006.

[68] M. T. Jensen, "Robust and flexible scheduling with evolutionary computation," *Robust and Flexible Scheduling with Evolutionary Computation,* 2001.

[69] S. Gören, *Robustness and Stability for Scheduling Policies in A Single Machine Environment,* 2002.

[70] S. D. Wu, R. H. Storer, and P. C. Chang, "One-machine rescheduling heuristics with efficiency and stability as criteria," *Computers and Operations Research,* vol. 20, pp. 1-14, 1993.

[71] R. J. Abumaizar and J. A. Svestka, "Rescheduling job shops under random disruptions," *International Journal of Production Research,* vol. 35, pp. 2065-2082, 1997.

[72] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica: Journal of the Econometric Society,* pp. 497-520, 1960.

[73] R. J. Dakin, "A tree-search algorithm for mixed integer programming problems," *The Computer Journal,* vol. 8, pp. 250-255, 1965.

[74] E. Balas, "An additive algorithm for solving linear programs with zero-one variables," *Operations Research,* vol. 13, pp. 517-546, 1965.

[75] E. Lee and J. Mitchell, "Integer programming" in *Encyclopedia of Optimization*, ed: Springer US, 2001, pp. 1049-1059.

[76] J. Mitchell, "Integer programming: branch and cut algorithms Integer Programming: Branch and Cut Algorithms," in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds., ed: Springer US, 2009, pp. 1643-1650.

[77] R. Martí and G. Reinelt, "Branch-and-Bound," in *The Linear Ordering Problem*. vol. 175, ed: Springer Berlin Heidelberg, 2011, pp. 85-94.

[78] I. Androulakis, "MINLP: branch and bound global optimization algorithm MINLP: Branch and Bound Global Optimization Algorithm," in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds., ed: Springer US, 2009, pp. 2132-2138.

[79] G. L. Nemhauser and L. A. Wolsey, "Integer and Combinatorial Optimization. Interscience Series in Discrete Mathematics and Optimization," *ed: John Wiley & Sons,* 1988.

[80] H. A. Eiselt and C. L. Sandblom, "Branch and Bound Methods," in *Integer Programming and Network Models*, ed: Springer Berlin Heidelberg, 2000, pp. 205-228.

[81] R. E. Gomory, "An algorithm for integer solutions to linear programs," *Recent advances in mathematical programming,* vol. 64, pp. 260-302, 1963.

[82] M. Grötschel and O. Holland, "Solution of large-scale symmetric travelling salesman problems," *Mathematical Programming,* vol. 51, pp. 141-202, 1991.

[83] M. Padberg and G. Rinaldi, "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems," *SIAM review,* vol. 33, pp. 60-100, 1991.

[84] G. Stecco, Jean-Fran, O. Cordeau, and E. Moretti, "A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times," *Comput. Oper. Res.,* vol. 35, pp. 2635-2655, 2008.

[85] D. Gupta, "Branch and bound technique for three stage flow shop scheduling problem including breakdown interval and transportation time," *Journal of Information Engineering and Applications,* vol. 2, pp. 24-29, 2012.

[86] H. Crowder, E. L. Johnson, and M. Padberg, "Solving large-scale zero-one linear programming problems," *Operations Research,* vol. 31, pp. 803-834, 1983.

[87] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj, "Gomory cuts revisited," *Operations Research Letters,* vol. 19, pp. 1-9, 1996.

[88] E. Balas, S. Ceria, and G. Cornuéjols, "Mixed 0-1 programming by lift-and-project in a branch-and-cut framework," *Management Science,* vol. 42, pp. 1229-1246, 1996.

[89] L. A. Wolsey, *Integer programming* vol. 42: Wiley New York, 1998.

[90] A. Schrijver, "Theory of integer and linear programming," ed: Wiley, Chichester, 1986.

[91] J. E. Mitchell, "Branch-and-cut algorithms for combinatorial optimization problems," *Handbook of applied optimization,* pp. 65-77, 2002.

[92] R. Gomory, "An algorithm for integer solutions to linear programs, Princeton IBM Math," *Report (Nov. 1958),* 1958.

[93] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "On a linear-programming, combinatorial approach to the traveling-salesman problem," *Operations Research,* vol. 7, pp. 58-66, 1959.

[94] H. A. Taha, "Integer Programming-Theory," *Applications and Computations"-Academic Press-NY-1975,* 1975.

[95] V. Bowman and G. Nemhauser, "A finiteness proof for modified Dantzig cuts in integer programming," *Naval Research Logistics Quarterly,* vol. 17, pp. 309-313, 1970.

[96] V. Chvátal, "Edmonds polytopes and a hierarchy of combinatorial problems," *Discrete mathematics,* vol. 4, pp. 305-337, 1973.

[97] M. Neubauer, "Production scheduling and genetic algorithms," in *Information Management in Computer Integrated Manufacturing*. vol. 973, H. Adelsberger, J. Lažanský, and V. Mařík, Eds., ed: Springer Berlin Heidelberg, 1995, pp. 563-582.

[98] N. Jawahar, P. Aravindan, and S. G. Ponnambalam, "A genetic algorithm for scheduling flexible manufacturing systems," *The International Journal of Advanced Manufacturing Technology,* vol. 14, pp. 588-607, 1998/08/01 1998.

[99] C. Zhang, Y. Rao, and P. Li, "An effective hybrid genetic algorithm for the job shop scheduling problem," *The International Journal of Advanced Manufacturing Technology,* vol. 39, pp. 965-974, 2008/11/01 2008.

[100] Y. Wang, N. Xiao, H. Yin, E. Hu, C. Zhao, and Y. Jiang, "A two-stage genetic algorithm for large size job shop scheduling problems," *The International Journal of Advanced Manufacturing Technology,* vol. 39, pp. 813-820, 2008/11/01 2008.

[101] J. Gao, M. Gen, and L. Sun, "Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm," *Journal of Intelligent Manufacturing,* vol. 17, pp. 493-507, 2006/08/01 2006.

[102] I. Driss, K. Mouss, and A. Laggoun, "A new genetic algorithm for flexible job-shop scheduling problems," *Journal of Mechanical Science and Technology,* vol. 29, pp. 1273-1281, 2015/03/01 2015.

[103] A. Noorul Haq, K. Balasubramanian, B. Sashidharan, and R. B. Karthick, "Parallel line job shop scheduling using genetic algorithm," *The International Journal of Advanced Manufacturing Technology,* vol. 35, pp. 1047-1052, 2008/01/01 2008.

[104] Y. Wang, H. Yin, and J. Wang, "Genetic algorithm with new encoding scheme for job shop scheduling," *The International Journal of Advanced Manufacturing Technology,* vol. 44, pp. 977-984, 2009/10/01 2009.

[105] J. Holland, "Adaptation in natural and artificial systems, 1975," ed: Univ. of Michigan Press. A. Kershenbaum, P. Kermani, GA Grover,“MENTOR: An algorithm for mesh network topological optimization and routing”, IEEE Trans. Communications, 1991.

[106] L. Davis, "Genetic algorithms and simulated annealing," 1987.

[107] S. Forrest, "Genetic algorithms: principles of natural selection applied to computation," *Science,* vol. 261, pp. 872-878, 1993.

[108] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Artificial intelligence,* vol. 40, pp. 235-282, 1989.

[109] S. J. Louis and G. J. Rawlins, "Designer Genetic Algorithms: Genetic Algorithms in Structure Design," in *ICGA*, 1991, pp. 53-60.

[110] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," *Genetic algorithms and simulated annealing,* vol. 4, pp. 42-60, 1987.

[111] G. Zhang, X. Shao, P. Li, and L. Gao, "An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem," *Computers & Industrial Engineering,* vol. 56, pp. 1309-1318, 2009.

Appendix

Appendix 1: C++ Code for generating constraint equations

```
#include<bits/stdc++.h>
using namespace std;

const int MAXJOB=20+1;
const int MAXMACHINE=15+1;
const int MAXOP=4;
const int MAXEQN=80000;
const int MAXVAR=5000;

int no_machine,no_job;
int procTime[MAXJOB][MAXOP][MAXMACHINE],no_op[MAXJOB];

int assVarId[MAXJOB][MAXOP][MAXMACHINE];
int seqVarId[MAXJOB][MAXOP][MAXJOB][MAXOP];
int compTimeVarId[MAXJOB][MAXOP];
int maxCompTimeVarId;
int mat[MAXEQN][MAXVAR],col[MAXEQN],no_eqn,no_var,M;

FILE *fpr,*fpw,*fpe;

void print()
{
    int i,j,k;

    for( i=1; i<=no_eqn; i++ )
    {
        for(j=1; j<=no_var; j++)
        {
            fprintf(fpe,"%d ",mat[i][j]);
        }
        // fprintf(fpe,"=");

        fprintf(fpe,"%d ",col[i]);
        fprintf(fpe,"\n");
    }
}

int main()
{
    fpr=fopen("in.txt","r");
    fpw=fopen("log.txt","w");
    fpe=fopen("eqn.txt","w");

    fscanf( fpr,"%d",&no_machine );
    fprintf( fpw,"total number of machine %d\n",no_machine );

    fscanf( fpr,"%d",&no_job );
    fprintf( fpw,"total number of job %d\n",no_job );

    int i,j,k,l,h;

    for( j=1; j<=no_job; j++ )
    {
        fscanf( fpr,"%d",&no_op[j] );
        fprintf( fpw,"%dth job has %d operations\n",j,no_op[j] );


        for( i=1; i<=no_op[j]; i++ )
        {
            for( k=1; k<=no_machine; k++ )
            {
                fscanf( fpr,"%d",&procTime[j][i][k] ); /// ith job ith operation on kth machine
processing time
                fprintf( fpw,"%d ",procTime[j][i][k] );
            }
            fprintf( fpw,"\n" );
        }
        fprintf( fpw,"\n" );
    }

    fscanf( fpr,"%d",&M ); /// simplex Big M
    fprintf( fpw,"value of Big M %d \n",M );




    no_eqn=0;
    no_var=0;


    fprintf( fpw,"assignment variables start x%d\n",no_var+1 );
    /// first set of variable
    for( j=1; j<=no_job; j++ )
```

```
84      {
85          for( i=1; i<=no_op[j]; i++ )
86          {
87              for( k=1; k<=no_machine; k++ )
88              {
89                  assVarId[j][i][k]=++no_var;
90              }
91          }
92      }
93
94
95  //      cout<<assVarId[2][3][2]<<endl;
96
97
98      fprintf( fpw,"assignment variables end at x%d\n\n",no_var );
99
100
101
102     fprintf( fpw,"sequencing variables start x%d\n",no_var+1 );
103     for( j=1; j<=no_job; j++ )
104     {
105         for( i=1; i<=no_op[j]; i++ )
106         {
107             for( h=1; h<i; h++ )seqVarId[j][i][j][h]=++no_var;
108             for( l=1; l<=no_job; l++ )
109             {
110                 if(l==j)continue;
111                 for(h=1; h<=no_op[l]; h++)
112                 {
113                     seqVarId[j][i][l][h]=++no_var;
114                 }
115             }
116         }
117     }
118     fprintf( fpw,"sequencing variables end at x%d\n\n",no_var );
119
120 //      cout<<seqVarId[1][2][1][1]<<endl;
121
122
123     fprintf( fpw,"completion variables start at x%d\n",no_var+1 );
124     for( j=1; j<=no_job; j++ )
125     {
126         for(i=1; i<=no_op[j]; i++)
127         {
128             compTimeVarId[j][i]=++no_var;
129         }
130     }
131     fprintf( fpw,"completion variables end at x%d\n\n",no_var );
132
133     maxCompTimeVarId=++no_var;
134
135     fprintf( fpw,"max completion time variable is x%d\n\n",no_var );
136
137
138
139     ///1st set of eqn - Operations assignment constraint
140
141     for( j=1; j<=no_job; j++ )
142     {
143         for( i=1; i<=no_op[j]; i++ )
144         {
145             no_eqn++;
146             for( k=1; k<=no_machine; k++ )
147             {
148                 mat[no_eqn][ assVarId[j][i][k] ]=1;
149             }
150             col[no_eqn]=1;
151         }
152     }
153
154
155 for( j=1; j<=no_job; j++ )
156     {
157         for( i=1; i<=no_op[j]; i++ )
158         {
159             no_eqn++;
160             for( k=1; k<=no_machine; k++ )
161             {
162                 mat[no_eqn][ assVarId[j][i][k] ]=-1;
163             }
164             col[no_eqn]=-1;
165         }
166     }
167
```

```
168         //print();exit(0);
169
170
171         ///2nd set of eqn - Technical sequencing of operations constraint
172
173
174
175         for( j=1; j<=no_job; j++ )
176         {
177             for( i=2; i<=no_op[j]; i++ )
178             {
179                 no_eqn++;
180
181                 for( k=1; k<=no_machine; k++ )
182                 {
183                     mat[ no_eqn ][ assVarId[j][i][k] ]=procTime[j][i][k];
184                 }
185
186                 mat[no_eqn][ compTimeVarId[j][i] ]=-1;
187                 mat[no_eqn][ compTimeVarId[j][i-1] ]=+1;
188
189             }
190         }
191
192          //print();exit(0);
193
194
195         ///3rd set of eqn - 1st operation assignment
196
197
198
199         for( j=1; j<=no_job; j++ )
200         {
201             i=1;
202             {
203                 no_eqn++;
204
205                 for( k=1; k<=no_machine; k++ )
206                 {
207                     mat[ no_eqn ][ assVarId[j][i][k] ]=procTime[j][i][k];
208                 }
209
210                 mat[no_eqn][ compTimeVarId[j][i] ]=-1;
211
212             }
213         }
214
215          //print();exit(0);
216
217
218         ///4th set of eqn - Machine Availability Constraint
219
220
221         for( j=1; j<=no_job; j++ )
222         {
223             for( i=1; i<=no_op[j]; i++ )
224             {
225                 for( k=1; k<=no_machine; k++ )
226                 {
227                     no_eqn++;
228                     {
229                         mat[no_eqn][ assVarId[j][i][k] ]=procTime[j][i][k];
230                         mat[no_eqn][ compTimeVarId[j][i] ]=-1;
231                     }
232                 col[no_eqn]=0;
233                 }
234             }
235         }
236
237         //print();exit(0);
238
239
240          ///5th set of eqn - Sequencing of operations of different job
241
242         for( k=1; k<=no_machine; k++ )
243         {
244
245             for( j=1; j<=no_job; j++ )
246             {
247                 for( i=1; i<=no_op[j]; i++ )
248                 {
249                     for( h=1; h<i; h++ )
250                     {
251                         no_eqn++;
```

```
252
253                     l=j;
254
255                     mat[no_eqn][ seqVarId[j][i][l][h] ]=+M;
256                     mat[no_eqn][ assVarId[j][i][k] ]=+M;
257                     mat[no_eqn][ assVarId[l][h][k] ]=+M;
258                     mat[no_eqn][compTimeVarId[j][i]]=-1;
259                     mat[no_eqn][compTimeVarId[l][h]]=+1;
260
261                     col[no_eqn]=3*M-procTime[j][i][k];
262
263                 }
264
265                 for( l=1; l<=no_job; l++ )
266                 {
267                     if(l==j)continue;
268                     for(h=1; h<=no_op[l]; h++)
269                     {
270                         no_eqn++;
271                         mat[no_eqn][ seqVarId[j][i][l][h] ]=+M;
272                         mat[no_eqn][ assVarId[j][i][k] ]=+M;
273                         mat[no_eqn][ assVarId[l][h][k] ]=+M;
274                         mat[no_eqn][compTimeVarId[j][i]]=-1;
275                         mat[no_eqn][compTimeVarId[l][h]]=+1;
276
277                         col[no_eqn]=3*M-procTime[j][i][k];
278
279                     }
280                 }
281             }
282         }
283     }
284
285     //print();exit(0);
286
287
288     ///6th set of eqn - Sequencing of operations of different job
289
290     for( k=1; k<=no_machine; k++ )
291     {
292
293         for( j=1; j<=no_job; j++ )
294         {
295             for( i=1; i<=no_op[j]; i++ )
296             {
297                 for( h=1; h<i; h++ )
298                 {
299                     no_eqn++;
300
301                     l=j;
302
303                     mat[no_eqn][ seqVarId[j][i][l][h] ]=-M;
304                     mat[no_eqn][ assVarId[j][i][k] ]=+M;
305                     mat[no_eqn][ assVarId[l][h][k] ]=+M;
306                     mat[no_eqn][compTimeVarId[j][i]]=+1;
307                     mat[no_eqn][compTimeVarId[l][h]]=-1;
308
309                     col[no_eqn]=2*M-procTime[l][h][k];
310
311                 }
312
313                 for( l=1; l<=no_job; l++ )
314                 {
315                     if(l==j)continue;
316                     for(h=1; h<=no_op[l]; h++)
317                     {
318                         no_eqn++;
319
320                         mat[no_eqn][ seqVarId[j][i][l][h] ]=-M;
321                         mat[no_eqn][ assVarId[j][i][k] ]=+M;
322                         mat[no_eqn][ assVarId[l][h][k] ]=+M;
323                         mat[no_eqn][compTimeVarId[j][i]]=+1;
324                         mat[no_eqn][compTimeVarId[l][h]]=-1;
325
326                         col[no_eqn]=2*M-procTime[l][h][k];
327                     }
328                 }
329             }
330         }
331     }
332
333     //print();exit(0);
334
335     ///7th set of eqn - Maximum completion time constraint
```

```
336
337         for( j=1;j<=no_job;j++ )
338         {
339             i=no_op[j];
340             no_eqn++;
341
342             mat[no_eqn][compTimeVarId[j][i]]=1;
343             mat[no_eqn][maxCompTimeVarId]=-1;
344
345         }
346
347         print();
348
349
350
351
352         return 0;
353     }
354
```

Appendix 2: MATLAB Code for Branch and Cut Algorithm


```
clc
clear

J = 5; %Number of Jobs
M = 5; %Number of Machines
maxop = 3; %Number of maximum operation of a job
N = 285; %Number of Variables
Nb = 269; %Number of binary variable
Nop = N-Nb-1; %Number of operations
Nvoid = 3; %Number of void operations
nCol = N+1; %Number of Column
linewidth1 = 8; %Line-width of Gantt chart bars
linewidth2 = 8;

%Reading text file

    fid1 = fopen('eqn1.txt');  %Specify file name here
    %firstRowString = fgetl(fid);
    data1 = textscan(fid1, repmat('%f',1,nCol),'CollectOutput', true);
    fclose(fid1);

%Developing matrix

    cons1=data1{1};
    bleq=cons1(1:Nop,end);
    b1=cons1((Nop+1):end,end);
    Aleq=cons1(1:Nop,1:N);
    A1=cons1((Nop+1):end,1:N);
    intcon = 1:N; %Number of intigers
    f = zeros(1,N);
    f(end) = 1;
    lb1 = zeros(1,N);
    %lb2 = zeros(1,N);
    ub1=[ones(1,Nb),100*ones(1,(N-Nb))];

            options1 = optimoptions(@intlinprog,'OutputFcn',@savemilpsolutions,'PlotFcns',@
            optimplotmilp);
            [x1,f1,exitflag1] = intlinprog(f,intcon,A1,b1,Aleq,bleq,lb1,ub1,options1);


            pause(90);


%Reading text file
    fid2 = fopen('eqn2.txt');
    %firstRowString = fgetl(fid);
    data2 = textscan(fid2, repmat('%f',1,nCol),'CollectOutput', true);
    fclose(fid2);

%Developing matrix

    cons2=data2{1};
    b2eq=cons2(1:Nop,end);
```

```
    b2=cons2((Nop+1):end,end);
    A2eq=cons2(1:Nop,1:N);
    A2=cons2((Nop+1):end,1:N);
    %intcon = 1:N; %Number of intigers
    %f = zeros(1,N);
    %f(end) = 1;
    %lb1 = zeros(1,N);
    lb2 = zeros(1,N);
    ub2=[ones(1,Nb),100*ones(1,(N-Nb))];

        hold on
        options2 = optimoptions(@intlinprog,'OutputFcn',@savemilpsolutions,'PlotFcns',@
        optimplotmilp);
        [x2,f2,exitflag2] = intlinprog(f,intcon,A2,b2,A2eq,b2eq,lb2,ub2,options2);


        prob1 = 0.6; %Probability of scenario 1
        prob2 = 1-prob1; %Probability of scenario 2

    fval = prob1*f1 + prob2*f2 + prob1*(abs( f1 - (prob1*f1 + prob2*f2))) + prob2*(abs( f2 - (
    prob1*f1 + prob2*f2))); %objective function value

xout1 = x1; %Schedule of the scenario 1
comptime1 = [(1:Nop);(find(xout1,Nop,'first'))';xout1((N-Nop):(N-1))']';

%Reading operations data of scenario 1

        fid3 = fopen('opdata1.txt');
        %firstRowString = fgetl(fid);
        data3 = textscan(fid3, repmat('%f',1,M+2),'CollectOutput', true);
        fclose(fid3);
        opdata1 = data3 {1};
        proctime1 = opdata1(:,3:end);
        jobno = opdata1(:,1);
        opno = opdata1(:,2);
        trproctime1 = (proctime1)';
        for i = 1:Nop
            pt1(i)=[trproctime1(comptime1(i,2))]
            m1=rem (comptime1(i,2),M);
            if m1==0
                mach1(i)=M
            else mach1(i)=m1
            end
        end

        SlOUT = [(jobno)';(opno)';(comptime1(:,2))';(mach1);(pt1);(comptime1(:,3))']';

xout2 = x2; %Schedule of the scenario 2
comptime2 = [(1:Nop);(find(xout2,Nop,'first'))';xout2((N-Nop):(N-1))']';

%Reading operations data of scenario 2

        fid4 = fopen('opdata2.txt');
```

```
        %firstRowString = fgetl(fid);
        data4 = textscan(fid4, repmat('%f',1,M+2),'CollectOutput', true);
        fclose(fid4);
        opdata2 = data4 {1};
        proctime2 = opdata2(:,3:end);
        jobno = opdata2(:,1);
        opno = opdata2(:,2);
        trproctime2 = (proctime2)';
        for i = 1:Nop
            pt2(i)=[trproctime2(comptime2(i,2))]
            m1=rem (comptime2(i,2),M);
            if m1==0
                mach2(i)=M
            else mach2(i)=m1
            end
        end


    S2OUT = [(jobno)';(opno)';(comptime2(:,2))';(mach2);(pt2);(comptime2(:,3))']';


    Allcolor = {'-b' '-g' '-r' '-c' '-m' '-y' '-k' '-bo' '-go' '-ro' '-co' '-mo' '-yo' '-ko'
    '-wo'};
    for j = 1:J

        for i = 1:maxop
                colmat{i,j} = Allcolor {j}
        end
    end

    color = reshape(colmat,1,J*maxop);


%Remove color if operation number is not same for all jobs

%   color(4) = [];
%   color(8-1) = [];
%   color(15-2) = [];
%   color(16-3) = [];
%   color(20-4) = [];

figure
    for i=1:Nop
    StopTimes = S1OUT(i,6);
    StartTimes = StopTimes-S1OUT(i,5);
    Machine = S1OUT(i,4);
    opID = S1OUT(i,1)*10 + S1OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*Machine,color{i},'LineWidth',linewidth1);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(Machine+0.2),num2str(opID));
    end
    title('Gantt Chart at Scenario 1 Machine Vs Time');
```

```
    xlabel('Time [days]');
    ylabel('Machine Number');

figure
    for i=1:Nop
    StopTimes = S2OUT(i,6);
    StartTimes = StopTimes-S2OUT(i,5);
    Machine = S2OUT(i,4);
    opID = S2OUT(i,1)*10 + S2OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*Machine,color{i},'LineWidth',linewidth1);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(Machine+0.2),num2str(opID));
    end
    title('Gantt Chart at Scenario 2 Machine Vs Time');
    xlabel('Time [days]');
    ylabel('Machine Number');

    Allcolor = {'-bo' '-go' '-ro' '-co' '-mo' '-yo' '-ko' '-bo' '-go' '-ro' '-co' '-mo' '-yo'
    '-ko'};
    for j = 1:J

        for i = 1:maxop
                colmat{i,j} = Allcolor {j}
        end
    end

    color = reshape(colmat,1,J*maxop);
    %linewidth2 = 10;

%Remove color if operation number is not same for all jobs

%   color(4) = [];
%   color(8-1) = [];
%   color(15-2) = [];
%   color(16-3) = [];
%   color(20-4) = [];

figure
    for i=1:Nop
    StopTimes = S1OUT(i,6);
    StartTimes = StopTimes-S1OUT(i,5);
    jobno = S1OUT(i,1);
    %opID = S1OUT(i,1)*10 + S1OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*jobno,color{i},'LineWidth',linewidth2);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(jobno+0.2),num2str(S1OUT(i,4)));
    end
    title('Gantt Chart at Scenario 1 Job Vs Time');
    xlabel('Time [days]');
    ylabel('Job Number');
```

```
figure
    for i=1:Nop
    StopTimes = S2OUT(i,6);
    StartTimes = StopTimes-S2OUT(i,5);
    jobno = S2OUT(i,1);
    %opID = S2OUT(i,1)*10 + S2OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*jobno,color{i},'LineWidth',linewidth2);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(jobno+0.2),num2str(S2OUT(i,4)));
    end
    title('Gantt Chart at Scenario 2 Job Vs Time');
    xlabel('Time [days]');
    ylabel('Job Number');
```

Appendix 3: MATLAB Code for Genetic Algorithm

6/22/15 2:38 AM D:\Thesis\Thesis Bijoy...\makespan ga 45.m 1 of 5

```
clc
clear

J = 5; %Number of Jobs
M = 5; %Number of Machines
maxop = 4; %Number of maximum operation of a job
N = 285; %Number of Variables
Nb = 269; %Number of binary variable
Nop = N-Nb-1; %Number of operations
Nvoid = 3; %Number of void operations
nCol = N+1; %Number of Column
linewidth1 = 8; %Line-width of Gantt chart bars
linewidth2 = 8;
gen = 300; %Genetic Algorithm no of generations

%Reading constraint equations at scenario 1

    fid1 = fopen('eqn1.txt');  %Specify file name here
    %firstRowString = fgetl(fid);
    data1 = textscan(fid1, repmat('%f',1,nCol),'CollectOutput', true);
    fclose(fid1);

%Developing matrix

        cons1=data1{1};
%       bleq=cons1(1:Nop,end);
        b1=cons1(:,end);
%       Aleq=cons1(1:Nop,1:N);
        A1=cons1(:,1:N);
        intcon = 1:N; %Number of intigers
        % f = zeros(1,N);
        % f(end) = 1;
        lb1 = zeros(1,N);
        %lb2 = zeros(1,N);
        ub1=[ones(1,Nb),15*ones(1,(N-Nb))];


    opts1 = gaoptimset('PlotFcns',@gaplotbestf,'PopulationSize',N,'PopInitRange',[lb1;↙
ub1],...
                    'Generations',gen,'FitnessScalingFcn',@fitscalingprop);
        rng(1,'twister') % for reproducibility
        [x1,f1,exitflag1,Output1] = ga(@makespan_ga,N,A1,b1,[],[],lb1,ub1,[],intcon,↙
opts1);

        fprintf('The number of generations was : %d\n', Output1.generations);
        fprintf('The number of function evaluations was : %d\n', Output1.funccount);
        fprintf('The best function value found was : %g\n', f1);

%         pause(60);

%Reading constraint equations at scenario 2
```

```
    fid2 = fopen('eqn2.txt');
    %firstRowString = fgetl(fid);
    data2 = textscan(fid2, repmat('%f',1,nCol),'CollectOutput', true);
    fclose(fid2);

%Developing matrix

        cons2=data2{1};
%       b2eq=cons2(1:Nop,end);
        b2=cons2(:,end);
%       A2eq=cons2(1:Nop,1:N);
        A2=cons2(:,1:N);
        %intcon = 1:N; %Number of intigers
        %f = zeros(1,N);
        %f(end) = 1;
        %lb1 = zeros(1,N);
        lb2 = zeros(1,N);
        ub2=[ones(1,Nb),15*ones(1,(N-Nb))];

opts2 = gaoptimset('PlotFcns',@gaplotbestf,'PopulationSize',N,'PopInitRange',[lb2;↙
ub2],...
                   'Generations',gen,'FitnessScalingFcn',@fitscalingprop);
        [x2,f2,exitflag2,Output2] = ga(@makespan_ga,N,A2,b2,[],[],lb2,ub2,[],intcon,↙
opts2);

        fprintf('The number of generations was : %d\n', Output2.generations);
        fprintf('The number of function evaluations was : %d\n', Output2.funccount);
        fprintf('The best function value found was : %g\n', f2);

% prob1 = 0.6; %Probability of scenario 1
% prob2 = 1-prob1; %Probability of scenario 2
% f = prob1*f1 + prob2*f2 + prob1 ( f1 - (prob1*f1 + prob2*f2)) + prob2 ( f2 - (prob1*f1↙
+ prob2*f2));

    xout1 = x1; %Schedule of the scenario 1
    comptime1 = [(1:Nop);(find(xout1,Nop,'first'));xout1((N-Nop):(N-1))]';

    xout2 = x2; %Schedule of the scenario 2
    comptime2 = [(1:Nop);(find(xout2,Nop,'first'));xout2((N-Nop):(N-1))]';


%Reading operations data of scenario 1

            fid3 = fopen('opdata1.txt');
            %firstRowString = fgetl(fid);
            data3 = textscan(fid3, repmat('%f',1,M+2),'CollectOutput', true);
            fclose(fid3);
            opdata1 = data3 {1};
            proctime1 = opdata1(:,3:end);
            jobno = opdata1(:,1);
```

```
            opno = opdata1(:,2);
            trproctime1 = (proctime1)';
            for i = 1:Nop
                pt1(i)=[trproctime1(comptime1(i,2))]
                m1=rem (comptime1(i,2),M);
                if m1==0
                    mach1(i)=M
                else mach1(i)=m1
                end
            end

            S1OUT = [(jobno)';(opno)';(comptime1(:,2))';(mach1);(pt1);(comptime1(:,↙
3))']';


%Reading operations data of scenario 2

        fid4 = fopen('opdata2.txt');
        %firstRowString = fgetl(fid);
        data4 = textscan(fid4, repmat('%f',1,M+2),'CollectOutput', true);
        fclose(fid4);
        opdata2 = data4 {1};
        proctime2 = opdata2(:,3:end);
        jobno = opdata2(:,1);
        opno = opdata2(:,2);
        trproctime2 = (proctime2)';
        for i = 1:Nop
            pt2(i)=[trproctime2(comptime2(i,2))]
            m1=rem (comptime2(i,2),M);
            if m1==0
                mach2(i)=M
            else mach2(i)=m1
            end
        end


    S2OUT = [(jobno)';(opno)';(comptime2(:,2))';(mach2);(pt2);(comptime2(:,3))']';


      Allcolor = {'-b' '-g' '-r' '-c' '-m' '-y' '-k' '-bo' '-go' '-ro' '-co' '-mo' '-yo'↙
'-ko' '-wo'};
        for j = 1:J

            for i = 1:maxop
                    colmat{i,j} = Allcolor {j}
            end
        end

        color = reshape(colmat,1,J*maxop);

    %Remove color if operation number is not same for all jobs
```

```
        color(4) = [];
        color(8-1) = [];
        color(15-2) = [];
        color(16-3) = [];
        color(20-4) = [];


    figure
    for i=1:Nop
    StopTimes = S1OUT(i,6);
    StartTimes = StopTimes-S1OUT(i,5);
    Machine = S1OUT(i,4);
    opID = S1OUT(i,1)*10 + S1OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*Machine,color{i},'LineWidth',linewidth1);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(Machine+0.2),num2str(opID));
    end
    title('Gantt Chart at Scenario 1 Machine Vs Time');
    xlabel('Time [days]');
    ylabel('Machine Number');

figure
    for i=1:Nop
    StopTimes = S2OUT(i,6);
    StartTimes = StopTimes-S2OUT(i,5);
    Machine = S2OUT(i,4);
    opID = S2OUT(i,1)*10 + S2OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*Machine,color{i},'LineWidth',linewidth1);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(Machine+0.2),num2str(opID));
    end
    title('Gantt Chart at Scenario 2 Machine Vs Time');
    xlabel('Time [days]');
    ylabel('Machine Number');

    Allcolor = {'-bo' '-go' '-ro' '-co' '-mo' '-yo' '-ko' '-bo' '-go' '-ro' '-co' '-mo'↙
'-yo' '-ko'};
    for j = 1:J

        for i = 1:maxop
                colmat{i,j} = Allcolor {j}
        end
    end

    color = reshape(colmat,1,J*maxop);
    %linewidth2 = 10;

%Remove color if operation number is not same for all jobs
```

```
        color(4) = [];
        color(8-1) = [];
        color(15-2) = [];
        color(16-3) = [];
        color(20-4) = [];

figure
    for i=1:Nop
    StopTimes = S1OUT(i,6);
    StartTimes = StopTimes-S1OUT(i,5);
    jobno = S1OUT(i,1);
    %opID = S1OUT(i,1)*10 + S1OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*jobno,color{i},'LineWidth',linewidth2);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(jobno+0.2),num2str(S1OUT(i,4)));
    end
    title('Gantt Chart at Scenario 1 Job Vs Time');
    xlabel('Time [days]');
    ylabel('Job Number');

figure
    for i=1:Nop
    StopTimes = S2OUT(i,6);
    StartTimes = StopTimes-S2OUT(i,5);
    jobno = S2OUT(i,1);
    %opID = S2OUT(i,1)*10 + S2OUT(i,2)
    hold on
    plot([StartTimes;StopTimes],[1;1]*jobno,color{i},'LineWidth',linewidth2);
    position=StartTimes+(StopTimes-StartTimes)/2;
    text(position,(jobno+0.2),num2str(S2OUT(i,4)));
    end
    title('Gantt Chart at Scenario 2 Job Vs Time');
    xlabel('Time [days]');
    ylabel('Job Number');
```

Appendix 4: Output of Branch and Cut Algorithm

MATLAB Command Window Page 1

```
LP:                Optimal objective value is 11.000000.

Cut Generation:    Applied 5 Gomory cuts,
                   and 1 zero-half cut.
                   Lower bound is 11.000000.

Branch and Bound:

   nodes     total   num int        integer       relative
explored  time (s)  solution           fval        gap (%)
     138      0.70         1   1.500000e+01   2.500000e+01
     227      0.93         2   1.300000e+01   1.428571e+01
     830      1.63         3   1.200000e+01   7.692308e+00
   10238     10.78         4   1.100000e+01   0.000000e+00

Optimal solution found.

Intlinprog stopped because the objective value is within a gap tolerance of the optimal↙
value, options.TolGapAbs = 0 (the default
value). The intcon variables are integer within tolerance, options.TolInteger = 1e-05↙
(the default value).

LP:                Optimal objective value is 7.000000.

Cut Generation:    Applied 1 Gomory cut, 3 strong CG cuts,
                   and 1 zero-half cut.
                   Lower bound is 7.000000.

Branch and Bound:

   nodes     total   num int        integer       relative
explored  time (s)  solution           fval        gap (%)
     194      0.70         1   1.000000e+01   2.727273e+01
     399      1.03         2   9.000000e+00   2.000000e+01
    2244      2.91         3   8.000000e+00   1.111111e+01
   12244     12.10         3   8.000000e+00   1.111111e+01
   22244     21.40         3   8.000000e+00   1.111111e+01
   32244     30.68         3   8.000000e+00   1.111111e+01
   42244     40.10         3   8.000000e+00   1.111111e+01
   52244     49.38         3   8.000000e+00   1.111111e+01
   62244     58.71         3   8.000000e+00   1.111111e+01
   72244     68.11         3   8.000000e+00   1.111111e+01
   82244     77.50         3   8.000000e+00   1.111111e+01
   92244     86.98         3   8.000000e+00   1.111111e+01
  102244     96.40         3   8.000000e+00   1.111111e+01
  112244    105.74         3   8.000000e+00   1.111111e+01
  122244    115.21         3   8.000000e+00   1.111111e+01
  132244    124.50         3   8.000000e+00   1.111111e+01
  142244    133.80         3   8.000000e+00   1.111111e+01
  152244    143.24         3   8.000000e+00   1.111111e+01
  162244    152.62         3   8.000000e+00   1.111111e+01
```

172244	162.03	3	8.000000e+00	1.111111e+01
182244	171.70	3	8.000000e+00	1.111111e+01
192244	181.17	3	8.000000e+00	1.111111e+01
202244	190.25	3	8.000000e+00	1.111111e+01
212244	199.11	3	8.000000e+00	1.111111e+01
222244	208.16	3	8.000000e+00	1.111111e+01
232244	217.40	3	8.000000e+00	1.111111e+01
242244	226.67	3	8.000000e+00	1.111111e+01
252244	235.91	3	8.000000e+00	1.111111e+01
262244	245.14	3	8.000000e+00	1.111111e+01
272244	254.40	3	8.000000e+00	1.111111e+01
282244	263.37	3	8.000000e+00	1.111111e+01
292244	272.55	3	8.000000e+00	1.111111e+01
302244	281.79	3	8.000000e+00	1.111111e+01
312244	291.08	3	8.000000e+00	1.111111e+01
322244	300.41	3	8.000000e+00	1.111111e+01
332244	309.79	3	8.000000e+00	1.111111e+01
342244	319.15	3	8.000000e+00	1.111111e+01
352244	328.12	3	8.000000e+00	1.111111e+01
362244	337.42	3	8.000000e+00	1.111111e+01
372244	346.66	3	8.000000e+00	1.111111e+01
382244	355.91	3	8.000000e+00	1.111111e+01
392244	365.17	3	8.000000e+00	1.111111e+01
402244	374.45	3	8.000000e+00	1.111111e+01
412244	383.73	3	8.000000e+00	1.111111e+01
422244	393.23	3	8.000000e+00	1.111111e+01
432244	402.52	3	8.000000e+00	1.111111e+01
442244	411.82	3	8.000000e+00	1.111111e+01
452244	421.17	3	8.000000e+00	1.111111e+01
462244	430.46	3	8.000000e+00	1.111111e+01
472244	439.77	3	8.000000e+00	1.111111e+01
482244	449.07	3	8.000000e+00	1.111111e+01
492244	458.62	3	8.000000e+00	1.111111e+01
502244	467.93	3	8.000000e+00	1.111111e+01
512244	477.30	3	8.000000e+00	1.111111e+01
522244	486.54	3	8.000000e+00	1.111111e+01
532244	495.85	3	8.000000e+00	1.111111e+01
542244	505.15	3	8.000000e+00	1.111111e+01
552244	514.47	3	8.000000e+00	1.111111e+01
562244	523.72	3	8.000000e+00	1.111111e+01
572244	532.84	3	8.000000e+00	1.111111e+01
nodes explored	total time (s)	num int solution	integer fval	relative gap (%)
582244	541.69	3	8.000000e+00	1.111111e+01
592244	550.62	3	8.000000e+00	1.111111e+01
601867	559.13	3	8.000000e+00	0.000000e+00

Optimal solution found.

Intlinprog stopped because the objective value is within a gap tolerance of the optimal↙
value, options.TolGapAbs = 0 (the default

value). The intcon variables are integer within tolerance, options.TolInteger = 1e-05↙
(the default value).

```
LP:                Optimal objective value is 12.000000.

Cut Generation:    Applied 2 Gomory cuts, 1 strong CG cut,
                   and 3 zero-half cuts.
                   Lower bound is 12.000000.

Branch and Bound:

   nodes     total   num int        integer       relative
explored  time (s)  solution           fval        gap (%)
     139      1.84         1   1.400000e+01   1.333333e+01
     179      2.14         2   1.200000e+01   0.000000e+00

Optimal solution found.

Intlinprog stopped because the objective value is within a gap tolerance of the optimal↙
value, options.TolGapAbs = 0 (the default
value). The intcon variables are integer within tolerance, options.TolInteger = 1e-05↙
(the default value).

LP:                Optimal objective value is 9.526114.

Cut Generation:    Applied 4 Gomory cuts, 3 zero-half cuts,
                   and 1 flow cover cut.
                   Lower bound is 10.184659.

Branch and Bound:

   nodes     total   num int        integer       relative
explored  time (s)  solution           fval        gap (%)
      92      0.47         1   1.700000e+01   3.333333e+01
     205      0.72         2   1.100000e+01   0.000000e+00

Optimal solution found.

Intlinprog stopped because the objective value is within a gap tolerance of the optimal↙
value, options.TolGapAbs = 0 (the default
value). The intcon variables are integer within tolerance, options.TolInteger = 1e-05↙
(the default value).
```

Appendix 5: Computation time of Branch and Cut Algorithm

Profile Summary

Generated 22-Jun-2015 02:52:23 using cpu time.

Function Name	Calls	Total Time	Self Time*
intlin45	1	572.882 s	0.592 s
intlinprog	2	571.992 s	0.001 s
...hAndCut>IntlinprogBranchAndCut.run	2	571.881 s	0.030 s
slbiClient	2	571.792 s	0.010 s
optim\private\slbiMex (MEX-file)	2	570.073 s	558.344 s
slbiCallback	75	13.286 s	0.084 s
callAllOptimPlotFcns	75	13.060 s	1.835 s
optimplotmilp	75	6.721 s	0.062 s
legend	73	5.905 s	0.019 s
scr...private\legendHGUsingMATLABClasses	73	5.886 s	0.130 s
...ndHGUsingMATLABClasses>make_legend	73	5.665 s	0.095 s
Legend.doMethod	730	5.402 s	0.055 s
Legend.doUpdate	73	3.680 s	0.274 s
Legend.Legend>Legend.Legend	73	3.485 s	0.243 s
Legend.doSetup	73	2.890 s	0.162 s
Legend.doMethod>set_contextmenu	73	2.521 s	0.702 s
graph2dhelper	1022	2.026 s	0.114 s
scribe\private\createScribeUIMenuEntry	657	1.433 s	0.032 s
Legend.doMethod>createEntries	73	1.216 s	0.334 s
Legend.Legend>Legend.set.Axes	73	0.917 s	0.007 s

Appendix 6: Output of Genetic Algorithm

```
MATLAB Command Window                                          Page 1

Optimization terminated: average change in the penalty fitness value less than options.↙
TolFun
but constraints are not satisfied.
The number of generations was : 208
The number of function evaluations was : 59566
The best function value found was : 14
Optimization terminated: average change in the penalty fitness value less than options.↙
TolFun
but constraints are not satisfied.
The number of generations was : 184
The number of function evaluations was : 52726
The best function value found was : 15
```

```
MATLAB Command Window                                          Page 1

Optimization terminated: average change in the penalty fitness value less than options.↙
TolFun
but constraints are not satisfied.
The number of generations was : 246
The number of function evaluations was : 70396
The best function value found was : 30
Optimization terminated: average change in the penalty fitness value less than options.↙
TolFun
but constraints are not satisfied.
The number of generations was : 266
The number of function evaluations was : 76096
The best function value found was : 30
```

Appendix 7: Computation time of Genetic Algorithm

Profile Summary

Generated 22-Jun-2015 02:26:52 using cpu time.

Function Name	Calls	Total Time	Self Time*
makespan_ga_45	1	331.402 s	0.621 s
ga	2	330.595 s	0.004 s
globaloptim\private\gaminlp	2	251.900 s	0.001 s
globaloptim\private\gapenalty	2	251.870 s	0.003 s
globaloptim\private\galincon	2	251.853 s	0.289 s
globaloptim\private\stepGA	392	241.051 s	0.235 s
glo...im\private\gaminlpcrossoverlaplace	392	184.154 s	167.970 s
globaloptim\private\gacommon	2	78.677 s	0.012 s
glo...tim\private\preProcessLinearConstr	2	78.632 s	0.019 s
linprog	4	78.487 s	0.030 s
globaloptim\private\initialfeasible	2	68.950 s	0.022 s
optim\private\lipsol	2	68.918 s	0.144 s
optim\private\lipsol>ShermanSolve	76	68.237 s	0.065 s
optim\private\lipsol>sherman	76	68.172 s	68.172 s
optim\private\lipsol>direction	82	65.012 s	0.151 s
globaloptim\private\gaminlpmutationpower	392	32.330 s	30.546 s
...gaminlppenaltyfcn(x,problem,conScale)	394	22.433 s	0.051 s
globaloptim\private\gaminlppenaltyfcn	394	22.382 s	19.981 s
globaloptim\private\gadsplot	396	10.251 s	6.723 s
qpsub	4	9.535 s	3.749 s
glo...im\private\gaminlpsatisfyintconstr	110330	8.673 s	2.171 s