DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING – FACULTY OF ENGINEERING AND THE BUILT ENVIRONMENT – UNIVERSITY OF PRETORIA, SOUTH AFRICA

Reentrant permutation flow shop scheduling with a deteriorating schedule

Matsebe Juliet Makgoba

January 2021

A dissertation submitted in partial fulfilment of the requirements for MEng in Industrial Engineering

Abstract

The classic flow shop problem assumes that jobs make only single passes through the processing machines and that the processing times are not affected by the length of the delay before jobs are processed. These assumptions are being relaxed in recent papers that consider reentrance problems and those with schedule deterioration. In this study, these two assumptions are both relaxed, and a model of a reentrant flowshop with a deteriorating schedule is considered. A linear programming formulation of the problem is first presented. Three solution heuristics are considered under different deterioration scenarios. It was observed that both Nawaz Enscor and Ham (NEH) algorithm and Genetic Algorithm (GA) performed much better than the Campbell Dudek and Smith (CDS) algorithm. Overall, when considering both the quality of solution and computational time together, the NEH algorithm seems to have performed much better than the others as the size of problems increases. This model would find useful applications in some metallurgical and manufacturing processes where such problems are usually encountered.

Acknowledgements

My most generous gratitude goes to the person who played a significant role in guiding me to make this research possible, Dr Olufemi Adetunji. His guidance enabled me to gather all the knowledge possible, starting with the very basics of the subject. He motivated me to keep going up to the last stage. Through his guidance, I was able to enhance my research skills. I am thankful for the time he selflessly afforded to assist me on this journey.

I much appreciate the support I received from my family and their understanding of all the missed social events and gatherings. The support and encouragement received from friends and colleagues made it possible for me to focus on achieving the end goal.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction and background

This chapter gives a brief background on job scheduling in a flowshop environment and the variants to the problem which have been studied since its introduction. It also discusses the research objectives and outline of the study.

1.1. Introduction

In the current global market, companies strive to build more competitive supply chains to get their products to the market faster and cheaper than their competition. A supply chain is a network of activities that include the sourcing and conversion of raw materials into useable goods and the delivery of the goods to end-users (Simchi-Levi, 2008). A successful supply chain is the one that can deliver products efficiently and cost-effectively. The process of converting system inputs to outputs within a supply chain is often loosely considered as manufacturing. However, service industries such as information management, transportation and distribution are also an essential part of converting system inputs to outputs (Pinedo, 2016). One of the main drivers of supply chain competition is customer service, i.e. the ability of a business to continually meet its customers' requirements, which affords it a competitive advantage against their opposition. This research focuses on scheduling, which is concerned with the sequencing of jobs or operations during manufacturing.

Scheduling is an area of study within planning and control in the field of operations management. Planning and control involve the determination of quantities to be produced, at appropriate timing and with sound quality. The following planning aspects related to task performance are addressed; nature of task, time, date, location, and responsible personnel. Planning is composed of three levels; strategic, tactical, and operational (Stevenson, 2002). The strategic plan sets the direction for the business and is long-term. The tactical plan has a medium-term planning horizon and is focused on the operational part of the company. The operational plan is focused on the day-to-day operations of the shop-floor. In controlling, the system is monitored to determine if the organization is operating according to plan, and the arrangements required to get back to plan if it is no longer followed. Both planning and control are necessary for realizing the target levels of organizational objectives. *Table 1* indicates the extent of planning and control for the different levels of planning.

Table 1: Levels of planning

	Planning	**Control**	**Activities**
Strategic	High	Low	Resource planning & Rough cut capacity planning
Tactical	Medium	Medium	Detailed capacity planning
Operational	Low	High	Shop floor control (loading, scheduling, sequencing)

Note: Compiled with information from Operations Management Instructor slides, McGraw-Hill/Irwin, 2012

Scheduling is focused on determining the timing for allocating jobs to resources. The resources can be machines, facilities or people. Scheduling activities occur in every organization; for example, in the manufacturing industry, schedules are created for workers or a set of jobs. In a hospital environment, schedules can be for surgical procedures to be conducted during a specified period. In the beauty industry, places like salons use appointment systems to create schedules while the schooling environment schedules classes for teachers/instructors. Like any field of study, scheduling has its jargon used to describe the problems. The next subsection gives a brief description of the terminology (notations) used in job scheduling. This is necessary for the purpose of uniformity when describing the problems.

1.1.1. Scheduling notation

Graham et al. (1979) introduced the classic method of describing scheduling problems through the use of the three-field notation ($\alpha \mid \beta \mid \gamma$). The notation describes the machine environment (α), the problem constraints and characteristics (β), and the scheduling objective (γ). This notation is generally accepted as a standard framework for problem classification in scheduling. The description and examples of the fields of the notation are as follows.

Machine environment

The machine environment field is used to indicate the type of operation (open, job or flowshop) or the layout of the facility (one machine, n machines and machines in series or parallel). The type of operation and set-up for a facility is decided based on the types of products or process steps required for production. For example in a *job shop,* jobs visit the machines in no particular order and may not be processed on all the machines within the system. In an *open shop*, jobs are scheduled at the discretion of the planner or the scheduler and in a *flowshop,* jobs follow the same machine ordering. The flowshop configuration is a widely researched problem with work that dates back to the 1950s, with the seminal work done by Johnson in 1953.

Figure 1 gives the distinction between the different manufacturing process types and the specific industries in which each of these process types can be applied. The requirement for high customization of project and job shop processes makes manufacturing of such processes in high volumes unsuitable as it may lead to unnecessarily high inventory level and consequently high holding cost. Flowshops and continuous production processes are better suited to products that need high volume production. The choice of an appropriate process type to make a product also influences machine layouts with product layout being suitable for high volume discrete manufacturing while (flowshop) and process layout for discontinuous manufacturing (job shop) processes. Also, flowshop production machines' layout may comprise of identical parallel machines, unrelated parallel machines, and/or parallel machines with different speeds.

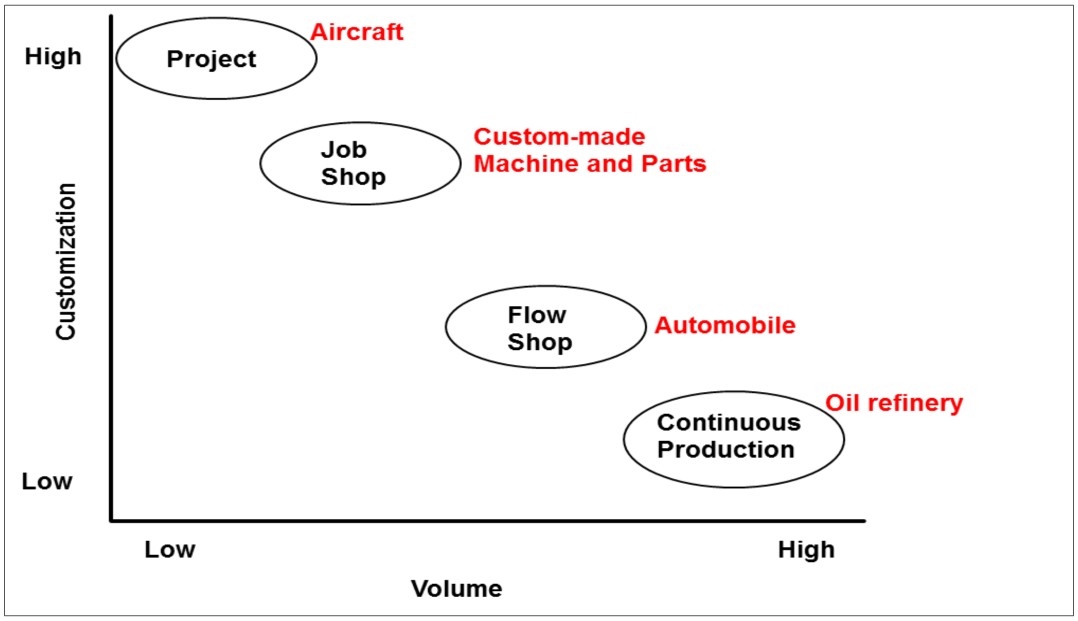


Figure 1: Process-Product alignment matrix for production volume alignment

Job Constraints and characteristics

In some scheduling problems, this field is left blank. A blank field implies that the problem will not be subjected to any specific constraint. Constraints that can be set for scheduling problems include job precedence, preemptions, due date and release times. A brief description of how these frequently used constraints are applied follows:

- Job precedence: this constraint is used to set the sequence to be followed in the schedule. Precedence between two jobs implies that a particular job may not be started unless the other one has been completed.
- Preemptions: Operations may be interrupted and then resumed at a later stage. This characteristic can be useful when quality checks do not yield satisfactory results. The job can be corrected offline while another job is processed.
- Due date: a date on which a job is expected to be completed. A job can be completed either before or after the due date. The due date is an important constraint that can be used as a guideline for determining job sequences during planning.

- Release times: the release time/date is said to be the earliest possible time at which a job can be started. Not all jobs can be ready for processing at the beginning of the planning horizon as there may be initial processing steps required on some jobs.

Scheduling objectives

Objectives in scheduling can also be referred to as performance measures; they are factors that can be used to determine the effectiveness of the process. Systems may be subjected to single or multiple objectives. Some of the commonly used objectives are flowtime, makespan, and lateness.

- Flow time: the flowtime of a job is a measure of the completion time of that job on the last machine in the process. This time is inclusive of the waiting time before processing and transportation time between the machines, and not only composed of the actual processing time of the job (Stevenson, 2002). This measure is vital in an environment where there is a need to manage work-in-process inventory levels and meeting product delivery dates. The typical performance measures being studied in this category are the *total flowtime* and the *mean flowtime*.
- Makespan: the time between the processing of the first job on the first machine and completion of the last job on the last machine. Unlike the flowtime, the measure of makespan focuses on the overall time required to process all the jobs within a planning horizon and not individual jobs. The minimisation of makespan as a performance measure translates to determining the shortest time needed to process all jobs.
- Lateness: job lateness is the difference between the actual completion time of a job and the promised delivery date. A job is considered to be late if it does not meet its due date. Lateness can either be positive (tardiness) or negative (earliness). Tardiness is an important criterion to consider in planning to ensure that no penalty costs are incurred due to products being delivered later than the promised date. Finishing a job earlier than its due date is also problematic as it increases inventory holding cost. The holding cost is described as a cost that is accrued per unit product, per time period stock is being

kept (Simchi-Levi et al., 2008). It is thus vital that products are delivered in time for the required purposes.

1.2. Problem classification, research objectives and contributions to literature

In the general scheduling notation, there aren't standard representations for re-entrance and schedule deterioration in most standard documentations available. Al-Harkan (n.d.), however, used recrc (recirculation) for reantrant jobs, while the notation for deterioration is not standard in any text and "deter" is adopted here. This study, therefore, focuses on a problem classified as $F_m/prmu, recrc, deter/C_{max}$. The problem is that of a flowshop setup, with a permutation constraint with the objective being to minimise makespan. The study involves scheduling a reentrant flowshop with deteriorating jobs. The reentrant flowshop problem has been studied for various machine environments, however, work on deteriorating jobs has significantly been on one and two machine environments. To date, there is no evidence of work published which integrates the reentrant problem with deteriorating jobs, and this serves as a motivation to consider this problem. This study aims to develop a mathematical programming model and utilize heuristic algorithms to solve problems related to scheduling of N jobs on M machines in a reentrant permutation flowshop with schedule deterioration. The problem is related to operations such as steel manufacturing, wherein certain production steps may need to be repeated to achieve the desirable product features. Scheduling of jobs in a manner that will optimise some production objectives while keeping customer satisfaction high is a challenge many production planners need to handle. Makespan minimisation in this type of production environment can lead to improved productivity and thus cost-effective running of the operations. Some known heuristics were modified to suit and solve the problem at hand as may be necessary. Solutions achieved by the various heuristics were compared to each other to determine the one which can be said to lead to finding the least makespan value within reasonable computational time.

1.3. Outline of the following chapters

A brief overview of how this document is arranged is discussed next. This document consists of five (5) chapters, and the outline of remaining chapters is provided as follows:

In Chapter 2, the literature review of the relevant scheduling areas including the important variants of flowshop scheduling problem that are pertinent such as job reentrance and deteriorating jobs is conducted. The manner in which research in this field has evolved is also reviewed. The various solution techniques available in literature for solving flowshop scheduling problems are then discussed. The focus of chapter 3 is the development of the solution approach to solve the problem. A plan of action regarding the selection of solution techniques used in solving the problem of interest is discussed. Chapter 4 presents the results and analysis of experimental computations. Results obtained from the various solution techniques are also compared to one another. In chapter 5, concluding remarks on the solution technique that resulted in the lowest makespan are made. Recommendations regarding possible areas for further study of the topic are also proposed.

Chapter 2: Literature review

This chapter focuses on basic flow-shop scheduling problem and its diverse modifications and/or variants. A discussion on general flow-shop scheduling is presented first, followed by the two variants which are the bases of this study, i.e. the flowshop problem which incorporates job reentrance and schedule deterioration.

2.1. Flowshop scheduling

In manufacturing, a facility is referred to as a flowshop when its machine set-up is in series and all jobs follow the same order during processing (Pinedo, 2016). For a flowshop with m machines, each job is processed on the first machine in the series, followed by the second machine and so on, until it reaches the m^{th} machine and all the jobs follow the same sequence. Flowshop scheduling was first studied by Johnson (1953), who based his research on two and three machine problems and provided solution methods. Systems with machines less than or equal to three can be solved successfully using Johnson's algorithm. The study focused on the generation of an optimal schedule for a two and three-stage production facility. In two machine flowshop problems for minimization of makespan, the shortest processing time (SPT) dispatching rule is often used. Solving a scheduling problem for a three machine environment proved not to be an easy task, hence the modified Johnson's rule was developed. This method compresses the three machine environment into a two machine environment to solve the problem using Johnson's algorithm, provided the problem meets the set criteria for application of the rule.

In practice, manufacturing facilities can have more than three machines on which jobs need to be processed. The issue with these types of operations is that there are no efficient exact solution procedures that are known to solve the scheduling problems (Tyagi et al., 2013). Relaxing some of the assumptions made with Johnson's rule to address the flowshop problems introduces some complexity to the system. These complex flowshop problems need to be solved using heuristic methods. Complexities may arise, for instance, when scheduling a flowshop with N jobs on M machines, a permutation flowshop arrangement, or perhaps a hybrid flowshop arrangement. Permutation and hybridization are typical complexities in flowshop scheduling, which have been researched in the past, and they are discussed next.

Permutation flowshop scheduling

In flowshop scheduling, all jobs have the same processing order; however, this doesn't imply that jobs will be processed on all the machines in the system. The constraint which gives a guarantee that jobs will not pass any machine in the shop is termed permutation. In their paper, Rios-Mercado and Bard (1999) scheduled N jobs on M machines for a permutation flowshop. They developed a branch and bound enumeration scheme for minimizing the makespan. The model incorporated lower and upper bounding, a dominance elimination logic and a partial enumeration strategy. Upper and lower bounding serves to limit the search space and thus ensuring that a solution can be achieved within a reasonable computation time.

Genetic algorithm solution (GAs) has also proven to be aloe to successfully find near-optimal solutions for complex scheduling problems, where methods such as the branch and bound failed due to the time complexity of exact solutions (Iyer and Saxena, 2004). In this paper, an improved GA heuristic was used in finding solution for a permutation flowshop in an assembly line whose objective is to minimize the makespan. The improved GA solution was implemented by ensuring that it matches the structural information of the system being studied. It was shown in the study that the modified GA performs better than the standard GA.

Hybrid flowshop scheduling

A hybrid flowshop, also known as a multi-processor, is a type of flowshop arrangement with two or more production stages in series. Each production stage is composed of identical machines, working in parallel. This characteristic offers some flexibility to the system while increasing productivity. Choi and Lee (2007) studied a two-stage hybrid flowshop with the objective of minimizing the number of tardy jobs. They utilized the branch and bound method to develop their solution. The heuristic they developed resulted in optimal solutions for moderately sized problems, i.e., with four machines at each stage and fifteen jobs.

The work in flowshop scheduling has caught the attention of a lot of researchers, and some variants have been explored, owing to the maiden two- and three machine problems. One such variant is the reentrant characteristic and is discussed next.

2.2. Reentrant flowshop problem

The traditional classification of manufacturing systems differentiates the systems by referring to them as either a job shop or flowshop. The classical flowshop is characterized by jobs that are processed on each machine within the system only once. Chen et al. (2008), however, stated that in reality, this assumption is sometimes violated. The allowance of jobs to return to one or more machines for further processing is termed reentrance.

The reentrant property is observed in processes such as semiconductor manufacturing (Chen et al., 2008) wherein components need to be processed more than once before the final product is achieved and cold drawing operations in steel tube manufacturing. Reentrance can also be used in tool machining shops (Graves et al., 1983), where a particular tool might require a polishing stage or heat treatment in between the machining stages to achieve the final tooling finish. This system is referred to as a reentrant flowshop, and *Figure 2* shows a simple illustration of the reentrant property for a permutation flowshop.

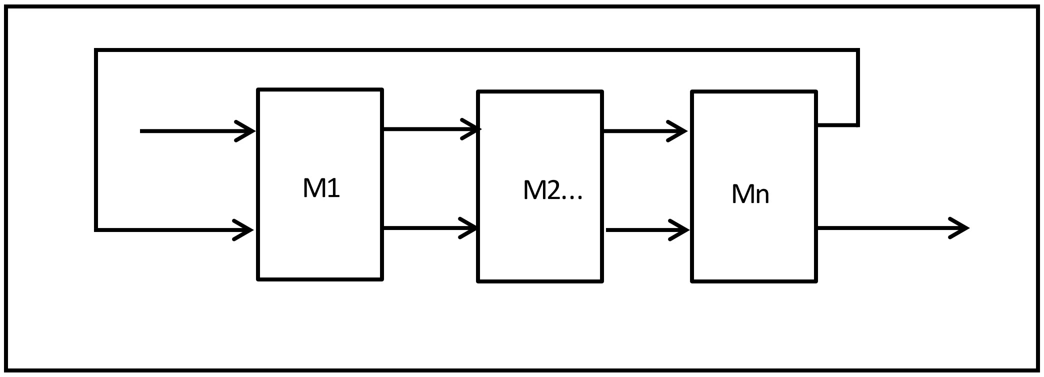


Figure 2: Schematic diagram of a reentrant flowshop

A job is processed on all machines in the shop and then returns into the system for the next level's operation. The illustration is somewhat overly simplified as jobs will not necessarily return into the system just once; jobs may return for the number of passes or levels required to yield the end product or service based on the system requirements. To describe the process, one may compare the reentrant shop with the traditional flowshop set-up. In a conventional flowshop, scheduling a set of jobs $N = \{J_1, J_2, \ldots, J_n\}$ on machines $M = \{M_1, M_2, \ldots, M_m\}$ require that all jobs in set N are processed in the sequence M_1 then M_2 up to M_m only once. In a reentrant shop, jobs in set N are processed in the sequence M_1 then M_2 up to M_m and return to the same set of machines for a second, third or n^{th} pass operation while following the same sequence as in the first pass.

For decades, the reentrant flowshop problem has attracted the attention of many researchers, and this can be attributed to the drive to reduce operating costs and increase profits by manufacturing facilities. Instead of procuring two or more machines that can perform the same operation, a single machine can be used. The number of machines within a shop is directly linked to the number of operators required to operate them, and as such, the reentrant shop can also result in lower labour costs (a portion of fixed costs).

The earliest work identified on reentrant flowshop scheduling was that of Graves et al. (1983). These were one of the first researchers to realize that not all of the existing models would be beneficial if applied in different types of systems. The manufacturing process that was used for their study is that of an integrated circuit fabrication facility. It was initially suggested that the system could be operated as a job shop with the use of sequencing rules to determine an optimal sequence of jobs at each machine. This implied that a simple Gantt chart for sequencing of jobs in this system could be used; however, for a large number of jobs, it would be nearly impossible to manage. A heuristic algorithm was, thus, developed to minimize the throughput time using a cyclic scheduling method at specified production rates. The heuristic developed was referred to as a cyclic Gantt chart. In the cyclic schedule, the chart is divided into manageable cycle times to enable the scheduler to know the number of cycles required to complete a single job. One of the unique features about the model is that the schedule that is generated could be adapted when conditions in the system such as shutdowns, machine breakdowns, and operator unavailability arose. The downside, however, was that the model was developed as a computer program from which the results needed to be transferred onto a shop schedule. Any errors that take place during the transfer of the schedule from the program to the workshop can change the entire process. Since the program was developed, there have been various technological advancements, which in the current times are an added advantage to reducing human error.

Despite slow progress in the research of reentrant flowshop problems, Pan and Chen (2003) furthered work in this field with a focus on the minimization of makespan in a reentrant permutation flowshop. They proved that the reentrant flowshop problem is NP-hard, even for a two-machine shop. The duo developed several heuristics in their search for a solution approach for the problem. Chen (2006) extended the work done on the reentrant permutation flowshop by developing a branch and bound algorithm. For the algorithm to reach an optimal solution quickly, a branching rule, upper and lower bounding rule, and a fathoming rule were utilised. Chen et al. (2007) developed an integer programming model and used heuristics to find an initial solution. Tabu search was then applied to improve the initial solution in scheduling jobs in a reentrant N machine environment to minimize makespan. The combination of the pure Tabu search with neighborhood search heuristics enhances the efficiency of the Tabu search by ensuring that the local minima are also explored.

Rios-Mercado and Bard (1999) had earlier presented research conducted on scheduling by using a branch and bound algorithm for permutation flowshops with sequence-dependent set-up times. Their algorithm included the use of lower and upper bounding procedures, dominance rules and partial enumeration to reach a solution. As more and more researchers explored the ability of the branch and bound approach to solve the various reentrant problems, the possibility of incorporating the traditional scheduling heuristics for finding upper bounds were also taken into consideration (Choi and Kim, 2007).

In the paper of He et al. (2011), they opted to use a different approach to model the reentrant problem. A model based on partial differential equations was presented. The model has the ability to handle a large number of jobs on many processing stages. In the first part of their study, they presented a basic continuum model for material flows. They proved using computational examples that it results in an inaccurate solution for the reentrant problem. The inaccuracies were then corrected by modifying the continuum model and incorporating the reentrant degree/ factor of the products. The impact of varying degrees of the reentrant factor on total system behaviour was also studied through computational experimentation.

More recent work includes scheduling a reentrant no-wait shop and systems with multiple objectives. One such problem was studied by Rifai et al. (2016) wherein three objective functions, namely maximum completion time, total production cost, and average tardiness, were considered. A multi-objective adaptive large neighborhood search (MOALNS) algorithm was developed to find near-optimal solution for the problem. TasoujiHassanpour et al. (2015) proved that the no-wait reentrant problem is NP-hard and utilized heuristic algorithms to solve the problem. Genetic algorithm, bottleneck based heuristic and simulated annealing heuristic were used to solve the problem. The simulated annealing algorithm out-performed the genetic algorithm and the bottleneck based algorithm in terms of finding the best solution within reasonable computation time.

The majority of flowshop problems in the study only involve a single machine at each production stage. There is, however, another type of flowshop referred to as a hybrid flowshop. This type of flowshop involves the use of more than one machine at each production stage (Moursli, 1999). The hybrid flowshop is common in semiconductor production facilities (Kim and Lee, 2008). Much like the use of the reentrant property, using multiple machines per production stage is aimed at improving the productivity and efficiency of the facility. The reentrant property has also been studied in this type of flowshop environment with work including that of Choi et al. (2009), Kim and Lee (2008) and Zhang and Chen (2017). Heuristic methods have been used to solve various scheduling problems. Choi et al. (2009) applied the branch and bound algorithm. They compared it to modified heuristics (Johnson's, CDS and NEH) to schedule a two-stage reentrant hybrid flowshop to minimize makespan under the maximum allowable due dates. Kim and Lee (2008) adopted the same methodology and applied the modified CDS and NEH heuristics for scheduling jobs in a reentrant hybrid flowshop with unrelated parallel machines.

In cases where more than two passes or levels of processing are required, the sequence of operation is repeated. The repeated cycles that the jobs need to go through may lead to deterioration as the jobs queue to be processed. The deterioration factor affects the total processing time of jobs and can lead to high operating costs. The next section focuses on the deterioration factor and how it affects permutation flowshop scheduling.

2.3. Deterioration

The deterioration of jobs can be described in various ways. It can refer to deterioration while waiting for service due to the unavailability of machines or deterioration caused by fatigue or tiredness of machine operators. The deterioration factor of jobs can also be influenced by scheduled maintenance or cleaning of machines (Bank et al., 2012). In some instances, jobs require a preparation step before processing; the cold-drawn tube manufacturing is a typical example. The input to the process is prepared by applying lubricants and warming it up. The material loses temperature while it waits to be processed and will need to be re-heated to ensure it is at the correct temperature. Ingot and bloom rolling can also be affected by the deterioration of processing times. In these steel rolling operations, the input material is reheated to a predetermined temperature before rolling. Each steel grade has a critical temperature below which rolling becomes difficult and may cause machine damage. Below this temperature, the input ingots/blooms are said to have deteriorated and require a reheating cycle. In all cases of deteriorating jobs, the result is processing times that increase with an increase in waiting time before processing. A deteriorating job is thus referred to as a job that will take more time to be processed later than when it is processed first in the schedule.

The job deterioration problem is relatively new, currently spanning only three decades of research. Despite that, the job deterioration problem has had variants of the original problem studied since its inception. The first decade did not see much development. Studies were conducted with a focus on single machine problems with the objective being minimization of makespan (Browne and Yechiali, 1990, and Kubiak and van de Velde, 1998) and minimisation of flowtime (Mosheiov, 1991). Browne and Yechiali (1990) pioneered the study of deteriorating jobs, with a focus on a single machine problem with simple linear deterioration and an objective of makespan minimization. The duo proved that the processing time of a job increases linearly in relation to the waiting time, and they used scheduling policies to develop a solution.

Meanwhile, Mosheiov (1991) studied the minimization of flowtime for N jobs on a single machine and proved that the optimal sequence for the problem is V-shaped. The V-shape property essentially means that the first set of jobs to be scheduled is arranged in decreasing order of the deterioration rate. In contrast, the remaining set of jobs is scheduled in increasing order of their deterioration rate.

A study proving the NP-hardness of a two- and three-machine scheduling problem for deteriorating jobs was conducted by Kononov et al. (2001). The types of deteriorating jobs studied were the simple linear and proportional deterioration with the objective of minimizing the makespan. Hindi and Mhlanga (2001) also studied the simple linear deterioration but coupled it with the study of jobs with basic processing times. The difference between the two types of deteriorating jobs in this study is that in simple linear deterioration, it is assumed that jobs are only available at a positive time t_0 and that for jobs with basic processing time, the scheduling horizon starts at $t = 0$ for all jobs. The total processing time of jobs with basic processing time is made up of two components; the time component induced by deterioration and the actual processing time component. Thus, the longer a job waits in the queue before processing, the longer it will take to be completed. Heuristics were used to schedule jobs for the parallel machine environment to minimise makespan. The choice of the solution technique was based on the fact that scheduling problems with parallel machines are NP-hard, and thus introducing deterioration increases the complexity of the problem.

Wang et al. (2006) presented a paper on the study of minimization of total completion time in a two-machine flowshop with deteriorating jobs. An assumption that the deterioration has a linear function in relation to the processing time was made. A branch and bound algorithm was used together with several dominance properties and two lower bounds to speed up the elimination process during enumeration. Bank et al. (2012) studied a similar problem with the objective, however, being to minimize total tardiness. The same approach was used to develop the solution, where a branch and bound with dominance properties and upper and lower bounds were used to reach an efficient schedule. Job tardiness is extremely important in cases where companies are more concerned with meeting their customer's expectations in terms of due dates, and supply chains are striving to out-perform each other with regards to customer satisfaction.

A lot of focus for the scheduling of deteriorating jobs has been on manufacturing facilities with machines that are less than or equal to three. Single machine scheduling of deteriorating jobs has been studied by Browne and Yechiali (1990), Mosheiov (1991), Wang and Xia (2005), Ji and Cheng (2010) and Wang et al. (2011). Shiau et al. (2007) and Wang et al. (2006) studied deteriorating jobs for two machine manufacturing systems. Jafari et al. (2017) studied three machine problems. Wang et al. (2019), however, studied an M machine (m > 3) problem for scheduling deteriorating jobs. They utilized a metaheuristic called multi-verse optimizer (MVO) to find a solution to the problem. They were one of the few identified in literature to have studied the problem for $m > 3$ machines.

The review of the different studies conducted in the scheduling of manufacturing systems is indicative of the various solution methods available in the literature. The next subsection focuses on an in-depth analysis of some of these solution methods.

2.4. Solution techniques for flowshop scheduling

There is a vast range of solution methods available in literature that can be applied to flowshop scheduling. Ruiz and Maroto (2005) compared the performance of twenty-five (25) solution methods to evaluate their performance for scheduling a permutation flowshop. Tyagi et al. (2013) in their own survey of flowshop scheduling showed how solution methods have evolved over six decades. They grouped the solution methods discovered per decade and also arranged them according to the categories they belong to, i.e., exact methods, heuristics and meta-heuristics. This subsection focuses on the popular solution techniques available in literature for scheduling flowshop problems, reviewing how the methods function and how they have been used in literature. The review of these methods is necessary for the selection of suitable methods for solving the problem of interest. The algorithms and heuristics to be discussed are Johnson's algorithm, CDS, NEH, genetic algorithm (GA), general sequencing rules, branch and bound algorithm and Tabu search. The main review focus would be on the techniques adopted in solving the problem of interest.

2.4.1. Johnson's algorithm

Johnson (1953) pioneered the study of flowshop scheduling with the maiden research focused on the two and three machine problems. Johnson's rule can only be applied to two and a specific type of three machine problems. There are some conditions that a three machine set-up needs to satisfy for Johnson's rule to be used. The requirements for scheduling a three machine flowshop will be discussed later in this subsection. Assumptions made to apply Johnson's algorithm in scheduling two and three machine problems are that:

- All jobs are available from time zero
- All job processing times are known
- There are no machine breakdowns
- There is unlimited space for jobs waiting to be processed
- Pre-emption is not allowed
- Machine set-up times are included in the processing time
- Each machine can only handle one job at a time
- All of the job processing times are known
- Only one machine of each type is used

The original Johnson's algorithm

Johnson's algorithm was developed during an era when technological advancements were slow. The algorithm was designed in such a way as to enable it to be used manually, and still, be able to achieve an optimum schedule. Steps followed for execution of Johnson's rule are detailed in Algorithm 1.

```
Algorithm 1: Johnson's two-machine algorithm
Step 1: For a set of jobs to be scheduled on two machines, let $p_{mj}$ be the processing time of job $j$ on machine $m$
Step 2: Put in set I all jobs for which contain $p_{1j} < p_{2j}$
Step 3: Put in set II all jobs for which contain $p_{1j} > p_{2j}$
Step 4: Jobs with $p_{1j} = p_{2j}$ can be placed in either set
Step 5: Arrange jobs in set I in increasing order, and those in set II in decreasing order. If there are any ties, break them arbitrarily.
Step 6: Create an optimal sequence by combining jobs in set I and set II
Step 7: Compute the objective function
```

The limitation of Johnson's algorithm was that it could only be applied to two machine problems, and thus needed an extension. The extended version of the algorithm was aimed at scheduling a three machine problem by manual computation.

The modified Johnson's algorithm

The modified Johnson's algorithm applies to a three machine problem, which satisfies either or both of the following conditions:

- The maximum processing time of job j on the second machine is less than or equal to the minimum processing time of job j on the first machine, i.e. $\max(p_{2j}) \leq \min(p_{1j})$
- The maximum processing time of job j on the second machine is less than or equal to the minimum processing time of job j on the third machine, i.e. $\max(p_{2j}) \leq \min(p_{3j})$

Algorithm 2: Johnson's three-machine algorithm

```
Step 1: Check for the minimum value of processing time on machines $M_1$ and $M_3$ and check for the maximum value of processing time on machine $M_2$
Step 2: Subject the problem to the three machine conditions
Step 3: Generate $M'_1$ and $M'_2$; where $M'_1 = M_1 + M_2$ and $M'_2 = M_2 + M_3$
Step 4: Apply Johnson's two-machine algorithm using $M'_1$ and $M'_2$
Step 5: Use the optimal sequence in step 4 to evaluate the objective function in the original problem
```

Johnson's algorithm in application

Johnson's algorithm is undoubtedly the cornerstone of flowshop scheduling and has, in many instances, been modified to match the problems being addressed. Hsu et al. (2006) opted to do just that when they modified Johnson's rule and combined it with despatching rules to solve the problem of minimizing the makespan in a two-stage flowshop with a function constraint on alternative machines. They proved that combining their version of the modified Johnson's rule with the First-Fit despatching rule performed better than the pure form of the heuristic. Regardless of the original algorithm only being applied to two-machine and some three-machine problems, it can be said with confidence that with any necessary modifications, Johnson's rule can be used on several variants of the classic flowshop problem.

Choudhari and Khanna (2017) presented a scheduling problem with four machines and N jobs and an objective of minimizing makespan. Due to the restrictions on the number of machines, the duo could not use the pure form of Johnson's algorithm to solve the problem at hand. They developed an algorithm that converted the four-machine problem into a three-machine, and then a two-machine problem. The job sequence obtained by applying Johnson's rule was then substituted back into the original problem to compute for the makespan.

2.4.2. CDS Algorithm

With further advancements in the scheduling of jobs on several machines, the trio of Campbell, Dudek, and Smith made their contribution to the study in the year 1970 (Tyagi et al., 2013). This gave rise to their now well-known algorithm for scheduling of k jobs on n machines, the CDS algorithm. The algorithm converts a n machine problem (i.e., more than two) into several surrogate two machines (Campbell et al., 1970). In so doing, the problem gets reduced to a two-machine problem as with the modified Johnson's algorithm. Algorithm 3 is a short description of how the CDS algorithm is executed.

Algorithm 3: CDS algorithm

Step 1: Formulate ($\boldsymbol{n}$-1) 2-machine surrogate problems from the original problem. The data is generated for $\boldsymbol{M'_1}$ and $\boldsymbol{M'_2}$, the surrogate machines in the following manner:

For $\boldsymbol{M = 1, \dots, n-1}$ and $\boldsymbol{j = 1, \dots, k}$

$\boldsymbol{M'_1 = \sum_{m=1}^{n-1} p_{mj}}$ and $\boldsymbol{M'_2 = \sum_{m=n-M+1}^{2} p_{mj}}$

Step 2: Use Johnson's algorithm to schedule the surrogate problems

- For the set of jobs to be scheduled on two machines, let $\boldsymbol{p_{mj}}$ be the processing time of job $\boldsymbol{j}$ on machine $\boldsymbol{m}$
- Generate set I for jobs that contain $\boldsymbol{p_{1j} < p_{2j}}$
- Generate set II for jobs that contain $\boldsymbol{p_{1j} > p_{2j}}$
- Jobs with $\boldsymbol{p_{1j} = p_{2j}}$ can be placed in either set
- Order jobs in set I in increasing order, and those in set II in decreasing order. Break any ties arbitrarily.
- Create an optimal sequence by combining jobs in set I and set II
- Evaluate the objective function

Step 3: Select the schedule which satisfies the objective function

CDS in application

The original CDS algorithm was developed to allow for scheduling of large problems. The current trend of research in this area of study is the modification of the original algorithm. Choi et al. (2009), for instance, modified the algorithm to suit scheduling a two-stage hybrid reentrant flowshop with an objective of minimizing makespan. The performance of the modified algorithm was compared to other algorithms that had also been modified, such as Johnson's and NEH. The CDS algorithm is relatively adaptable to various types of scheduling problems and can result in near-optimal solutions.

2.4.3. NEH algorithm

Nawaz, Enscore and Ham developed the NEH algorithm in 1983 (Tyagi et al., 2013). This solution method has a unique feature. The algorithm uses what is called insertion to establish the optimal schedule for the problem (Baskar 2016), as is described in Algorithm 4.

Algorithm 4: NEH algorithm

```
Step 1: Calculate the total processing time of each job and arrange the jobs in decreasing order
Step 2: Schedule the first two jobs from the list, i.e. the ones with the highest processing time. Form two partial sequences by interchanging the place of the jobs.
Step 3: Evaluate the objective function for the partial sequences
Step 4: Retain the partial sequence that minimizes the objective function; term it "the incumbent sequence."
Step 5: Insert the next job from the list in all possible places of the incumbent sequence. Evaluate the objective function for the partial sequences.
Step 6: Retain the schedule that results in the minimum, and discard the rest.
Step 7: End if all jobs have been scheduled, else go to step 5
```

NEH in apaplication

The power of insertion of the NEH algorithm was tested by Baskar (2016), wherein seven variants of the algorithm were created by varying the method of selecting the initial sequence and then comparing them with the original version. This study proved that even the simplest versions of the NEH algorithm are capable of arriving at an optimal solution. In a survey of permutation flowshop scheduling, it was demonstrated that the NEH out-performs other algorithms and heuristics if tested against Taillard's instance benchmark (Ruiz and Maroto 2005).

2.4.4. Genetic Algorithm (GA)

The genetic algorithm (GA) was developed by John Holland, together with his colleagues and students. The main aim of their research development was to relate artificial systems with natural systems (Goldberg, 1989). This search method can easily be adapted to any kind of optimization problem. GA is a search algorithm that was developed based on Darwin's biological theory of natural selection. This algorithm operates on an initial large population, selects schedules that out-perform the others, and carries them over to the next generation to re-create new schedules. New generations of the population are the evolved version of the previous generations.

The basic structure of the genetic algorithm consists of population initialization, fitness evaluation, GA operators (selection, crossover, and mutation), survivor selection, and termination. The basic GA structure is represented in *Figure 3*. The different components of GA are discussed next.

Objective and cost functions: In developing a GA, one needs to identify a variable within the system, which is to be optimized.

Population initialization: The initial population consists of n randomly generated job sequences, which are often referred to as chromosomes.

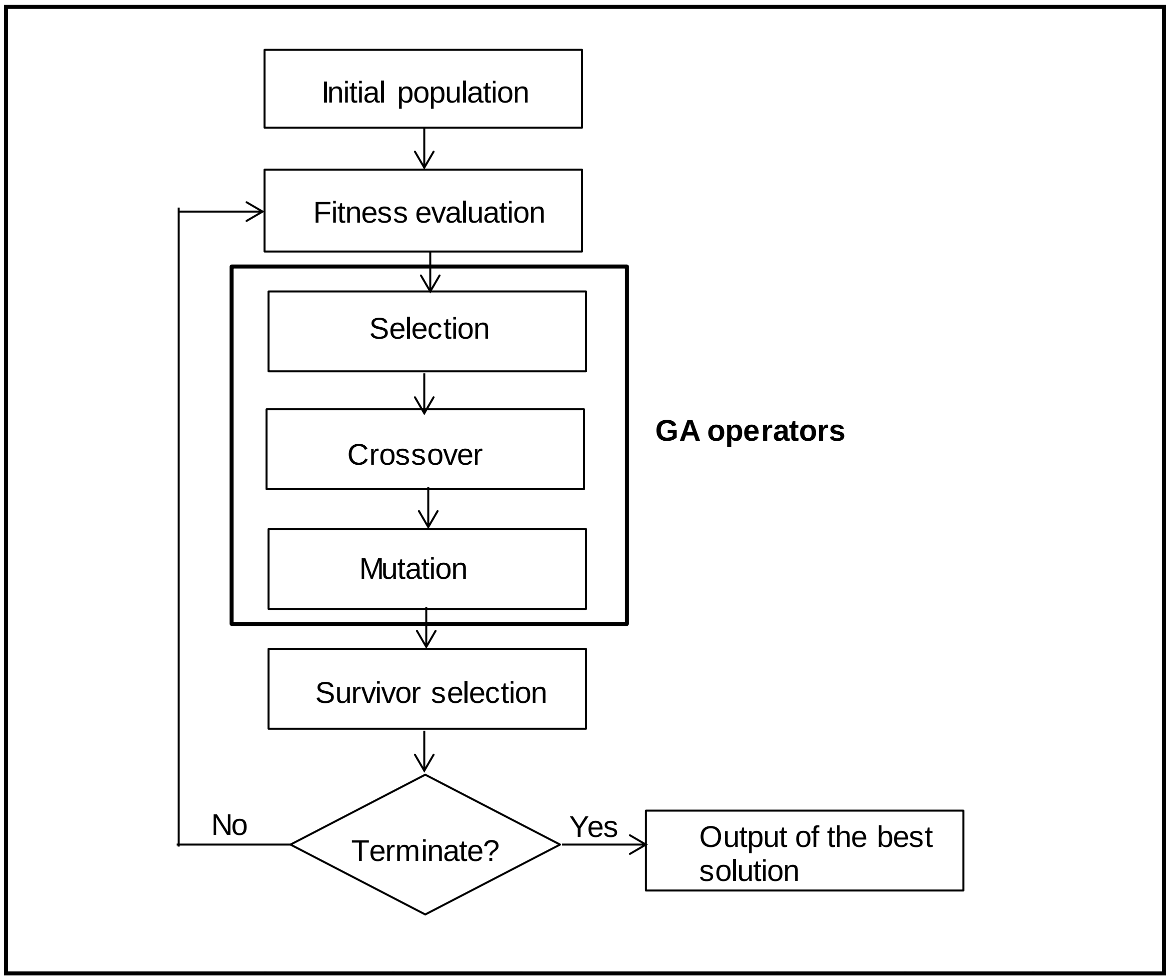


Figure 3: Basic GA structure

Fitness evaluation: The evaluation step is created to apply the theory of natural selection. A fitness function is used to assess the fittest sequences of the population.

Selection: The fittest sequences that will form part of the next generation are selected and are subjected to the GA operators to get diversification in the system.

Crossover: This is the operation responsible for the generation of a new population (child) from the parent population. Murata et al. (1996) reviewed several crossover operators, including one-point crossover, two-point crossover, and position-based crossover. Multi-point crossover operations are disruptive and thus assist in converging to the fittest chromosomes quicker.

Mutation: Mutation is an operation that is applied to a schedule to alter the job sequence deliberately. This is done in the hope of finding a better solution in the neighbourhood. Adjacent two-job change, arbitrary two-job change, and shift change are some of the mutation operators that are widely used (Murata et al., 1996).

Survivor selection: The mutated schedules are evaluated against the objective function using the local search heuristics. The job schedule that returns an optimal or a near-optimal solution gets carried over to the next generation. One can perform this step using one heuristic, which has been proven to out-perform its counterparts (Pan and Chen, 2003) or apply all of the identified heuristics and compare their performance in their specific case.

Termination: The decision to terminate the algorithm can be based on whether the returned solution satisfies the objective function or not or after a certain number of iterations. The decision to end the algorithm after several iterations is usually based on some form of experience from observing how the GA converges when performing computational experiments.

The GA tends to yield superior results compared to its counterpart search methods. A comparison of the Genetic Algorithm to the traditional search methods indicates the following (Goldberg, 1989):

- GA searches from a population of points and not from a single point.
- GA doesn't need any derivative information or other auxiliary information; the set objective and fitness functions influence the search space.
- GA uses probabilistic transition rules rather than deterministic.

Some of the known advantages of using GA over other algorithms were discussed by Pinedo (2016) and are as follows:

- GA can easily be coded
- GA can be applied to problems whose structural properties are not known

GA in application

GA has been widely used in scheduling due to its ability to be adapted to various types of problems. Modifications to the basic steps have been presented by different researchers for objective functions such as makespan minimization in a reentrant flowshop (Chen et al., 2008) and a reentrant no-wait flowshop (TasoujiHassanpour et al., 2015).

2.4.5. Other solution techniques

The previous subsection focused on solution methods which are foundational to the area of scheduling and are being considered for solving a reentrant flowshop problem with deteriorating jobs. This subsection gives a brief background of some other popular solution methods that have been successfully applied to scheduling flowshops. The solution methods to be discussed are sequencing rules, branch and bound algorithm and the Tabu search. These methods have either been used as stand-alone solution methods, or in combination with others. Although reviewed, these methods will not be used in solving the problem of interest. Some shortfalls of these methods, which led to them not being considered for this study, are highlighted in this subsection.

Sequencing rules

Some flowshop problems are solved using sequencing rules. Sequencing rules are used due to their simplicity of implementation. The sequencing rules can also be applied as a method for generating an initial solution, which gets improved on by the use of global search methods such as GA. Some disadvantages linked to the sequencing rules are that the quality of the solution cannot be guaranteed and that it is not easy to determine how far the solution is from optimal (Tyagi et al., 2013). The sequencing rules that are often used include:

- First-in first-out (FIFO) – jobs are sequenced in the order that they arrived at the machine.

- Shortest processing time (SPT) – job sequences are determined based on processing times, and as the name states the job with the shortest processing time is scheduled first.

- Longest processing time (LPT) – job sequences are determined based on processing times, and jobs with the longest processing times are scheduled first.

- Earliest due date (EDD) – jobs are sequenced in the order that they are due for delivery.

Branch and bound

The branch and bound algorithm is widely used in classic flowshop scheduling as it searches the entire space for the best solution. The size of the search space results in $n!$ possible schedules. The $n!$ combinations can be generated on an enumeration tree, however, this can be inefficient (Chen, 2006) and may lead to non-optimal solutions due to the necessity to truncate the search. The branch and bound algorithm is often used for simple three machine environments. Yet, with the rising interest in applying the branch and bound algorithm in reentrant shops, researchers such as Rios-Mercado and Bard (1999) and Chen (2006) have explored the possibility of applying the algorithm to a n job, m machine environment. Choi et al. (2009) also used the branch and bound algorithm for a two-stage reentrant hybrid flowshop for minimizing makespan under the maximum allowable due dates.

Tabu search

As described by Pinedo (2016), the Tabu search algorithm is an improvement type algorithm, meaning that it starts with a schedule (which can be generated arbitrarily) and work on manipulating and improving it until the best solution is achieved. This search algorithm can simply be combined with other methods, especially for creating the initial schedule. Eren and Güner (2007) used the EDD sequencing rule to generate the initial sequence. Chen et al. (2007) opted for a hybridized method, where they used a combination of a sequencing rule with a heuristic to generate the initial solution and then applied the Tabu search to improve the solution. Hybridization of the Tabu search ensures that the best solution doesn’t get trapped in the local minima. The Tabu search appears to perform best with the hybrid approach for initial schedule generation (Ruiz and Maroto, 2005). A successful selection of an initialization method requires prior knowledge of how the problem behaves.

At this point, it may be necessary to summarise what has been done in this chapter and consider what follows. The general flowshop scheduling problem together with its variants was reviewed. Further to that, reentrance and the deterioration factor were discussed to evaluate research progress made till date in this field. The review of the various types of environments studied uncovered the vast number of solution methods available in literature. The functionality and application of some of the solution methods was, thus, discussed. The next chapter focuses on developing a model for solving the problem of product reentrance and deteriorating jobs and the selection of solution methods to be used.

Chapter 3: Model development and solution approach

This chapter focuses on model design based on advancements already made in this area of study by other researchers. Two models, one based on reentrance, and the other based on deteriorating jobs, will be presented. For the scheduling problem being considered, the assumptions made are stated, and the linear programming model is formulated. The last section of the chapter presents the solution algorithms for solving the problem of interest.

3.1. Model eyeballing

In this section, two models that are foundational to the problem of deteriorating jobs in a reentrant flowshop are presented. The notation to be used throughout the section is introduced first, followed by the two models.

Notation:

Symbol	**Description**
$\boldsymbol{J}$	job index set; $J: \{j = 1,2, \dots, k\}$, where k is the number of jobs
$\boldsymbol{M}$	machine index set; $M: \{m = 1,2, \dots, n\}$, where n is the number of machines
$\boldsymbol{L}$	number of levels of job j; $L: \{l = 1,2, \dots, b\}$, where b is the number of levels
$\boldsymbol{i}$	position of job j in the sequence
$\boldsymbol{x_{ij}}$	1, if job j is scheduled in the ith position at each level; 0 otherwise
$\boldsymbol{s_{mli}}$	the starting time of a job scheduled in the ith position of level l on machine m
$\boldsymbol{a_{mli}}$	the deterioration rate of a job scheduled in the ith position of level l on machine m
$\boldsymbol{a_j}$	the deterioration rate of a job j
$\boldsymbol{p'_{jlm}}$	the normal processing time of the operation of job j on machine m at level l
$\boldsymbol{p_{jlm}}$	the actual processing time of the operation of job j on machine m at level l, where $p_{jlm} = p'_{jlm} + a_{mli} \cdot s_{mli}$
$\boldsymbol{C_{max}}$	completion time of the last job in the sequence
$\boldsymbol{C_j}$	completion time of job j

3.1.1. Reentrant flowshop model

Pan and Chen (2003) were the first to formulate models for the reentrant permutation flowshop (RPFS); which they had derived from models previously presented by other researchers focused on modelling permutation flowshops. Computations performed proved that the formulation derived from Wilson's model performs well with the modified heuristics. The modified Wilson's model is presented as follows:

Minimize C_{max} (1)

Subject to $\sum_{j=1}^{k} x_{ij} = 1 \quad i = 1,2,\dots,k$ (2)

$\sum_{i=1}^{k} x_{ij} = 1 \quad j = 1,2,\dots,k$ (3)

$s_{111} = 0$ (4)

$s_{1,1,i+1} = s_{11i} + \sum_{j=1}^{k} x_{ij} p'_{j11} \quad i = 1,2,\dots,q-1$ (5)

$s_{1,l,i+1} \geq s_{1li} + \sum_{j=1}^{k} x_{ij}\, p'_{jl1} \quad l = 2,3,\dots,b;\, i = 1,2,\dots,q-1$ (6)

$s_{1,l+1,1} \geq s_{1lk} + \sum_{j=1}^{k} x_{ik}\, p'_{jl1} \quad l = 1,2,\dots,b-1$ (7)

$s_{1,l+1,i} \geq s_{nli} + \sum_{j=1}^{k} x_{ij}\, p'_{jln} \quad l = 1,2,\dots,b-1;\, i = 1,2,\dots,q-1$ (8)

$s_{m,l,i+1} \geq s_{mli} + \sum_{j=1}^{k} x_{ij}\, p'_{jlm} \quad m = 2,3,\dots n;\, l = 1,2,\dots,b;\, i = 1,2,\dots,q-1$ (9)

$s_{m,l+1,1} = s_{mlq} + \sum_{j=1}^{k} x_{qj}\, p'_{j1n} \quad m = 1,2,\dots,n;\, l = 1,2,\dots,b-1$ (10)

$s_{m+1,1,1} \geq s_{m11} + \sum_{j=1}^{k} x_{1j}\, p'_{j1m} \quad m = 1,2,\dots,n-1$ (11)

$s_{m+1,l,i} \geq s_{mli} + \sum_{j=1}^{k} x_{ij}\, p'_{jlm} \quad m = 1,2,\dots n-1;\, l = 1,2,\dots,b;\, i = 1,2,\dots,q;\, (l,i) \notin \{(1,1)\}$ (12)

$s_{m+1,l+1,1} \geq s_{mlq} + \sum_{j=1}^{k} x_{qj}\, p'_{jlm} \quad m = 1,2,\dots,n-1;\, l = 1,2,\dots,b$ (13)

$C_{max} = s_{nbq} + \sum_{j=1}^{k} x_{qj}\, p'_{jbn}$ (14)

$C_{max} \geq 0, s_{mli} \geq 0 \quad m = 1,2,\dots,n;\, l = 1,2,\dots,b;\, i = 1,2,\dots,q;$

$x_{ij} = 0 \text{ } or \text{ } 1 \quad i = 1,2,\dots,k; j = 1,2,\dots,k$ (15)

Equation (1) is the objective function. Constraints (2) and (3) describe the decision variables. They ensure that every job is in only one position and that each position has only one job assigned to it. Constraint (4) describes the starting time of the first operation scheduled at the first level of the first machine. Constraints (5) and (11) are expressions for the starting time of any job, on any machine at the first operation level, while (6) – (10), (12) and (13) enforce the precedence relations. Constraint (14) defines C_{max} to be the finish time of the last job processed on M_n (the last machine) at the last level. Constraint (15) enforces the non-negativity and binary restrictions.

3.1.2. Deterioration model

Two types of deterioration models were studied by Hindi and Mhlanga (2001); simple linear deterioration and jobs with basic processing time. Simple linear deterioration applies to problems where all jobs are subject to a deterioration rate and the processing time of a specific job j is calculated by $p_j = a_j s_j$. The basic processing time model is aimed at problems with jobs that are available from time $t = 0$. All jobs have a job-specific processing time. However, the total processing time is influenced by the delay a job incurs while it waits to be processed and, thus, the total processing time grows linearly with the delay. The focus of this study is on the model for jobs with basic processing time. Only the deterioration part of the model is being considered due to its similarity to other flowshop problems. The model for a single machine is presented as follows:

$p_j = p'_j + a_j s_j$ (16)

$C_j = \sum_{j=1}^{k} p_j$ (17)

$C_j - (1 + a_j)C_{j-1} = p'_j$ (18)

$C_j = \sum_{j=1}^{k} p'_j \prod_{r=j+1}^{k} (1 + a_r)$ (19)

$$C_{max} = \sum_{j=1}^{k} p'_j \prod_{r=j+1}^{k} (1 + a_r) \qquad (20)$$

Constraint (16) defines the total processing time of a deteriorating job. Constraint (17) is the expression of the completion time of a specific job j for index set $J = \{1,2, \dots, k\}$. Constraints (18) and (19) are expressions for the completion time of job j, with a deterioration rate. Constraint (20) defines C_{max}, the completion time of the last job processed on the machine.

3.2. Reentrant flowshop with deteriorating jobs model

To model the problem of the reentrant flowshop with deteriorating jobs, a linear programming model that incorporates the models presented earlier is formulated.

The assumptions made to solve the problem are as follows:

- All jobs are available from time zero (batch processing)
- Each machine can only process a single job at a time
- Jobs visit every machine in the same order $M_1, M_2, \dots M_n$ (permutation)
- No machine breakdowns
- All job processing times are known
- There is unlimited storage space for jobs waiting to be processed
- Pre-emption is not allowed
- Machine set-up times are included in the processing time

The model presented in this section considers a flowshop scheduling problem in which the jobs are both reentrant and the processing time deteriorates as a result of the delay before the commencement of processing of the jobs. This problem, hence, has the feature of the two models presented earlier in this chapter.

The linear programming model for the reentrant flowshop with deteriorating jobs is as follows:

Minimize C_{max} (21)

Subject to $\sum_{j=1}^{k} x_{ij} = 1 \qquad i = 1,2,\dots,k$ (22)

$\sum_{i=1}^{q} x_{ij} = 1 \qquad j = 1,2,\dots,k$ (23)

$s_{111} = 0$ (24)

$s_{1,1,i+1} = a_{11i}.s_{11i} + \sum_{j=1}^{k} x_{ij}p_{j11} \qquad i = 1,2,\dots,q-1$ (25)

$s_{1,l,i+1} \geq a_{1li}.s_{1li} + \sum_{j=1}^{k} x_{ij}\,p_{111} \qquad l = 2,3,\dots,b; i = 1,2,\dots,q-1$ (26)

$s_{m,l,i+1} \geq a_{mli}.s_{mli} + \sum_{j=1}^{k} x_{ij}\,p_{jlm} \qquad m = 2,3,\dots n; l = 1,2,\dots,b; i = 1,2,\dots,q-1$ (27)

$s_{1,l+1,1} \geq a_{1lq}.s_{1lq} + \sum_{j=1}^{k} x_{qj}\,p_{jl1} \qquad l = 1,2,\dots,b-1$ (28)

$s_{m,l+1,1} \geq a_{mlq}.s_{mlq} + \sum_{j=1}^{k} x_{qj}\,p_{jlm} \qquad m = 2,3,\dots n-1; l = 1,2,\dots,b-1$ (29)

$s_{m+1,1,1} = a_{m11}.s_{m11} + \sum_{j=1}^{k} x_{1j}\,p_{j1m} \qquad m = 1,2,\dots,n-1$ (30)

$s_{m+1,l+1,1} \geq a_{mlq}.s_{mlq} + \sum_{j=1}^{k} x_{qj}\,p_{jlm} \qquad m = 1,2,\dots,n-1; l = 1,2,\dots,b-1$ (31)

$s_{m+1,l,i} \geq a_{mli}s_{mli} + \sum_{j=1}^{k} x_{ij}\,p_{jlm} \quad m = 1,2,3; l = 1,2,\dots,b; i = 1,2,\dots,q; (l,i) \notin \{(1,1)\}$ (32)

$s_{m,l+1,i} \geq a_{nli}.s_{nli} + \sum_{j=1}^{k} x_{ij}\,p_{jln} \qquad l = 1,2,\dots,b-1; i = 1,2,\dots,q$ (33)

$C_{max} = \sum_{j=1}^{k} p_{jbn} \prod_{r=j+1}^{k}(1 + a_r)$ (34)

$C_{max} \geq 0, s_{mli} \geq 0 \quad m = 1,2,3; l = 1,2,\dots,b; i = 1,2,\dots,q;$

$x_{ij} = 0\ or\ 1 \quad i = 1,2,\dots,q; j = 1,2,\dots,k$ (35)

Equation (21) is the objective function. Constraints (22) and (23) describe the decision variables for the problem ensuring that each job is scheduled in only a single position and that a position has only a single job allocated to it, while constraint (24) describes the starting time of the first operation scheduled at the first level of the first machine. Constraints (25) and (30) define the starting time of any job, on any machine at the first operation level. Constraints (26) to (29) and

(31) to (33) describe the precedence of jobs in the process. Constraint (34) is the expression for calculating the makespan of the system, allowing for deterioration of processing time, while constraint (35) enforces the non-negativity and binary restrictions.

3.3. Solution algorithms

Various solution methods for scheduling flowshop problems were presented in the previous chapter. The following techniques, CDS, NEH, and GA, were selected for scheduling the flowshop problem with reentrance and deteriorating jobs. These methods have been proven to work well with large problem sizes and are relatively adaptable to various types of scheduling problems. The NEH was specifically selected to test submissions made by Ruiz and Maroto (2005) where they demonstrated that the NEH out-performs other algorithms and heuristics for scheduling problems, and by Baskar (2016) which indicated that even the simplest versions of the NEH algorithm are capable of arriving at an optimal solution.

The iconic CDS, NEH, and GA, with the incorporation of reentrance and deterioration, are presented as pseudo-codes 1, 2, and 3, respectively. The pseudo-codes are modifications to the original algorithms, made to accommodate reentrance and the deterioration of jobs.

Pseudo-code 1: CDS algorithm

```
//Create an aggregate processing time for each job for each machine across all levels
For every job across all levels $l = 1 \ to \ b$
    $p'_{jm} = \sum_{l=1}^{b} p_{jlm}$
EndFor
//solve $n - 1$ surrogate two machine problems
Set makespan to a large number
For $z$ varying from $1$ to $n - 1$
    For every job, $j$, on two pseudo machines, $M'_1$ and $M'_2$
        Create two surrogate processing times as follows
        $p'_{jm1} = \sum_{m=1}^{z} p'_{jm}$ as processing time of job $j$ on $M'_1$
        $p'_{jm2} = \sum_{m=n-z+1=1}^{n} p'_{jm}$ as processing time of job $j$ on $M'_2$
    EndFor
    Solve sequencing problem for $M'_1$ and $M'_2$ using Johnson's Algorithm
    If makespan from sequence is better
        Update makespan
        Update optimum sequence
    EndIf
EndFor
```

Johnson's Algorithm

```
Create two sets, $I$ and $II$ such that
    If $p'_{jm1} < p'_{jm2}$ allocate job to set $I$
    ElseIf $p'_{jm2} < p'_{jm1}$ allocate job to set $II$
    Else allocate to either set
    EndIf
Sort set $I$ in non-decreasing order of $p'_{jm1}$
Sort set $II$ in non-increasing order of $p'_{jm2}$
Append set $II$ to set $I$
```

Pseudo-code 2: NEH algorithm

```
For each job, $j$,
Determine the work content, $p'_j$, from $p'_j = \sum_{l=1}^{b} \sum_{m=1}^{n} p_{jlm}$
EndFor
Sort jobs into set $I$ in decreasing order of work content, $p'_j$,
Take the first two jobs out of set $I$ and form two partial sequences with them
Retain the partial sequence with the minimum makespan of the two
While there exists an unscheduled job in set $I$
    Remove the next job (with largest work content)
Form a set of new partial sequences by inserting the job in all possible positions in the currently retained partial sequence
    Retain one of the new partial sequence which has minimum makespan
EndWhile
```

Pseudo-code 3: GA

```
Set current iteration to 1
Generate initial population
Evaluate fitness and rank chromosomes based on makespan
Store the best chromosome
While current iteration is less than the required iterations
    Evaluate fitness and rank chromosomes based on makespan
    Update the best chromosome found
    Retain top performers and discard the remaining chromosomes
    Cross breed top performers to create new chromosomes make up population
    Mutate the top chromosomes
    Increment current iteration
EndWhile
```

The various components of GA, including the crossover and mutation operators that were modified to incorporate reentrance and deterioration, are discussed next.

Chromosome representation

A chromosome representation is designed as a vector of the same length as the number of jobs to be sequenced. It is created and populated with a random number between 1 and the total number of jobs. The representation is in such a way as to guarantee that every job is placed in exactly one position only in the vector.

Crossover Implementation

The crossover algorithm makes use of a logic termed herein as autogamy, meaning a chromosome is crossed with itself. The advantages of this procedure are twofold. The first is that it ensures that all jobs are placed precisely in one position after crossover. This is important because it is easy for some jobs to be missing while others are present in more than a single position after the crossover process. The second advantage is that it preserves some partial sequences of schedules that have performed well so that the benefit of promising sequences is not lost while trying to explore other areas during the search. The procedure randomly selects, through a probability mechanism, a breeding chromosome. It then randomly generates the crossover point. From the crossover point, two partial sequences (the head and the tail) are created. The tail partial sequence is swapped to the head position as a block while the head sequence is also swapped to the tail position as a block. This way, the promising partial sequences are preserved with minimum disturbance. If further mix in the sequence is desired, more than one crossover procedures may be implemented in a single reproduction process. How the partial sequences are repositioned may also alter the new sequences formed.

Mutation implementation

In implementing the mutation process, there is also the need to ensure that all jobs are sequenced in exactly one position. To do this, a swap procedure is implemented. Two random positions are generated within the sequence, and the jobs in these two positions are swapped. This procedure is repeated for the number of mutations required per chromosome.

It is apt at this point to recap what has been discussed in this chapter and provide a view of what comes next. This chapter focused on modelling the problem and was divided into three subsections. The first subsection concentrated on the review of models already presented in literature for reentrant flowshop and for jobs with deteriorating processing times. The second subsection was dedicated to developing a model which integrates the reentrance model and that of jobs with deteriorating schedules. The last subsection presented the modified algorithms as solution models for the problem of interest. The next chapter will address the design of model solution procedure and implementing same, i.e. generation of test data, conducting computational experiments and reporting on findings.

Chapter 4: Model solution

The model solution is composed of the experimental design, discussion of results and statistical analysis of model output. In the experimental design section, data generation methods previously used by researchers in the field of scheduling are evaluated to devise an approach for the problem of interest. The procedure for data generation used and the motivation for selecting it is explained. The results of the experiment on the three solution methods, CDS, NEH, and GA, are discussed. Since the data used for experimentation was randomly generated, the results are statistically analysed to determine the best performing solution method and the significance of the difference in the values achieved by the three algorithms.

4.1. Experimental design

Experimental design involves a discussion on the generation of data and the design of the experiments. Under data generation, the method of generating processing times and the rate of deterioration/ penalty to experiment with are discussed. The classification of test instances and implementation of the experiments are discussed under the design of experiments.

4.1.1. Data generation

There are various ways in which data for testing can be generated. It can either be done randomly, in a way that the test instances are similar to real manufacturing (Choi and Kim, 2007) or by benchmarking based on previous researchers' studies. Reeves (1995), Ruiz and Maroto (2005) and Baskar (2016), have all used Taillard's benchmark instances in their studies. Another option of data generation is the use of data for which an optimal solution is known. Chen (2006) used a set of data with known optimal solutions to compare the new methods proposed with those previously tested by other researchers. With no study known to date which integrates the reentrant problem with deteriorating jobs, the approach of the use of data with known optimal solutions was eliminated. Alternative methods for data generation were thus considered for this study.

Data generation for processing times

In generating data for experimentation, the practical number of machines and operational levels to study in relation to the type of manufacturing environment had to be considered. Test data sizes and the method of data generation used by Chen et al. (2007) for minimizing makespan in reentrant flowshops were adopted. The method involves the use of a wide range of data sizes. It tackles scheduling of jobs in manufacturing facilities with as few as two operational levels to as much as ten operational levels. Widening the test data range allows for observations to be made on the effect that manufacturing facility configurations have on the makespan.

Parameters used in the experimental environment are described as $k\,x\,n\,x\,b$; where k represents the number of jobs, n the number of machines and b the number of operational levels. The test data is divided into three categories: small, medium, and large problem sizes. The small problems are composed of eight matrix sizes: $3x3x3$, $3x3x4$, $3x4x2$, $4x3x3$, $4x4x3$, $4x5x3$, $4x4x4$, and $4x5x4$. Medium problems also consist of eight matrix sizes: $6x6x2$, $6x8x5$, $6x9x3$, $7x7x5$, $7x8x4$, $8x8x3$, $9x9x2$, and $10x10x2$. The large problems include five matrix sizes: $12x12x10$, $15x15x5$, $20x20x4$, $25x25x8$, and $30x30x5$. Data was generated randomly as no benchmark data was available. Two types of data sets were generated. The first set of processing times was generated in the range [1, 100] on all machines since most benchmark data is generated within this range (Chen et al., 2007). The set was then termed the same data range (SDR) set. This is because in this first approach to data generation, every machine involved in the scheduling process has a possible lower bound of 1 time unit and an upper bound of 100 time units. This is unlike the second approach that was termed the unique data range (UDR) in which each of the machines has different lower and upper bounds of data range, but all these bounds were between zero and hundred. The second set was generated by first randomly setting unique upper and lower bounds for the processing times of each machine in within the range [1, 100]. Every data point for such machine is then generated within its own randomly generated lower and upper bounds. An example of the upper bound (UB) and the lower bound (LB) setting for a $3x3x3$ matrix is illustrated in *Table 2*.

Table 2: Example of bounds setting for a 3x3x3 matrix

Machine 1		Machine 2		Machine 3	
UB	**LB**	**UB**	**LB**	**UB**	**LB**
71	14	88	72	59	70

These unique data ranges were then applied to all jobs on all operational levels and, hence, called the unique data range (UDR) set. *Table 3* shows an example of an instance full set of a $3x3x3$ processing time matrix.

Table 3: Example of processing times for a 3x3x3 matrix

Levels →	**1**			**2**			**3**		
Machines→ Jobs↓	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
1	63	84	66	22	86	65	21	75	62
2	33	73	63	18	75	66	31	81	61
3	67	80	70	57	88	70	35	84	70

A set of five instances of such processing time matrices was generated for each of the data set types (i.e. SDR and UDR) for each of the specified matrix dimensions, $k\ x\ n\ x\ b$, mentioned in each of the categories of problem sizes. The unique processing time ranges per machine were introduced to mimic typical manufacturing. In manufacturing, machines often have specific processing times based on their function.

Data generation for penalty matrices

Penalty matrices were generated to evaluate the effect of deterioration of processing times. The matrices were generated randomly as no benchmark data for a similar problem is available. Hindi and Mhlanga (2001), Jafari et al. (2017), Wang et al. (2006), Mazdeh et al. (2010), and Wang et al. (2011) all used job deterioration rates in the range [0, 1]. Jobs in a manufacturing facility may deteriorate at various rates. Thus Wang et al. (2011), Jafari et al. (2017), and Mazdeh et al. (2010) incorporated this in their studies by subjecting jobs to different deterioration rates within a selected range.

In this study of scheduling a reentrant flowshop with deteriorating jobs, a range of [0, 0.1] was used. This range was selected to test the effect of a rate that was considered as small by Ng et al. (2010) in their study of a two-machine flowshop problem with deteriorating jobs. Four delay scenarios were created within this range (*see Table 4*).

Table 4: Scenarios for delay penalty

Scenario #	Scenario	Details
1	No penalty	To observe the effect of job reentrance system without deterioration (delay penalty).
2	Penalty of 0.1 only on the first machine in the first operational level for jobs not scheduled first	To determine the effect of uniformly penalising delay only on the first machine.
3	0.1 penalty on the first machine and 0.05 penalty for all other jobs not scheduled first	To determine the effect of uniformly penalising all delays on all machines, with the delay on the first machine incurring a higher penalty
4	Random penalty in the range [0.01, 0.1] on all jobs not scheduled first	To determine the effect of random deterioration rates on all machines.

4.1.2. Design of experiment

Development of test instances

A test instance refers to a computation conducted on a particular processing time matrix. Computations were run for all four penalty scenarios on CDS, NEH and GA algorithms for the three test categories (i.e. small, medium and large problems). Unlike the CDS and NEH heuristics, the GA algorithm requires more information in addition to the processing time and deterioration rate. The additional GA parameters are discussed next.

GA parameter setting

Parameters required for the execution of the GA include population size, mutation rate, retention rate and the number of iterations (stopping criteria). The same mutation and retention rates were used for all problem sizes. In this section, a value of 0.5 was used for the retention and mutation rate.

Table 5: Parameter setting for GA

Problem size	**Matrix size**	**Population size**	**Number of iterations**
Small	3x3x3	4	10
	3x3x4	4	10
	3x4x2	5	10
	4x3x3	4	10
	4x4x3	4	10
	4x5x3	10	30
	4x4x4	8	30
	4x5x4	10	30
Medium	6x6x2	100	30
	6x8x5	150	30
	6x9x3	200	30
	7x7x5	150	30
	7x8x4	150	30
	8x8x3	200	60
	9x9x2	300	150
	10x10x2	500	150
Large	12x12x10	600	250
	15x15x5	800	350
	20x20x4	1000	700
	25x25x8	1200	700
	30x30x5	1500	700

The sensitivity of the solution to this rate is presented in a subsequent section. The population sizes and the number of iterations were varied for the different matrix sizes. The population sizes were chosen in relation to the number of machines for the matrix. For the small problems, fewer population sizes were used. Problems with fewer machines tend to repeat chromosomes due to the limited number of possible combinations. Limiting the population size to a small number eliminates this challenge. The number of iterations for each matrix size was selected if the minimum makespan values appear always to converge (or stabilise) within the chosen number of runs. Detailed parameters used for each problem size are presented in *Table 5*.

GA parameter variation

The influence of input parameters on the performance of the GA algorithm was studied by conducting simulations with parameter changes for mutation rate and population size. The effect of the mutation rate was evaluated while keeping all other parameters (i.e. population size, retention rate and the number of iterations) constant. Likewise, the effect of varying the population was evaluated while keeping all other parameters constant. In addition to testing a mutation rate of 0.5; rates of 0.3 and 0.7 were applied. The selected rates are a chosen to be 50 percent higher and lower than the initially selected mutation rate. The population sizes were also varied for selected matrix sizes. The effects of changes in this parameter were studied with half and then with a quarter of the initial population to maintain uniformity to understand how this might affect the speed of convergence of the algorithm and the quality of solution obtained. The parameter variation was only performed on the large problem sizes for scenarios 3 and 4 as these are expected to be the most significant data sets that would be influenced by these changes as the smaller matrices all return identical results in identical time for all previous experiments.

Implementation

The three algorithms, CDS, NEH, and GA, were coded in MATLAB R2019a, and all computational experiments were performed with these codes. The experiment was run on all categories of test instances. For each category, computations for makespan, subject to varying degrees of penalty described in *Table 4* were performed using the three algorithms.

During experimentation, the minimum makespan values achieved and their corresponding computation times were recorded. Graphs for the comparison of the minimum makespan values achieved by the three solution methods were plotted in MATLAB. Additionally, computations were performed for GA, with varied test parameters. The performance indicators evaluated for the parameter variations are the minimum makespan values achieved, their corresponding computational times and the speed of convergence of the solution. Graphs of the makespan values achieved for varied test parameters were plotted. Lastly, graphs demonstrating how the minimum makespan converges with the number of iterations were plotted for GA.

The makespan output analysis and statistical testing were performed in MS Excel. The output analysis was conducted by applying two measures; the count of the number of times that any given algorithm returned the lowest makespan value and the average proportion above the minimum makespan when an algorithm did not return the lowest makespan value. The procedure followed in executing the counting exercise involved several steps. It was mentioned earlier that data instances were generated for each type of data range (SDR and UDR) for a matrix dimension ($k\ x\ n\ x\ b$). The five makespan values returned for each matrix size per data range were evaluated. A value of one was allocated for the test instances that returned the lowest makespan value, and zero otherwise. The count of instances is, however, not mutually exclusive per test algorithm; i.e. a test instance of one algorithm is counted for as long as it returned the lowest value, regardless of another algorithm being counted for the same test instance if it also returns the same value.

The procedure for determining the average value above the minimum makespan was also stepwise. The minimum makespan value of the three algorithms for each of the matrix sizes and data range was determined. This was then termed the global minimum. The average of the five instances of makespan values was calculated for each matrix size within the data range for the three algorithms. The proportional value above the global minimum was then calculated by subtracting the global minimum from the average makespan for the instance and then dividing by the global minimum.

Statistical testing was applied to all instances of makespan values for each of the matrix sizes. The t-test was selected as the test of choice due to the random nature of the data and the small sample sizes being evaluated. Two sets of t-tests were performed; one for the count of minimum makespan values and the other one for the proportion of values above the global minimum makespan. The paired t-test for sample means was executed for the counts of minimum makespan and the proportion above minimum makespan.

4.2. Results

The presentation of the results is divided into three parts; minimum makespan, average computation times and the effects of the GA parameter variation. The minimum makespan and computational time results compare values achieved under the different test instances. The GA parameter variation part focuses on the results for computations conducted for the various mutation rates and population sizes. The influence of the varied test parameters on the makespan values attained and their corresponding computation times are evaluated.

4.2.2. Minimum makespan attainment

The makespan values obtained for all test categories using the different solution algorithms were plotted against each other to determine the best performing method as test conditions change. *Figure 4* to *Figure 8* are plots for the minimum makespan values of penalty scenario 2 (i.e. deterioration only on the first machine). Plots for other penalty scenarios are documented in *Annexure A*. The discussion of the makespan values achieved is sub-sectioned based on the various problem sizes.

Small problems

Since five instances were generated for each matrix dimension, the minimum makespan for set of data for each matrix was determined. The three algorithms returned similar minimum makespan values for the majority of the small problem sizes. Penalty scenario 3 (i.e. uniform deterioration on all machines, with the first machine incurring the highest penalty) and scenario 4 (i.e. random deterioration on all machines) resulted in significant increase in the minimum makespan values for every given problem size. The increase in minimum makespan values is attributed to the fact that all the machines are subjected to deterioration. The NEH and GA performed better than CDS as test conditions changed, i.e. increasing the number of operations and subjecting the processing times to deterioration.

Medium problems

NEH and GA returned the lowest makespan values for the majority of the test instances of the medium problem sizes as opposed to CDS. Subjecting the system to a penalty on all operational levels, with the CDS algorithm returns makespan values that are up to twice as those achieved by the other two algorithms.

Large problems

The CDS algorithm could not find the minimum makespan values for all test instances of the large problem sizes, (i.e. no deterioration). Introducing penalty on all operational levels (i.e. scenarios 3 and 4) for the large problem sizes widens the gap between the makespan values achieved by CDS and those by NEH and GA even further as compared to the small and medium problem sizes. Generally, the NEH and GA solution methods returned the lowest makespan values when compared to CDS as test conditions changed.

The NEH and GA algorithms appear to successfully return minimum makespan values for various problem sizes and penalty scenarios. GA, being a meta-heuristic method, strives to improve the solution with each iteration. The NEH's functionality, insertion, also ensures that only the best solutions are retained. The computation times required to reach the minimum makespan values for the tested solution methods are addressed next.

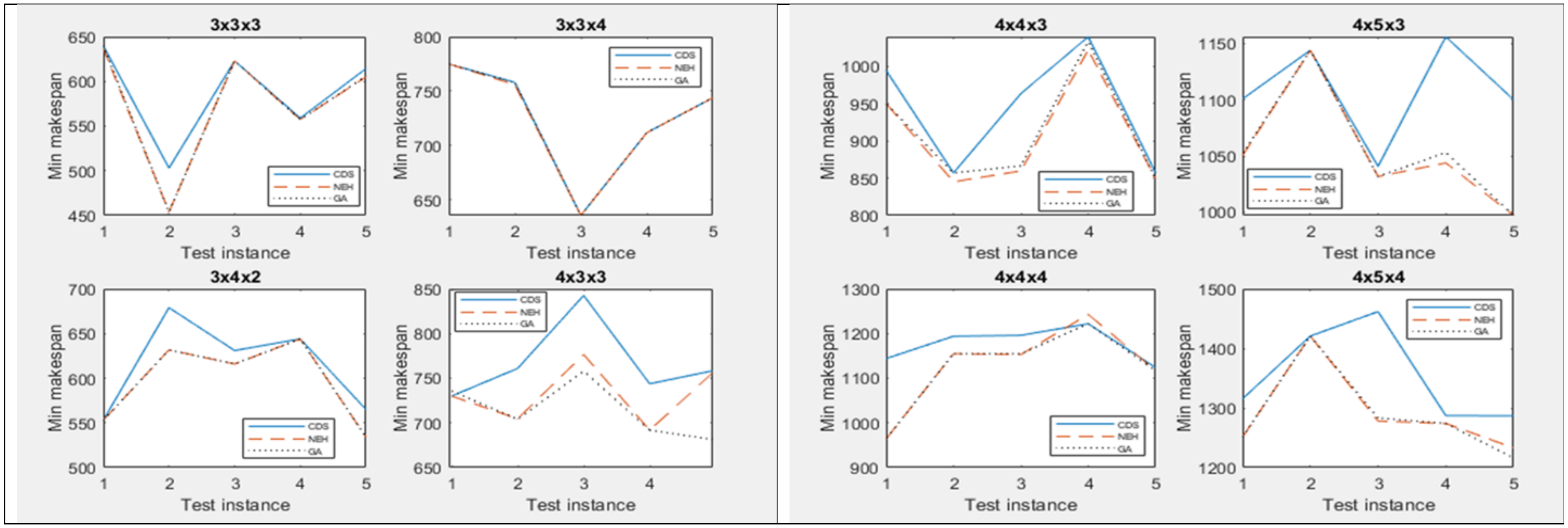


Figure 4: Minimum makespan plots for small problems with same data range for scenario 2

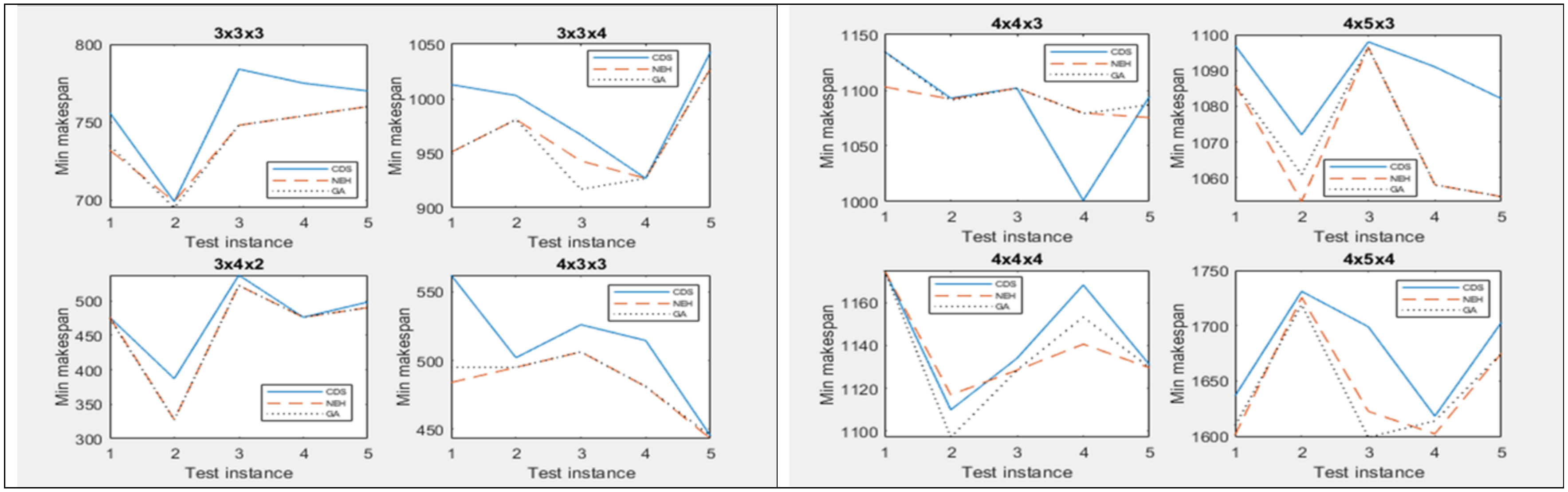


Figure 5: Minimum makespan plots for small problems with unique data range for scenario 2

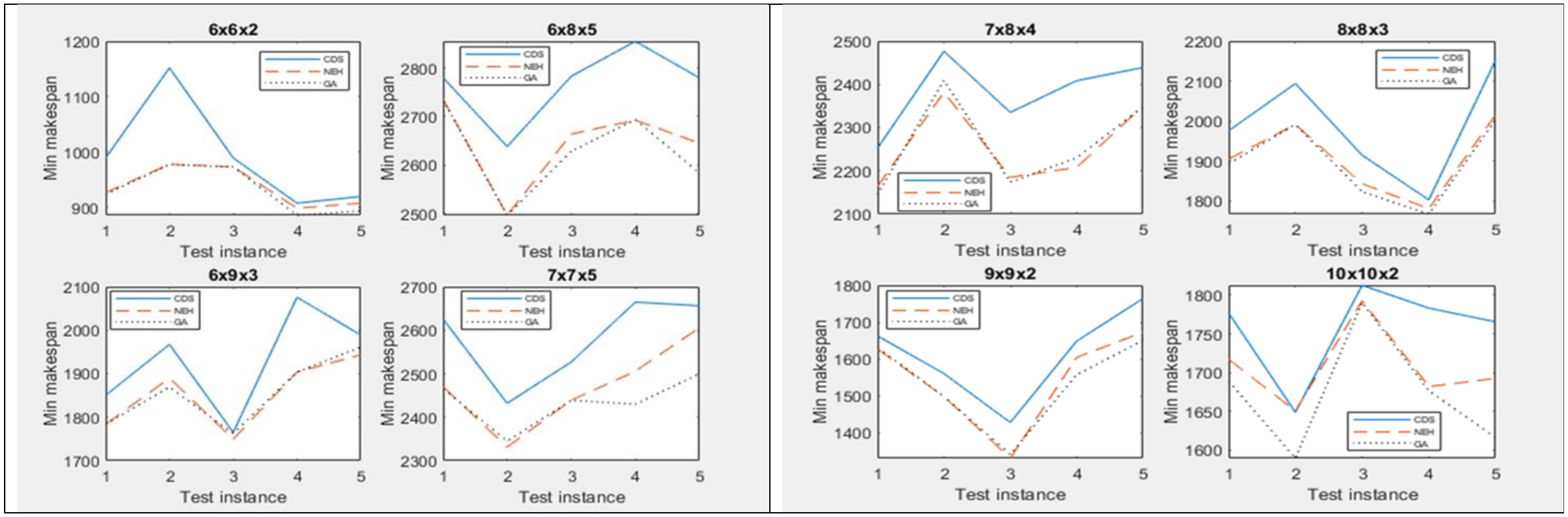


Figure 6: Minimum makespan plots for medium problems with same data range for scenario 2

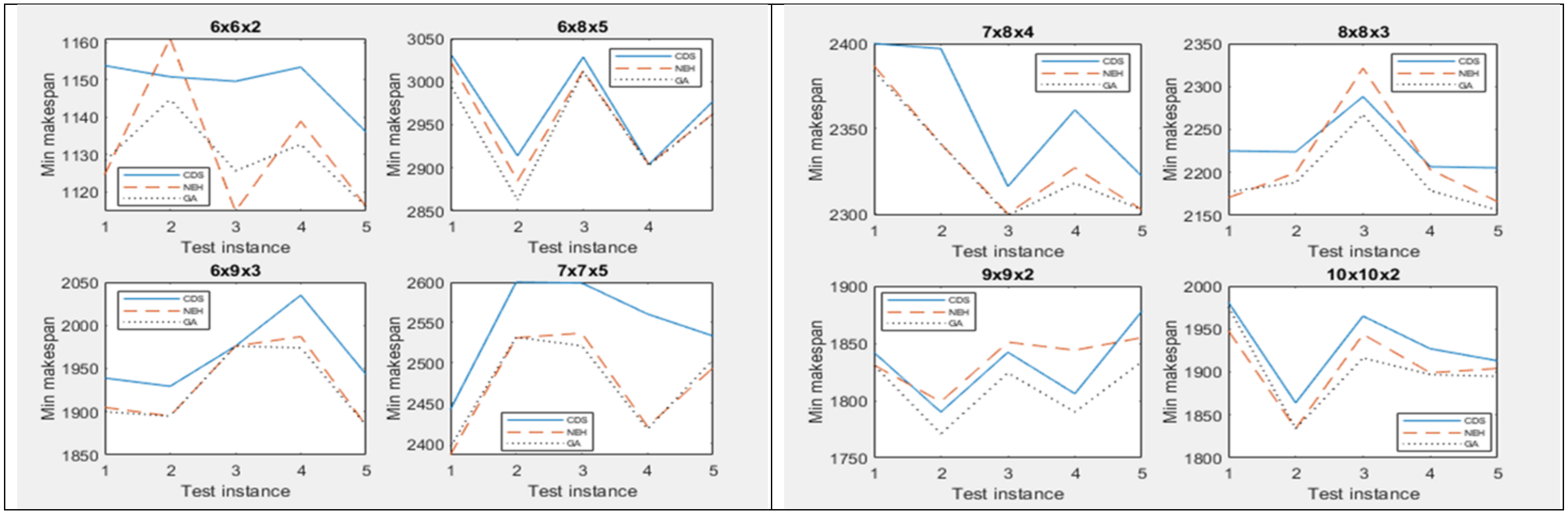


Figure 7: Minimum makespan plots for medium problems with unique data range for scenario 2

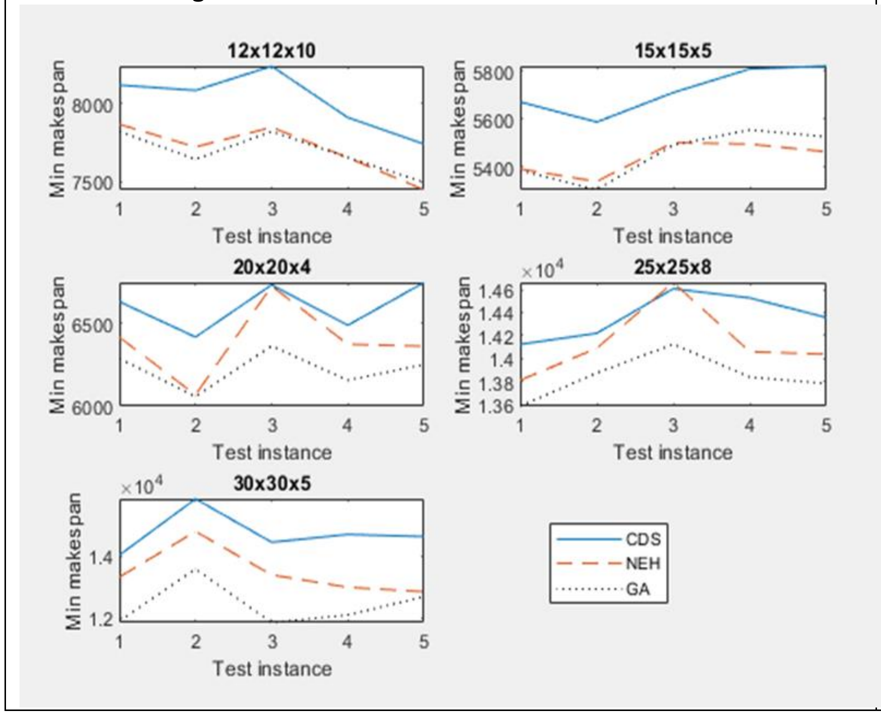
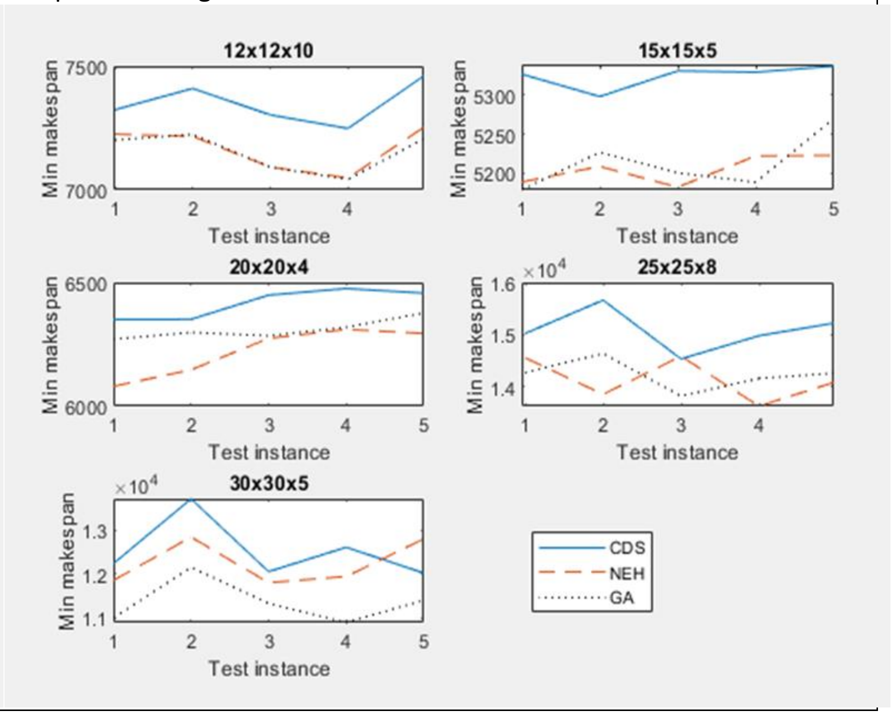
Same data range	Unique data range
	

Figure 8: Minimum makespan plots for large problems with scenario 2

4.2.3. Average computation times

The average computation times, indicating the minimum and maximum times attained by each solution method, are presented in *Table 6*. Detailed computation times for all penalty scenarios are reported in *Annexure C*. The CDS algorithm had the shortest computation times, followed by NEH. The computation times increased slightly from the small problem size to the large problems for both the CDS and NEH algorithms for all penalty scenarios. The GA algorithm had the longest computation times for the large problem size. The time complexity for GA is influenced by the large population sizes and the large number of iterations related to the problem sizes.

Table 6: Average computation times

Penalty type	Solution method	Small problems		Medium problems		Large problems	
		Min computation time	Max computation time	Min computation time	Max computation time	Min computation time	Max computation time
Scenario 1	CDS	0.0003	0.0393	0.0005	0.1402	0.0043	0.1017
	NEH	0.0002	0.0153	0.0006	0.0075	0.0157	0.5409
	GA	0.0252	0.2008	0.0993	0.8794	38	872
Scenario 2	CDS	0.0003	0.0890	0.0005	0.0650	0.0044	0.0993
	NEH	0.0003	0.0282	0.0009	0.0234	0.0166	0.2481
	GA	0.0251	0.3108	0.1589	7	42	915
Scenario 3	CDS	0.0003	0.0789	0.0005	0.0495	0.0051	0.0828
	NEH	0.0003	0.0558	0.0008	0.0995	0.0178	0.2942
	GA	0.0242	0.1700	0.1348	6	40	970
Scenario 4	CDS	0.0003	0.0468	0.0005	0.1554	0.0051	0.1385
	NEH	0.0003	0.0073	0.0007	0.0059	0.0174	0.3294
	GA	0.0313	0.1935	0.1735	7	46	874

4.2.4. GA parameter variation

The parameter variation aims to demonstrate the influence of varying the GA test parameters on the minimum makespan and the corresponding computation times to find the solutions. The convergence of the solution to a minimum makespan value is evaluated to determine if changing the test parameters can result in a better solution within a reasonable or shorter computation time. The minimum makespan results are presented graphically, while the computation times are tabulated. Plots for the rate of convergence are also presented.

Influence of mutation rate (mutrate)

The initial mutation rate used was 0.5, while 0.3 and 0.7 were the selected rates to use in evaluating the reaction of the algorithm to changes in mutation rate. In some cases for scenario 3 ($30x30x5$ for SDR set and $20x20x4$ for UDR set), the same makespan values were recorded for all mutation rates (*see Figure 9*). The observation made for scenario 4 in *Figure 10* is that the differences in makespan values among the various mutation rates become apparent as the number of machines and levels increase. Overall, the mutation rate of 0.3 resulted in slightly lower makespan values than the other two mutation rates. The computation times for the mutation rate of 0.3 are, however, in the same range as that for the other mutation rates used. The detailed computation times are presented in *Table 10*. Using a mutation rate of 0.3 seems to sometime achieve some marginally lower makespan value at about the same computation time, but such is seems not very significant.

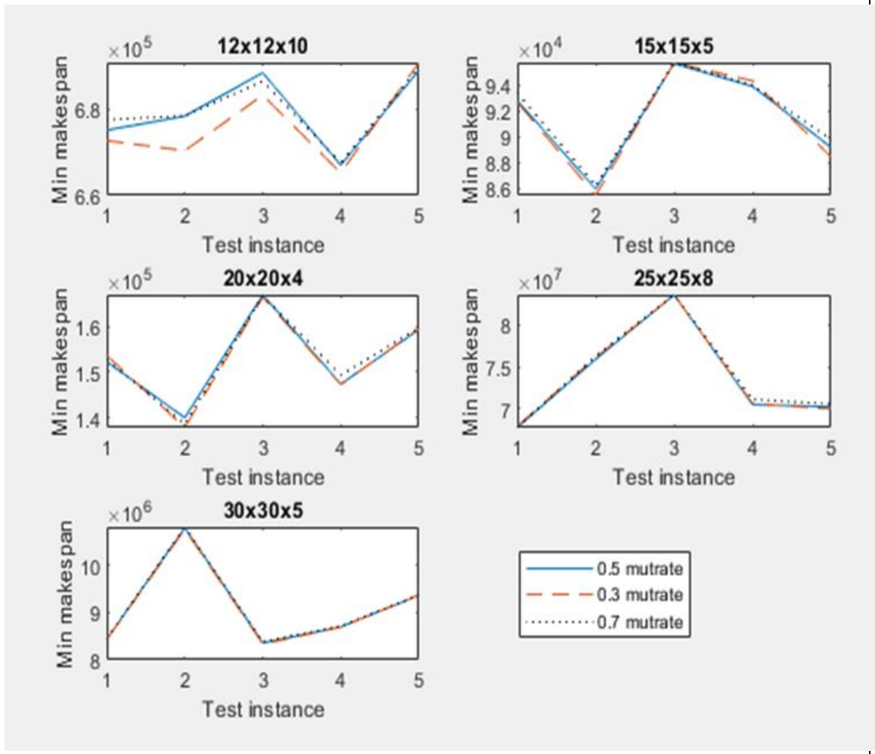
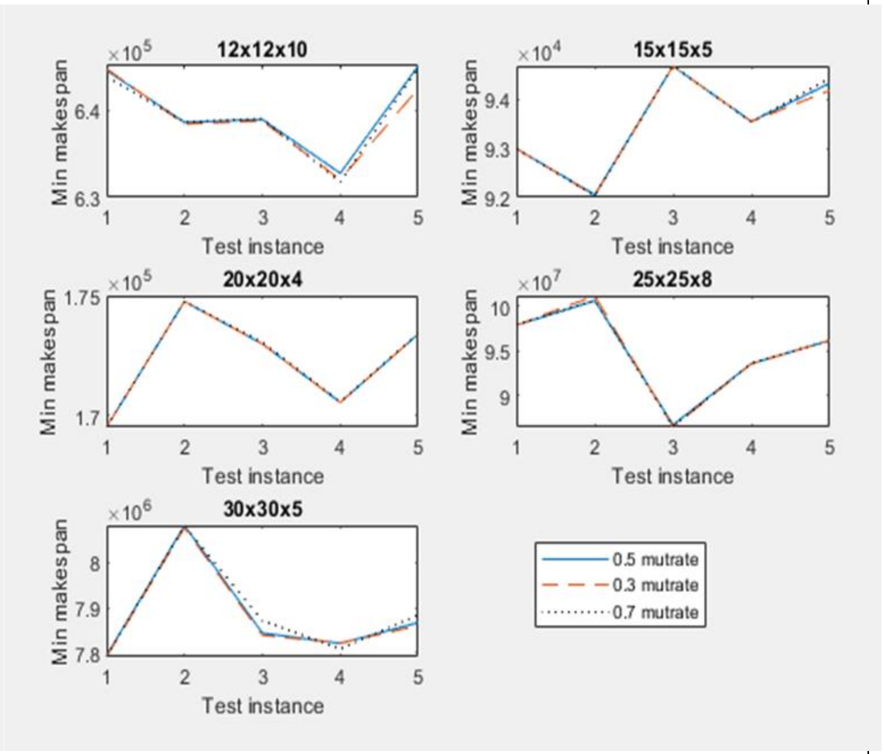
Same data range	Unique data range
	

Figure 9: Minimum makespan of various mutation rates with scenario 3

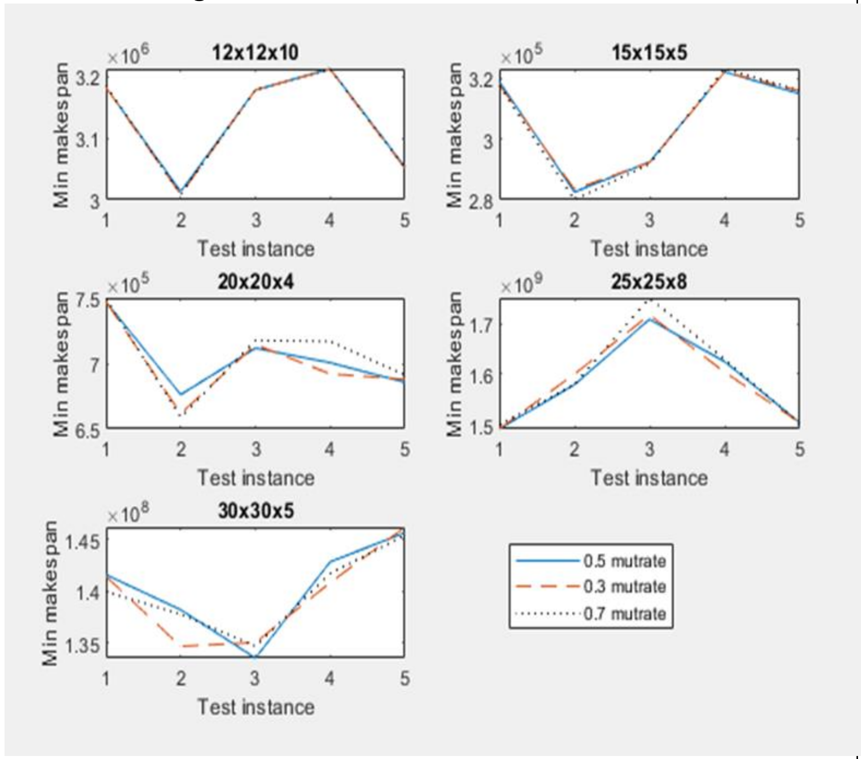
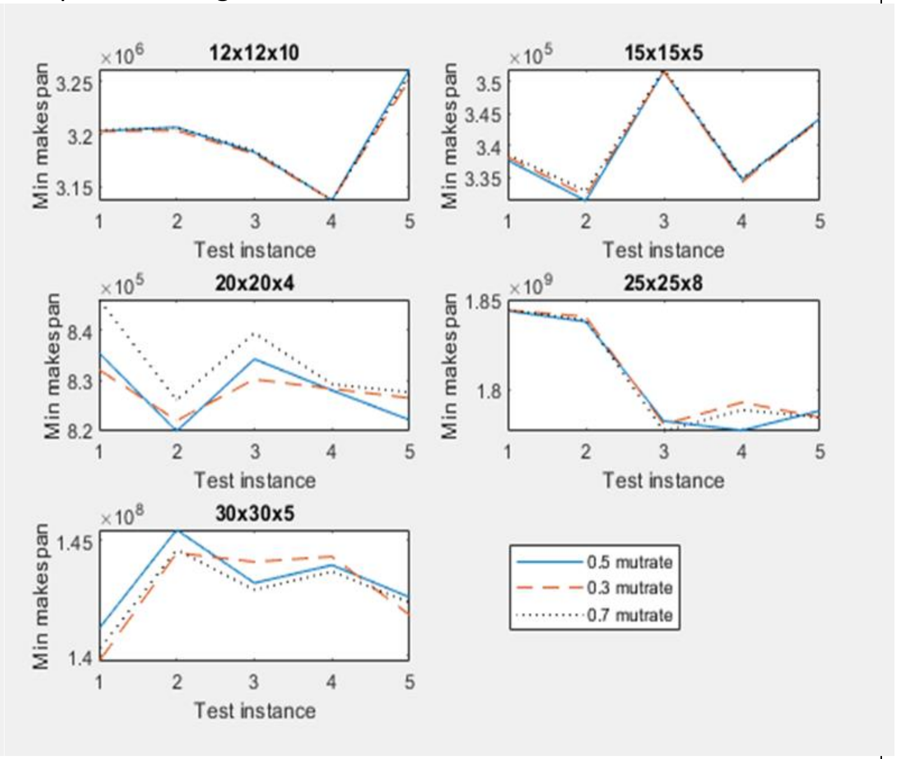
Same data range	Unique data range
	

Figure 10: Minimum makespan for various mutation rates with scenario 4

Table 7: Average computation times for various mutation rates

Problem size	Scenario 3						Scenario 4					
	Same data range			Unique data range			Same data range			Unique data range		
	0.5 mutrate	0.3 mutrate	0.7 mutrate	0.5 mutrate	0.3 mutrate	0.7 mutrate	0.5 mutrate	0.3 mutrate	0.7 mutrate	0.5 mutrate	0.3 mutrate	0.7 mutrate
12x12x10	40.2	40.4	41.6	40.7	38.9	42.1	46.6	45.0	38.8	38.1	44.9	38.4
15x15x5	68.5	68.8	39.6	41.1	52.1	48.9	81.8	80.7	36.8	37.0	36.9	37.2
20x20x4	286.7	233.1	174.0	174.5	166.8	159.7	255.6	235.5	165.0	163.1	162.8	162.5
25x25x8	750.8	941.2	615.4	592.0	709.4	517.7	798.6	761.7	550.8	544.7	525.1	581.5
30x30x5	970.2	862.9	828.4	820.2	812.0	902.2	867.2	874.3	834.2	761.3	814.5	810.8

Influence of population size

The initial population sizes differed for the various problem sizes experimented on, and thus the reduced population sizes were also different. The reduction of the population size for a genetic algorithm implies that there are fewer data points to operate on. The $12x12x10$ matrix had slight differences in the makespan values attained for the different population sizes for scenario 3. The other matrix sizes, also subject to the same penalty, however, did not exhibit any significant difference in the makespan values achieved. Some of the instances of the bigger matrices (i.e. $20x20x4$, $25x25x8$ and $30x30x5$) returned the same makespan values for penalty scenario 3. It appears that randomising the penalty (i.e. scenario 4) also plays a role in the minimum makespan value attained. Slight differences in the makespan values were noticed for the bigger matrix sizes. *Table 8* lists the average computation times, and as would be expected, there is a considerable reduction in the computation times for the smaller population sizes.

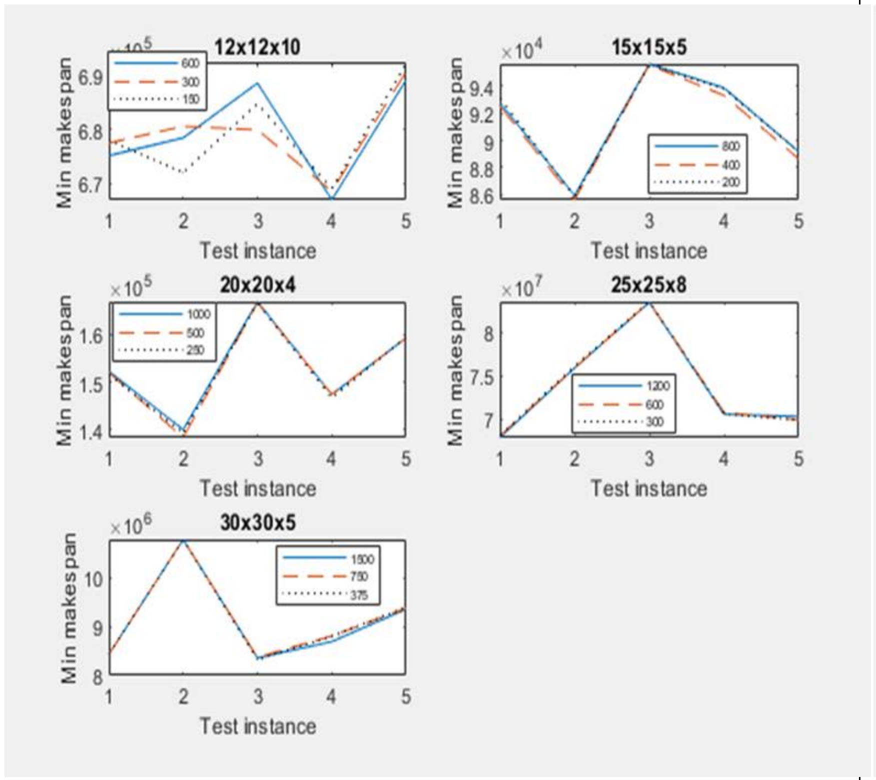
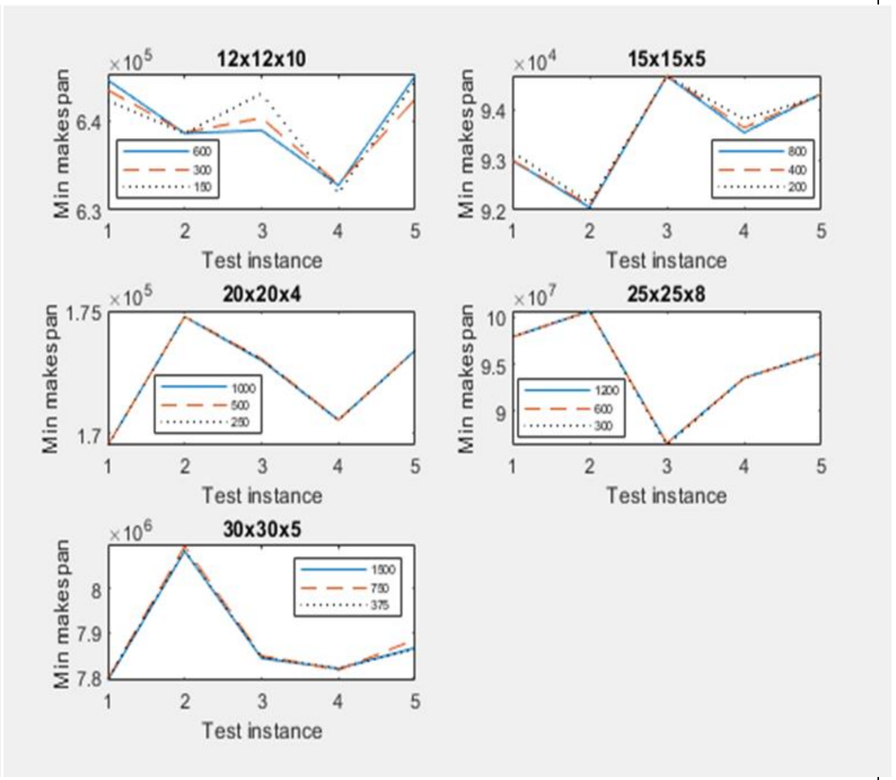
Same data range	Unique data range
	

Figure 11: Minimum makespan for various population sizes with scenario 3

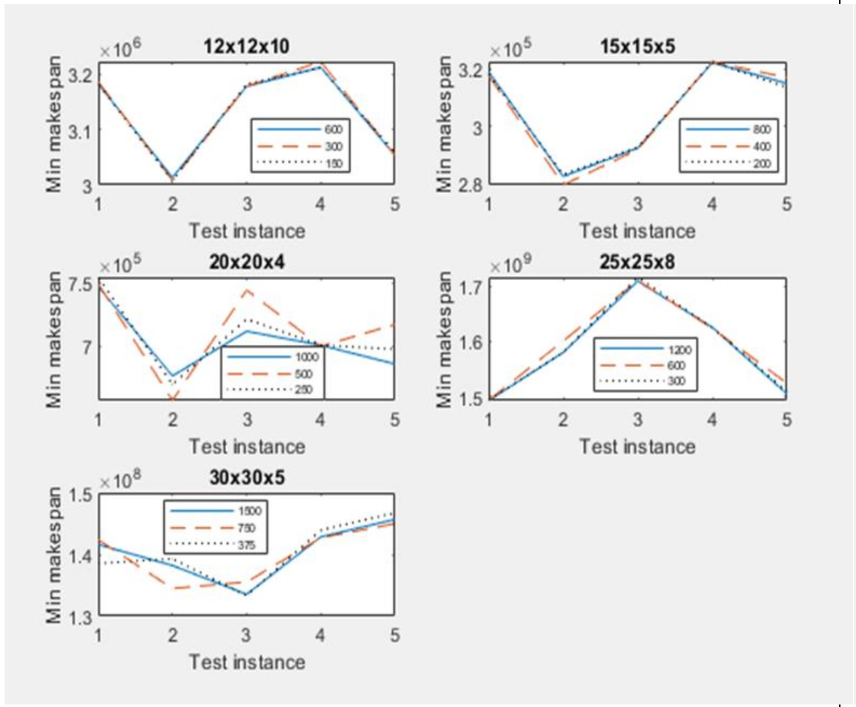
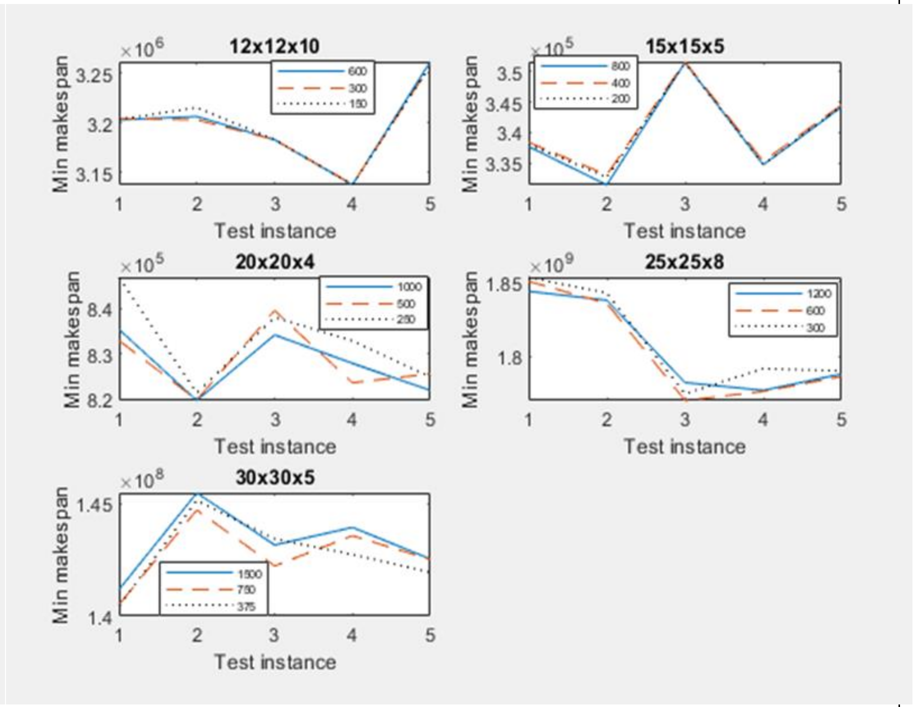
Same data range	Unique data range
	

Figure 12: Minimum makespan of various population sizes with scenario 4

Table 8: Average computation times for various population sizes

Problem size	Scenario 3						Scenario 4					
	Same data range			Unique data range			Same data range			Unique data range		
	Initial population	½ Initial Pop.	¼ Initial Pop.	Initial population	½ Initial Pop.	¼ Initial Pop.	Initial population	½ Initial Pop.	¼ Initial Pop.	Initial population	½ Initial Pop.	¼ Initial Pop.
12x12x10	40.2	40.4	19.2	18.2	9.4	9.4	46.6	45.0	17.6	18.2	8.6	8.6
15x15x5	68.5	68.8	16.9	17.9	8.5	8.4	81.8	80.7	16.7	17.0	7.7	7.8
20x20x4	286.7	233.1	79.8	78.2	34.8	33.9	255.6	235.5	72.8	73.0	34.8	34.0
25x25x8	750.8	941.2	282.5	280.8	122.9	122.1	798.6	761.7	248.2	257.2	147.6	135.2
30x30x5	970.2	862.9	177.7	182.3	361.2	346.4	867.2	874.3	360.9	351.9	175.1	165.9

Solution convergence

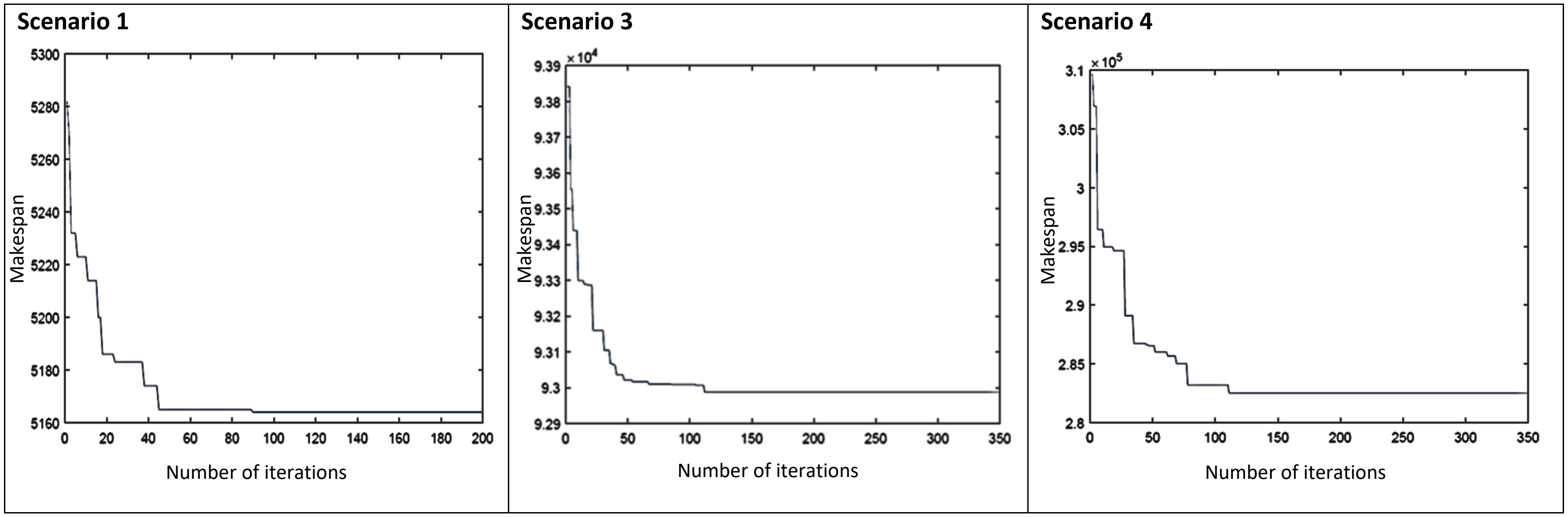


Figure 13: GA solution convergence for a 15x15x5 problem size with various penalty scenarios

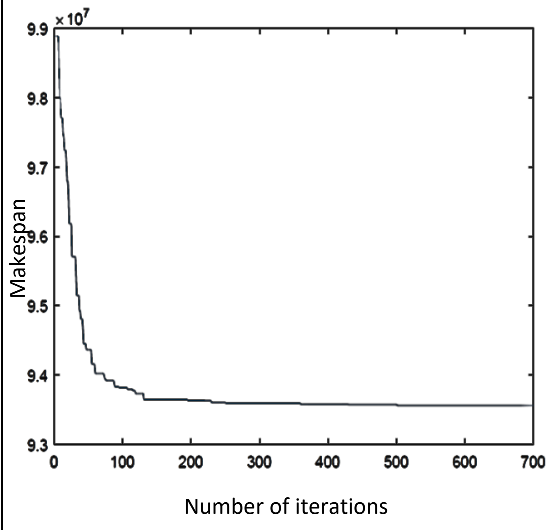
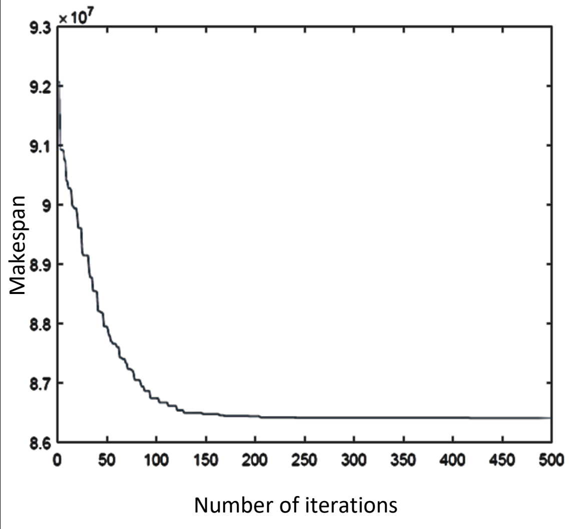
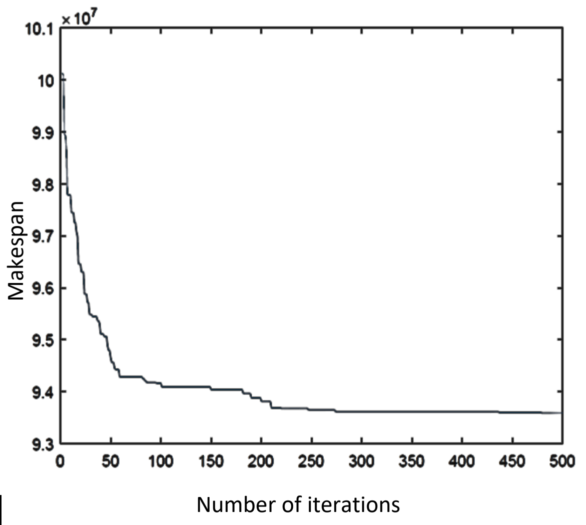
Mutation rate = 0.5	Mutation rate = 0.3	Mutation rate = 0.7
		

Figure 14: GA solution convergence for a 25x25x8 matrix for various mutation rates with scenario 3

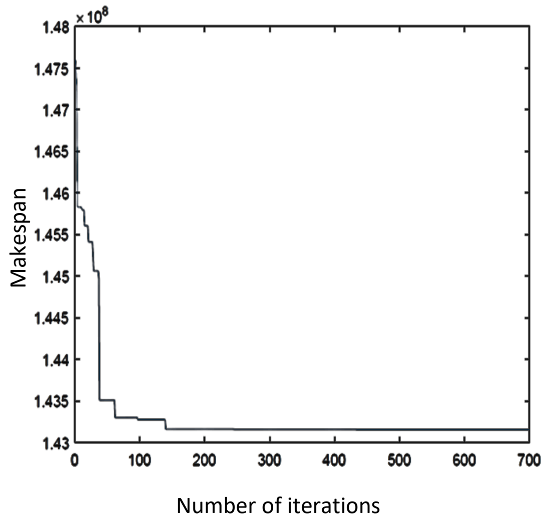
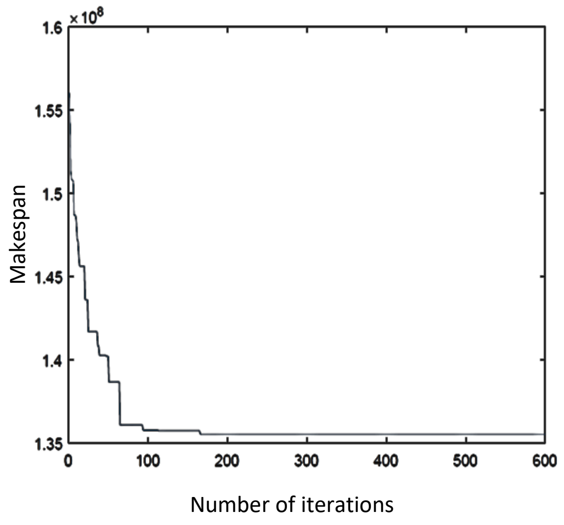
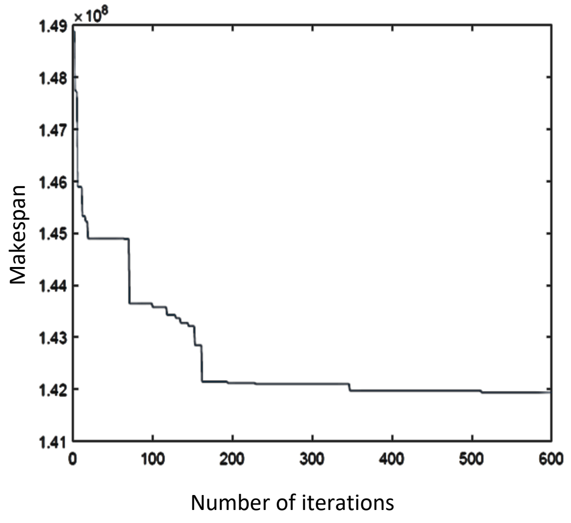
Population size = 1500	**Population size = 750**	**Population size = 375**
		

Figure 15: GA solution convergence for a 30x30x5 matrix for various population sizes with scenario 4

To determine the effect of the different parameters on the convergence of the minimum makespan, graphs for the minimum makespan values were plotted. The parameters of interest, in this case, are penalty, mutation rate and population size. *Figure 13* to *Figure 15*, together with figures in *Annexure D*, are used to demonstrate this effect.

The solution converged to a minimum value quicker for scenario 1 as compared to scenarios 3 and 4; i.e. subjecting the scheduling problem to a penalty lengthens the time to reach the lowest makespan value. The time to converge for the two deterioration penalties (scenarios 3 and 4) are similar.

The mutation rate of 0.3 led to a faster convergence for scenario 3 than scenario 4, and the opposite was observed for the mutation rates of 0.5 and 0.7. No definite trend can be observed on how the various mutation rates affect the convergence and stability of the minimum makespan value.

Figure 15 represents convergence in relation to the reduction of the population of the initial computation. A reduction of the population size by half does not seem to lead to a significant difference in the time taken for the model to converge, but reducing it to a quarter of the initial population does. It should be noted, however, that such convergence seems to sacrifice the quality of solution in a number of instances as the minimum makespan value returned is higher than what was initially observed when the original population size was used.

4.3. Statistical analysis

Statistical analysis is divided into two sections, makespan output analysis and statistical test of significance of differences. The makespan output analysis was conducted for computation of the three algorithms and also for GA with varied parameters.

4.3.2. Makespan output analysis

The total number of counts of when a solution method results in the lowest makespan value for the various test instances is presented in *Table 9*. The asterisks beside the counts indicate the algorithm(s) with the most number of lowest makespan for each test category. It should be noted that the maximum count obtainable for small and medium-sized problems is 40, while for large-sized problems it is 25. Results showing the counts per matrix size are documented in *Annexure B*.

Table 9: Count of minimum makespan attainment

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS	NEH	GA	CDS	NEH	GA
Scenario 1	Small	14	36	39*	8	34	35*
	Medium	0	20	34*	1	11	39*
	Large	0	15*	10	0	19*	7
Scenario 2	Small	11	35*	29	5	32*	30
	Medium	0	16	32*	1	16	33*
	Large	0	4	21*	0	13*	13*
Scenario 3	Small	0	33*	10	0	40*	3
	Medium	0	27*	13	0	32*	9
	Large	0	8	17*	0	18*	7
Scenario 4	Small	0	33*	8	0	33*	8
	Medium	0	10	30*	0	23*	17
	Large	0	10	15*	0	15*	10

The CDS algorithm reported several test instances that had the lowest makespan values for the small problems of scenario 1 (i.e. no deterioration) and scenario 2 (i.e. deterioration only on the first machine) for the SDR data set. The CDS algorithm also returned a few instances with the lowest makespan values for the small and medium problems of scenarios 1 and 2 for the UDR data set. Overall, the NEH and GA algorithms returned the most number of the lowest makespan values. For the SDR data set, GA returned more instances of the lowest makespan values than NEH. On the other hand, the NEH algorithm returned the most instances for the UDR data set, while dominating scenario 3 (i.e. uniform deterioration on all machines, with the first machine incurring the highest penalty) and scenario 4 (i.e. random deterioration on all machines).

Table 10: Average proportion above global minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS	NEH	GA	CDS	NEH	GA
Scenario 1	Small	0.157	0.114	0.113*	0.116	0.089*	0.089*
	Medium	0.121	0.073	0.067*	0.038	0.023	0.019*
	Large	0.076	0.021*	0.024	0.036	0.010*	0.011
Scenario 2	Small	0.159	0.119	0.117*	0.122	0.097*	0.097*
	Medium	0.124	0.076	0.068*	0.041	0.028	0.024*
	Large	0.109	0.058	0.031*	0.077	0.042	0.029*
Scenario 3	Small	0.893	0.178*	0.269	0.738	0.088*	0.406
	Medium	2.189	0.070*	0.085	2.124	0.055*	0.075
	Large	6264	1.029*	1.605	3264	0.050*	0.069
Scenario 4	Small	1.209	0.184*	0.247	1.013	0.105*	0.174
	Medium	4.097	0.125*	0.317	4.320	0.093*	0.109
	Large	99723	0.195	0.100*	100254	0.197	0.182*

The proportional values above the global minimum for the CDS algorithm got significantly large for the medium and large problem sizes of scenarios 3 and 4 as compared to scenarios 1 and 2 for both the SDR and UDR data sets. The NEH and GA returned the lowest proportional values, for the same number of instances of the SDR data set. The NEH, however, had the lowest proportional values, mostly for scenarios 3 and 4. For the UDR data set, the NEH returned the most instances of the lowest proportional values. As with the SDR data set, NEH dominated scenarios 3 and 4.

As the analysis for the varied GA parameters was conducted on large problems only, the maximum count obtainable for each parameter is 25. The mutation rate of 0.3 resulted in most cases of the minimum makespan values for both data ranges. The mutation rate of 0.7 resulted in the least minimum values for both data ranges. The initial population sizes returned the most minimum makespan values for all problem sizes for test instances experimented on. Reducing the population sizes, thus, seems to degrade the quality of the solution as some of the minimum makespan values achiever earlier were no longer attained.

Table 11: Count of minimum makespan for various mutation rates

Problem size	Scenario 3						Scenario 4					
	Same data range			Unique data range			Same data range			Unique data range		
	0.5 mutrate	0.3 mutrate	0.7 mutrate	0.5 mutrate	0.3 mutrate	0.7 mutrate	0.5 mutrate	0.3 mutrate	0.7 mutrate	0.5 mutrate	0.3 mutrate	0.7 mutrate
12x12x10	1	4	0	0	3	2	1	3	1	0	4	1
15x15x5	2	3	0	2	3	3	2	0	3	2	2	1
20x20x4	2	3	0	4	5	1	3	1	1	3	2	0
25x25x8	3	2	0	2	3	0	3	2	1	3	1	2
30x30x5	2	3	0	1	3	1	1	2	2	0	3	2
TOTAL	**10**	**15**	**0**	**9**	**17**	**7**	**10**	**8**	**8**	**8**	**12**	**6**

Table 12: Count of minimum makespan for various population sizes

Problem size	Scenario 3						Scenario 4					
	Same data range			Unique data range			Same data range			Unique data range		
	Initial population	½ Initial Pop.	¼ Initial Pop.	Initial population	½ Initial Pop.	¼ Initial Pop.	Initial population	½ Initial Pop.	¼ Initial Pop.	Initial population	½ Initial Pop.	¼ Initial Pop.
12x12x10	3	1	1	2	1	3	3	0	2	3	2	1
15x15x5	0	5	0	4	0	2	1	3	1	4	0	1
20x20x4	1	1	3	4	3	3	3	2	0	3	2	0
25x25x8	4	0	1	3	1	1	4	1	1	1	4	0
30x30x5	2	1	2	3	1	1	0	3	2	0	2	3
TOTAL	**10**	**8**	**7**	**16**	**6**	**10**	**11**	**9**	**6**	**11**	**10**	**5**

4.3.3. Statistical test of significance of differences

Statistical testing was performed to determine if the difference in the count of the number of instances in which each of the algorithms found the minimum makespans was significant or not. The null hypothesis, H_0, is that two algorithms being compared have no significant difference in the makespan values at 5 percent level of significance (alpha = 0.05). The alternative hypothesis, H_1, is that the algorithms have a significant difference in makespan values. This is important because the data were randomly generated. The statistical test values achieved are presented in *Table 13* and *Table 15* for the count of the minimum makespan and proportion above the minimum makespan value respectively. The absolute values of the t statistic are of interest in determining the algorithm(s) resulting in the minimum makespan. The evaluation of the significance of the t stat values returned against the t critical value is presented in *Table 14* and *Table 16*. Instances for which there is no significant difference between the two algorithms are denoted by N, and S represents results with a significant difference. Additionally, the algorithm that performed better for a particular problem size or penalty scenario is indicated in brackets for instances with a significant difference.

Count of minimum makespan

The NEH-GA comparison generally returned instances with the lowest t-stat values, however, with a few exceptions. The CDS-NEH comparison returned the lowest t stat values for scenario 2 (large problem sizes) and scenario 4 (medium problem sizes) for the SDR data set, and scenario 1 (medium problem sizes) for the UDR data set. The CDS-GA comparison returned the lowest t stat values for scenario 3 (small problem sizes) and scenario 4 (small problem sizes) for the SDR data set, and scenario 4 (small problem sizes) for the UDR data set. GA was the better performing algorithm for cases where there was a significant difference for the SDR data set. The NEH algorithm returned the majority of instances with the minimum makespan counts for the UDR data set. The difference in makespans was significant for any algorithm compared to CDS, with the CDS algorithm being the worst-performing. It is for this reason that there is no indication of the algorithm which performs better for CDS-NEH and CDS-GA comparisons.

Table 13: t Stat values for the count of minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	-6.2973	-1.3556	-8.0623	-8.5105	-0.2981	-8.1219
	Medium	-6.245	-3.0095	-14.8661	-3.6056	-8.5732	-27.2213
	Large	-6	1	-4	-8.7178	2.753	-3.0551
Scenario 2	Small	-6.958	1.5246	-5.1523	-8.1219	0.4953	-7.3193
	Medium	-5.099	-3.1225	-12.49	-4.8374	-3.4426	-12.49
	Large	-2.1381	-4.5434	-11.225	-5.099	0	-5.099
Scenario 3	Small	-13.5594	4.6577	-3.6056	UNDEF	21.9317	-1.7782
	Medium	-9	2.3333	-4.3333	-12.49	4.4733	-3.3649
	Large	-3.3607	-1.8904	-7.1414	-7.8558	2.4004	-3.0551
Scenario 4	Small	-13.5594	5.1058	-3.1225	-13.5594	5.1058	-3.1225
	Medium	-3.6056	-3.6056	-10.8167	-7.2639	0.9475	-5.369
	Large	-4	-1	-6	-6	1	-4

Table 14: Significance of t Stat values for the count of minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S (GA)	S
	Large	S	N	S	S	S (NEH)	S
Scenario 2	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S (GA)	S
	Large	S	S (GA)	S	S	N	S
Scenario 3	Small	S	S (NEH)	S	N	S (NEH)	N
	Medium	S	S (NEH)	S	S	S (NEH)	S
	Large	S	N	S	S	S (NEH)	S
Scenario 4	Small	S	S (NEH)	S	S	S (NEH)	S
	Medium	S	S (GA)	S	S	N	S
	Large	S	N	S	S	N	S

Proportion beyond minimum makespan

In the test for a proportion of values above the minimum makespan, the same observation was made for all penalties and problem sizes. The NEH-GA comparison resulted in the lowest t-stat values. The paired test of CDS with any method resulted in a higher t-stat value, suggesting that the CDS method returned the most makespan values greater than the minimum for each matrix size. The NEH-GA comparison was also the only one to produce a t-stat value lower than t critical. There was no significant difference between the t-stat value and the t critical value for scenarios 1 and 2 for both data ranges of small problems. Medium problems returned a t-stat value lower than t critical for scenario 4 with both data ranges. Large problems returned lower t-stat values for scenarios 1 and 3 for SDR, and scenarios 2 and 4 for UDR.

Table 15: t Stat values for proportion falling beyond the minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	5.2017	0.9105	5.2271	3.9067	0.3697	4.0086
	Medium	9.3686	3.1116	10.2402	6.774	4.1266	9.8536
	Large	21.2569	-1.7356	20.4309	24.4516	-3.0656	21.3496
Scenario 2	Small	4.9676	0.8628	5.1592	3.5914	-0.4565	3.5985
	Medium	9.1832	3.3348	10.6583	5.5364	3.0074	9.4545
	Large	6.9113	3.4371	6.3809	4.7638	1.3925	6.0476
Scenario 3	Small	15.3478	-3.5605	10.9191	17.5279	-2.9098	3.4483
	Medium	12.9587	-2.2846	12.997	12.4692	-5.3536	12.4092
	Large	3.3186	-1.3395	3.3186	2.9278	-2.2007	2.9278
Scenario 4	Small	11.6613	-5.1845	11.3359	12.7631	-5.3719	11.8116
	Medium	12.2037	-0.7797	8.344	12.2398	-1.4284	12.2818
	Large	2.7558	2.4267	2.7558	2.7092	0.3481	2.7091

Table 16: Significance of t Stat values for proportion falling beyond the minimum makespan

	Problem size	Same data range (SDR)			Unique data range (UDR)		
		CDS - NEH	NEH - GA	CDS - GA	CDS - NEH	NEH - GA	CDS - GA
Scenario 1	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S (GA)	S
	Large	S	N	S	S	S (NEH)	S
Scenario 2	Small	S	N	S	S	N	S
	Medium	S	S (GA)	S	S	S (GA)	S
	Large	S	S (GA)	S	S	N	S
Scenario 3	Small	S	S (NEH)	S	S	S (NEH)	S
	Medium	S	S (NEH)	S	S	S (NEH)	S
	Large	S	N	S	S	S (NEH)	S
Scenario 4	Small	S	S (NEH)	S	S	S (NEH)	S
	Medium	S	S (GA)	S	S	N	S
	Large	S	N	S	S	N	S

The evaluation of the significance of t-stat values for the count of the minimum makespan and proportion beyond minimum value indicated that there are several test instances where the NEH and GA returned similar makespan values. The CDS algorithm, on the other hand, fails to find the minimum makespan values.

This chapter's summary is now presented. The chapter focused on the design and implementation of computational experiments. The design of the experiment involved data generation to suit the problem of interest and the classification of test blocks. The method of execution of the computational experiments was discussed for the various test instances. Results of the experimentation were presented, discussed, and analysed. The next chapter will provide a summation of the findings of this research, and recommendations regarding further study on the topic are proposed.

Chapter 5: Conclusions and recommendations

In this chapter, concluding remarks are made, followed by this study's contributions to literature and recommendations for further research.

Conclusions

The problem of scheduling a reentrant permutation flow shop with deteriorating jobs having the objective of minimising makespan was studied. The model formulation was presented, and CDS, NEH and GA algorithms were utilised to solve the problem. The study involved test problems classified as small, medium and large. Simulations for various deterioration rates were conducted. The GA parameters, mutation rate, and population size were studied to evaluate their influence on the performance of the meta-heuristic in determining the minimum makespan.

The algorithms performed similarly for small problems that are not exposed to deterioration of processing time. The GA and NEH algorithms performed better than the CDS algorithm as the problem sizes increased (i.e. both in the number of jobs and machines). The NEH achieved the lowest makespan within reasonable computation times as problem sizes got bigger, and the complexity of deterioration of processing times increased. Essentially, the NEH algorithm is capable of handling changes that are introduced into the system being scheduled. In instances where the NEH achieved minimum makespan values above the global minimum, the proportional difference was small.

The performance of a GA may be affected by the parameters selected to conduct computational experiments. Two parameters: mutation rate and population size, were used to test this. The lowest mutation rate used, 0.3 resulted in the lowest of makespan values compared to 0.5 and 0.7; however, the computation times remained longer than NEH. Reducing the population size to a quarter of the initial value resulted in the shortest computation times, but also seems to have compromised the quality of the makespan value attainable. The NEH algorithm, thus, appears to be the overall best performing algorithm.

Contributions to literature

Despite the decades-long study of reentrant flowshops and deteriorating jobs, no evidence of work that merges the two factors could be found. A lot of focus on deterioration has been mainly on one and two machine flowshops. This study contributes to the scheduling of a M machine flowshop with deteriorating jobs. It is unlikely that a production facility can have only one production step. Thus the solution developed can be adapted in various production processes with deteriorating jobs. Schedulers and production planners alike can use the proposed scheduling solution to improve the efficiency of their operations. By utilising the proposed method, production schedules can be set up in a manner which ensures jobs in processes such as tube rolling, and ingot forging are prepared just in time for them to be processed, and not earlier. Using the NEH would be beneficial in providing a schedule timeously, and without having to determine supplementary information as compared to the GA algorithm.

Recommendations

For future studies, the NEH algorithm can be compared with other solution methods such as Palmer’s, branch and bound and meta-heuristics like Simulated Annealing (SA). These methods are known to have yielded satisfactory results in previous studies of flowshop scheduling. The problem sizes can also be increased and be evaluated with the same solution methods as were proposed for this work. Recent work in flowshop scheduling considers the evaluation of multiple objectives. In light of supply chain improvement, the due date is proposed as an additional objective to study.

References

1. Al-harkan, Ibrahim M. (n.d.) Algorithms for sequencing and scheduling. https://docplayer.net/14716133-Algorithms-for-sequencing-and-scheduling-ibrahim-m-alharkan.html. Accessed 3rd July, 2020.
2. Bank M., Fatemi Ghomi S.M.T., Jolai F., Behnamian J., 2012. Two-machine flow shop total tardiness scheduling problem with deteriorating jobs. Applied Mathematical Modelling, 36, 5418–5426.
3. Baskar A., 2016. Revisiting the NEH algorithm- the power of job insertion technique for optimizing the makespan in permutation flow shop scheduling. International Journal of Industrial Engineering Computations, 7, 353–366.
4. Browne S. and Yechiali U., 1990. Scheduling deteriorating jobs on a single processor. Operations Research, 38(3), 495-498.
5. Campbell H.G., Dudek R.A. and Smith M.L., 1970. A heuristic algorithm for the *n* job, *m* machine sequencing problem. Management Science, 6(10), B-631 – B-637.
6. Chen J-S., 2006. A branch and bound procedure for the reentrant permutation flow-shop scheduling problem. Int J Adv Manuf Technol, 29, 1186–1193.
7. Chen J-S., Pan J.C-H. and Wu C-K., 2007. Minimizing makespan in reentrant flow-shops using hybrid tabu search. Int J Adv Manuf Technol, 34, 353-361.
8. Chen J-S., Pan J.C-H. and Lin C-M., 2008. A hybrid genetic algorithm for the reentrant flow-shop scheduling problem, Expert Systems with Applications, 34, 570–577.
9. Choi H-S. and Lee D-H., 2007. A Branch and Bound Algorithm for Two-Stage Hybrid Flow Shop Scheduling: Minimizing the Number of Tardy Jobs. Journal of the Korean Institute of Industrial Engineers, 33(2), 213-220.
10. Choi H-S, Kim H-W, Lee D-H, Yoon J., Yun C.Y. and Chae K.B., 2009. Scheduling algorithms for two-stage reentrant hybrid flow shops: minimizing makespan under the maximum allowable due dates. Int J Adv Manuf Technol, 42, 963–973.

11. Choi S-W. and Kim Y-D., 2007. Minimizing makespan on a two-machine reentrant flow shop. Journal of the Operational Research Society, 58, 972-981.

12. Choudhari S.D. and Khanna R., 2017. Flow Shop Scheduling Problem with Four Machines N-Job for Minimizing Makespan. International Journal of Engineering Research & Technology, 6(2), 532-536.

13. Eren T. and Güner E., 2007. Minimizing total tardiness in a scheduling problem with a learning effect. Applied Mathematical Modelling, 31, 1351–1361.

14. Graham R. L., Lawler E. L., Lenstra J. K. and RinnooyKan A. H. G., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey, Annals of Discrete Mathematics, 5, 287–326.

15. Graves S.C., Meal H.C., Stefek D. and Zeghmi A.H., 1983. Scheduling of reentrant flow shops. Massachusetts Institute of Technology.

16. Goldberg D.E., 1989. Genetic Algorithms in search, optimization, and machine learning. The University of Alabama.

17. He F., Dong M. and Shao X., 2011. Modelling and analysis of material flows in reentrant supply chain networks using modified partial differential equations. Journal of applied mathematics, 2011, 1-14.

18. Hindi K.S. and Mhlanga S., 2001. Scheduling deteriorating jobs on parallel machines: a simulated annealing approach. Production Planning & Control, 12(1), 76-80.

19. Hsu C-J., Kuo W-H., Yang D-L. and Chern M-S., 2006. Minimizing the makespan in a two-stage flowshop scheduling problem with a function on alternative machines. Journal of Marine Science and Technology, 14(4), 213-217.

20. Iyer S.K. and Saxena B., 2004. Improved genetic algorithm for the permutation flowshop scheduling problem. Computers & Operations Research, 31, 593–606.

21. Johnson S.M., 1953. Optimal two- and three-stage production schedules with setup times included. The Rand Corporation.

22. Jafari A-A., Khademi-Zare H, Lotfi M.M. and Tavakkoli-Moghaddam R., 2017. A note on "On three-machine flow shop scheduling with deteriorating jobs". International Journal of Production Economics, 191, 250–252.

23. Ji M., Cheng T.C.E., 2010. Scheduling resumable simple linear deteriorating jobs on a single machine with an availability constraint to minimise makespan. Computers & Industrial Engineering, 59, 794–798.

24. Kim H-W. and Lee D-H., 2008. Heuristic algorithms for reentrant hybrid flow shop scheduling with unrelated parallel machines. IMechE: J. Engineering Manufacture, 223(Part B), 433-442.

25. Kononov A. and Gawiejnowicz S., 2001. NP-hard cases in scheduling deteriorating jobs on dedicated machines. Journal of the Operational Research Society, 52, 708-718

26. Kubiak W. and van de Velde S., 1998. Scheduling Deteriorating Jobs To Minimise Makespan. Naval Research Logistics, 45, 511-523.

27. Mazdeh M.M., Zaerpour F., Zareei A., Hajinezhad A., 2010. Parallel machines scheduling to minimise job tardiness and machine deteriorating cost with deteriorating jobs. Applied Mathematical Modelling, 34, 1498–1510.

28. Mosheiov G., 1991. V-shaped policies for scheduling deteriorating jobs. Operations Research. 39(6), 979-991.

29. Moursli O., 1999. The hybrid flowshop scheduling problem. PhD thesis: Université Catholique de Louvain, ESPO, Belgium.

30. Murata T., Ishibuchi H., Tanaka H., 1996. Genetic algorithms for flowshop scheduling problems. Computers and Industrial Engineering, 30(4), 1061-1071.

31. Ng C.T., Wang J.-B., Cheng T.C.E. and Liu L.L., 2010. A branch-and-bound algorithm for solving a two-machine flow shop problem with deteriorating jobs. Computers &Operations Research, 37, 83-90.

32. Pan JC-H. and Chen J-S., 2003. Minimizing makespan in reentrant permutation flow shops. Journal of the Operational Research Society, 54, 642-653.

33. Pinedo M., 2016. Scheduling: theory, algorithms and systems. 5th Edition. New York: Springer.

34. Reeves C.R., 1995. A genetic algorithm for flowshop sequencing. Computers Ops Res., 22(1), 5-13.

35. Rifai A.P., Nguyen H-T., Dawal S.Z.M., 2016. Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. Applied Soft Computing, 40, 42–57.

36. Rios-Mercado R.Z. and Bard J.F., 1999. A branch and bound algorithm for permutation flow shops with sequence dependent setup times. IIE Transactions, 31, 721-731.

37. Ruiz R. and Maroto C., 2005. A comprehensive review and evaluation of permutation flowshop heuristics to minimise flowtime. European Journal of Operational Research, DOI: 10.1016/j.ejor.2004.04.017

38. Simchi-Levi D., Kaminsky P., Simchi-Levi E, 2008. Designing and managing the supply chain: Concepts, Strategies and Case Studies. 3rd Edition. New York: McGraw-Hill.

39. Shiau Y-R., Lee W-C, Wu C-C and Chang C-M., 2007. Two-machine flowshop scheduling to minimise mean flow time under simple linear deterioration. Int J Adv Manuf Technol, 34, 774–782.

40. Stevenson W.J., 2002. Operations Management. 7th Edition. New York: McGraw-Hill Irwin.

41. TasoujiHassanpour S., Amin-Naseri M. R. and Nahavandi N., 2015. Solving Reentrant No-wait Flow shop Scheduling Problem. International Journal of Engineering (IJE), TRANSACTIONS C: Aspects, 28(6), 903-912.

42. Tyagi N., Varshney R.G. and Chandramouli A.B., 2013. Six Decades of Flowshop Scheduling Research. International Journal of Scientific & Engineering Research, 4(9), 854-864.

43. Wang H, Huang M. and Wang J., 2019. An effective metaheuristic algorithm for flowshop scheduling with deteriorating jobs. Journal of Intelligent Manufacturing, 30, 2733–2742.

44. Wang J-B., Ng C.T.D, Cheng T.C.E., Liu L-L., 2006. Minimizing total completion time in a two-machine flow shop with deteriorating jobs. Applied Mathematics and Computation, 180, 185–193.

45. Wang J-B., Wang J-J. and Ji P., 2011. Scheduling jobs with chain precedence constraints and deteriorating jobs. Journal of the Operational Research Society, 62, 1765–1770.

46. Wang J-B and Xia Z-Q, 2005. Scheduling jobs under decreasing linear deterioration. Information Processing Letters, 94, 63–69.

47. Zhang X.Y. and Chen L., 2017. A reentrant hybrid flow shop scheduling problem with machine eligibility constraints. International Journal of Production Research, DOI: 10.1080/00207543.2017.1408971

Annexure A: Minimum makespan graphs

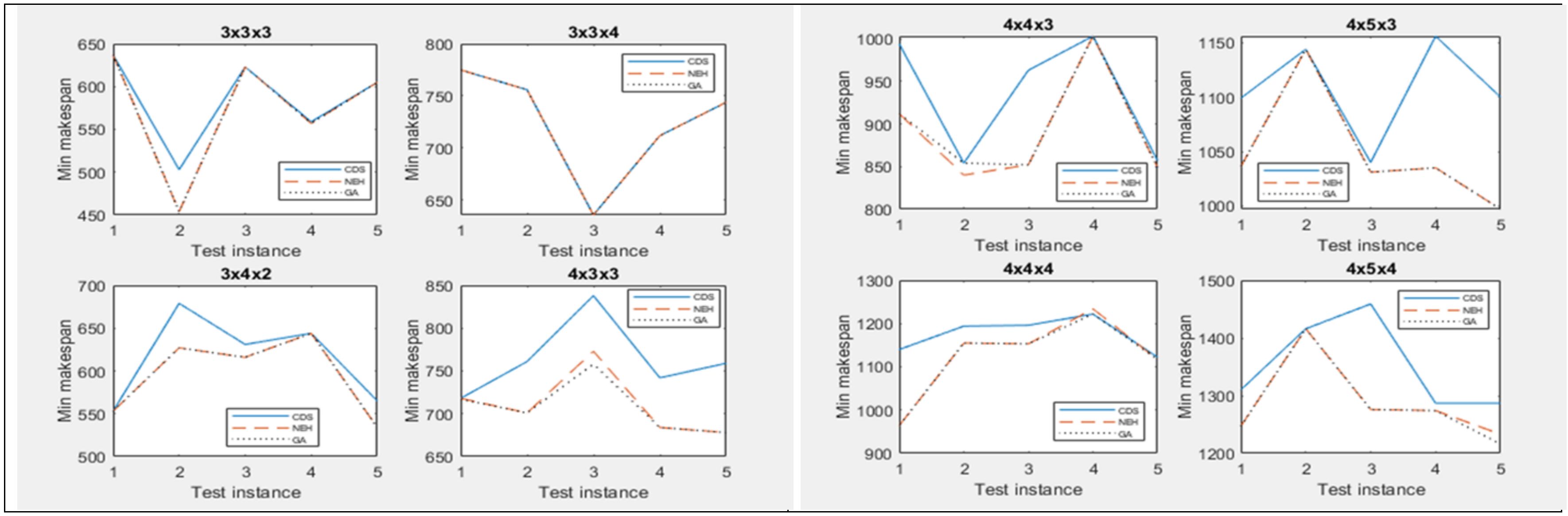


Figure 16: Minimum makespan plots for small problems and same data range with scenario 1

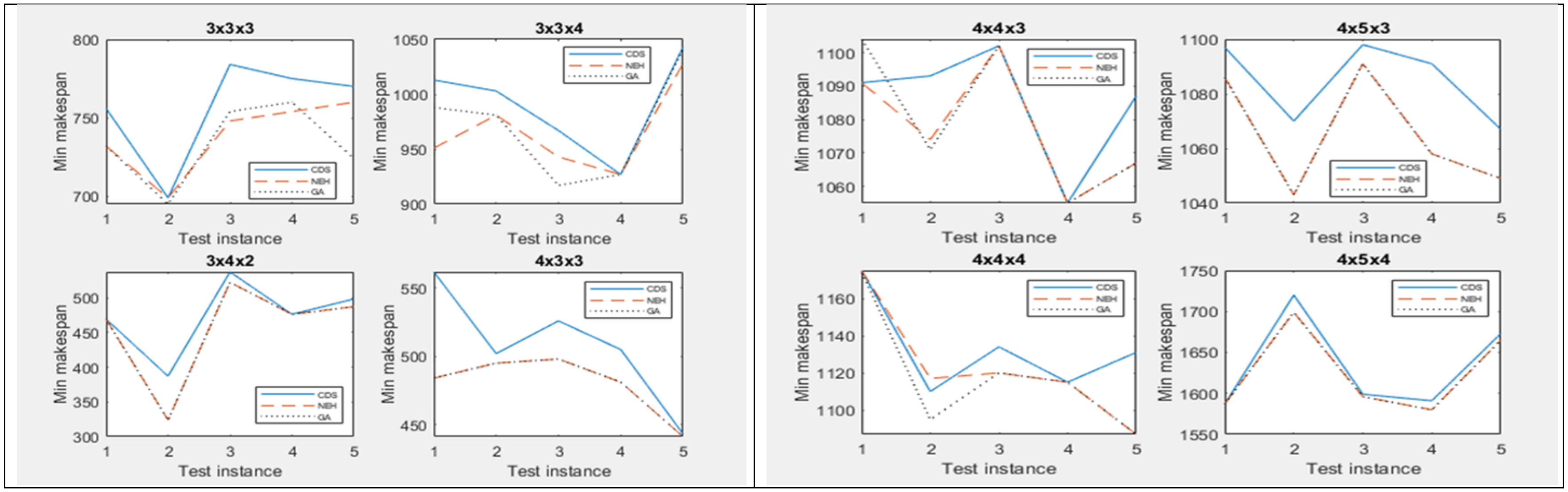


Figure 17: Minimum makespan plots for small problems and unique data range with scenario 1

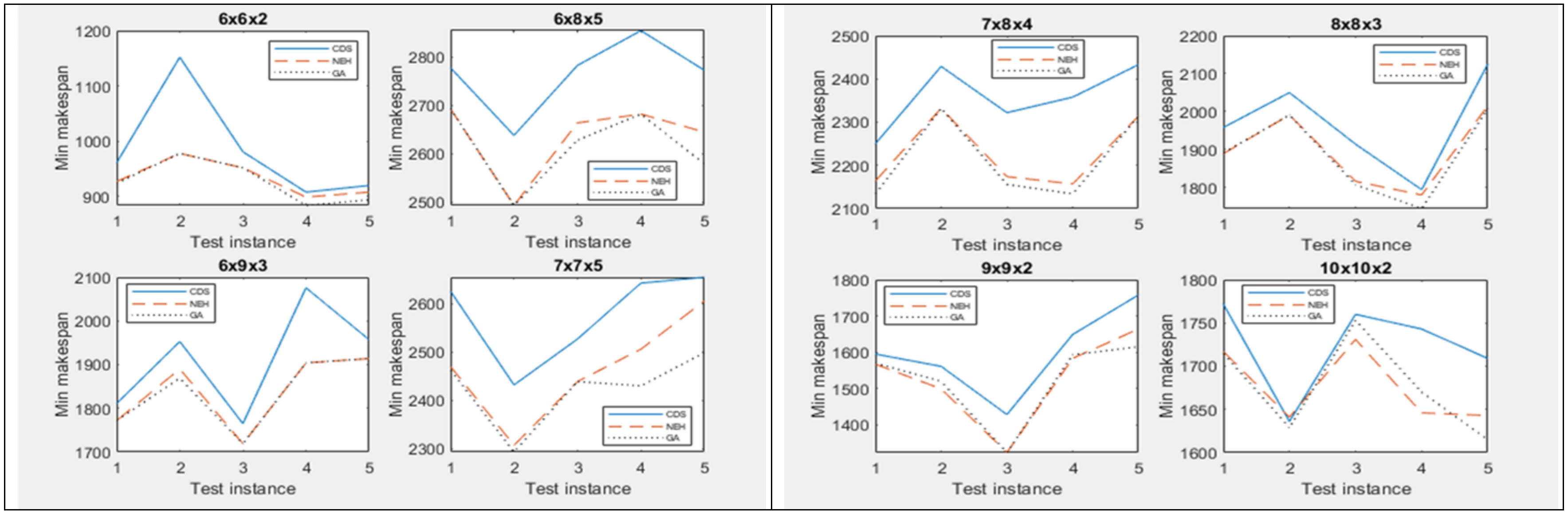


Figure 18: Minimum makespan for medium problems and same data range with scenario 1

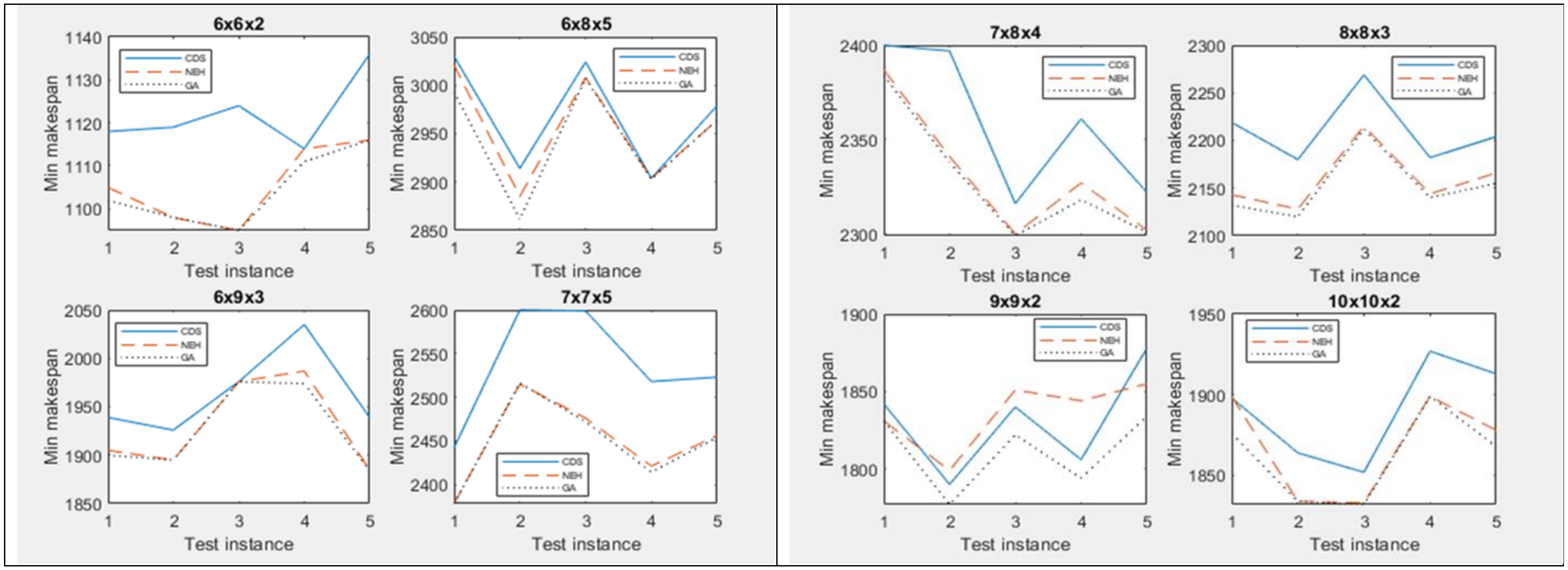


Figure 19: Minimum makespan plots for medium problems and unique data range with scenario 1

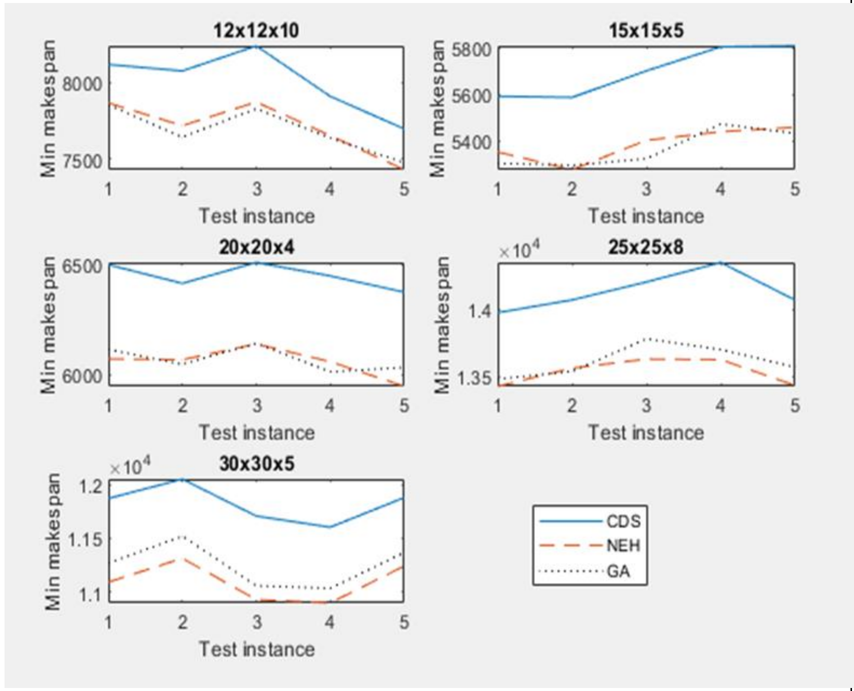
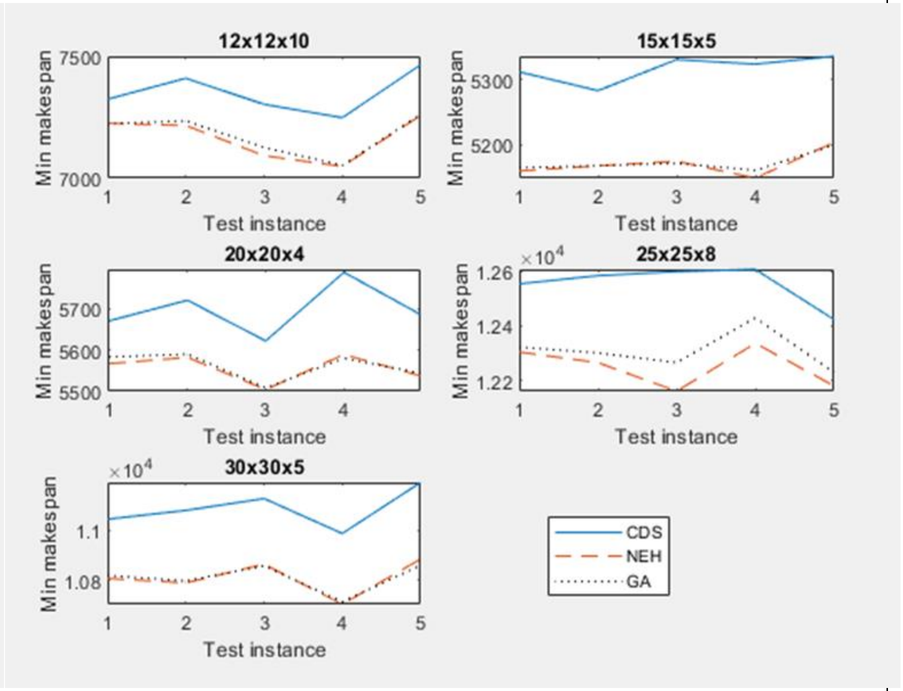
Same data range	Unique data range
	

Figure 20: Minimum makespan plots for large problems with scenario 1

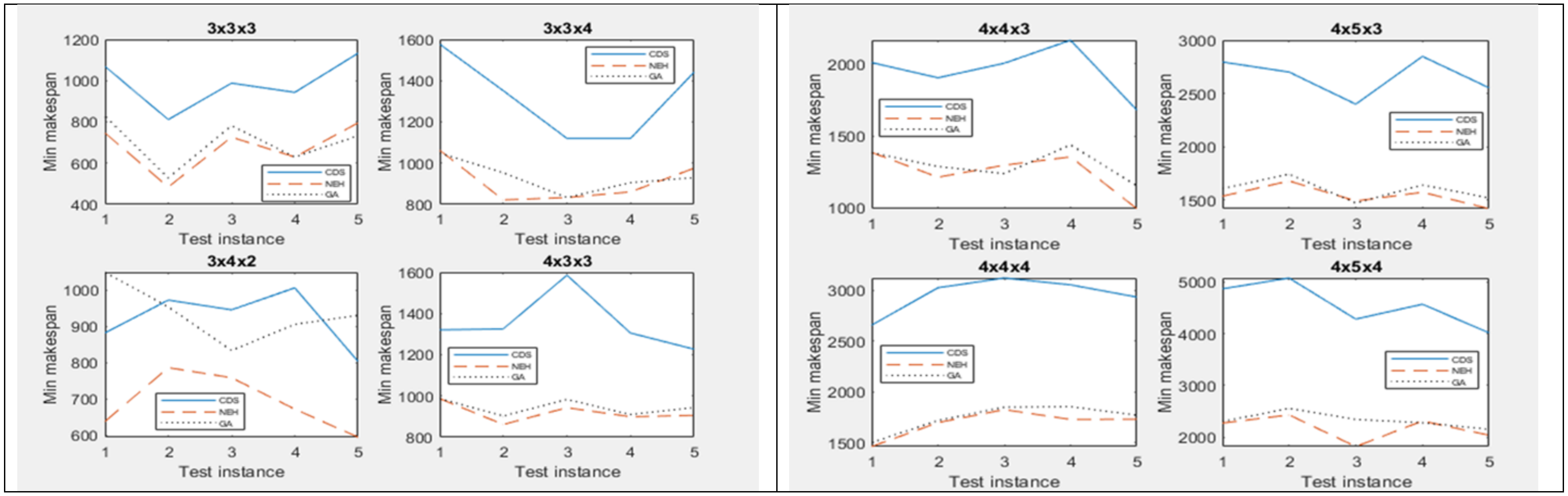


Figure 21: Minimum makespan plots for small problems and same data range with scenario 3

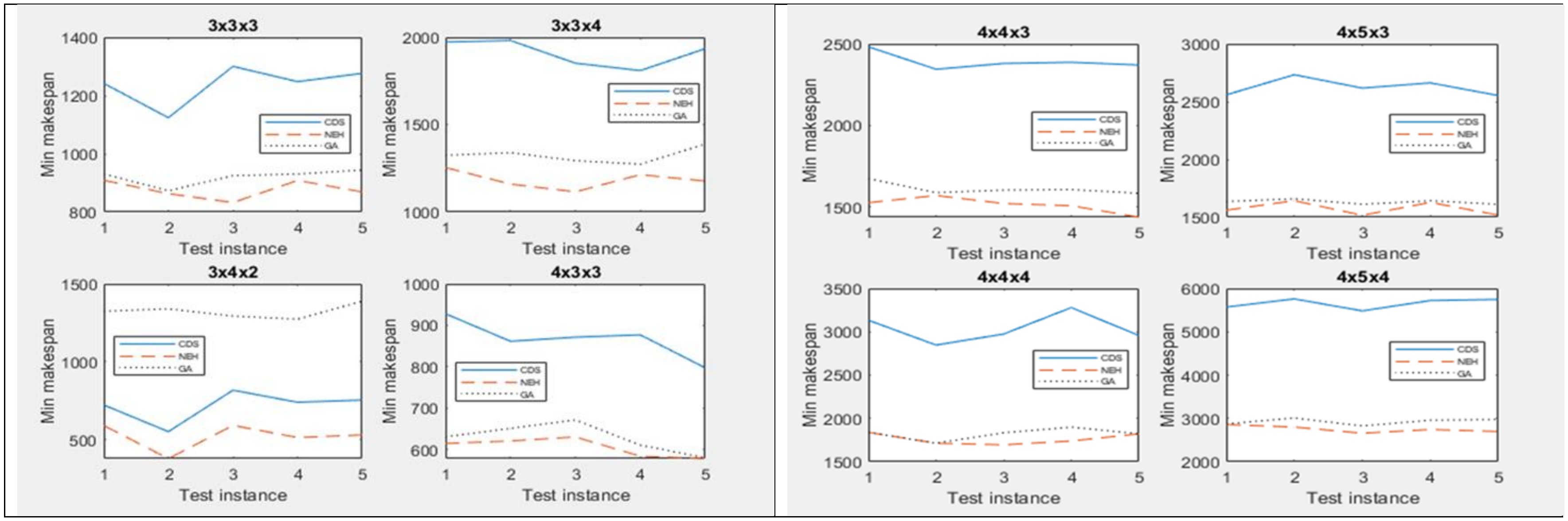


Figure 22: Minimum makespan plots for small problems and unique data range with scenario 3

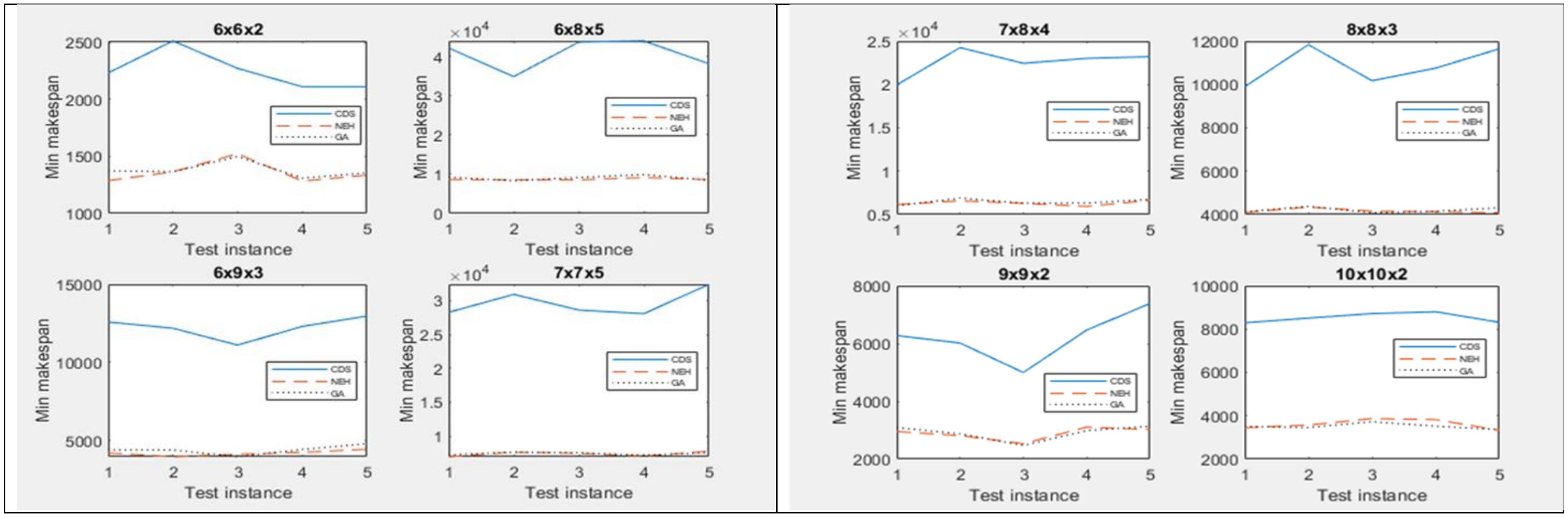


Figure 23: Minimum makespan plots for medium problems and same data range with scenario 3

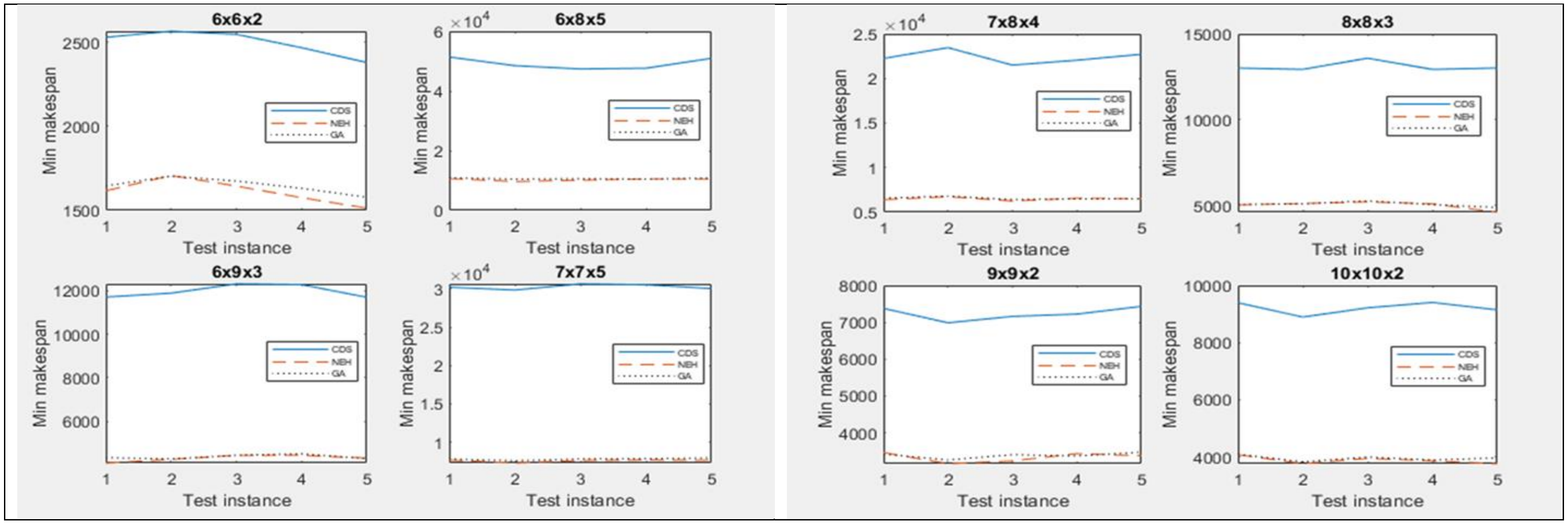


Figure 24: Minimum makespan plots for medium problems and unique data range with scenario 3

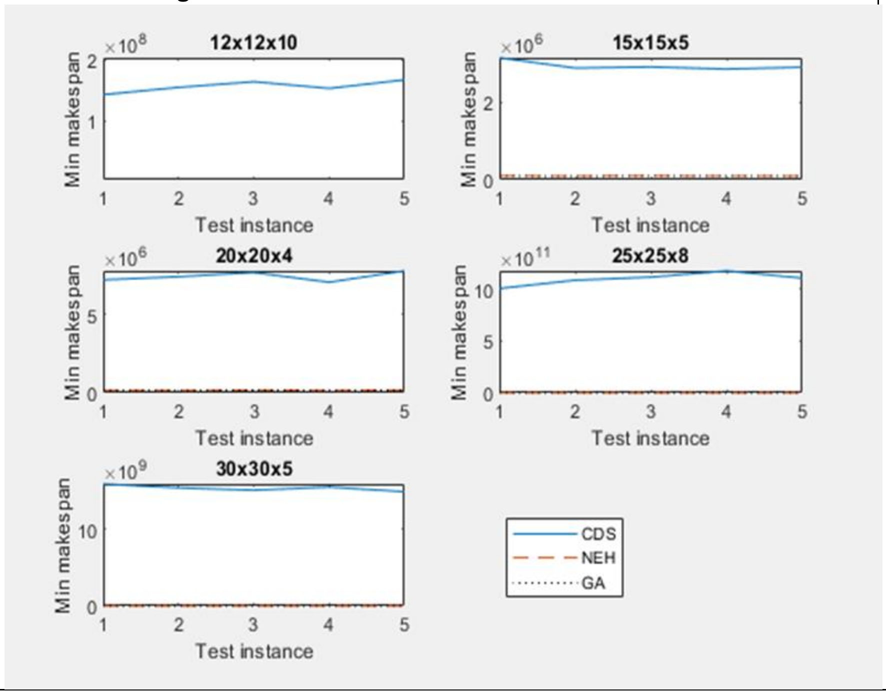
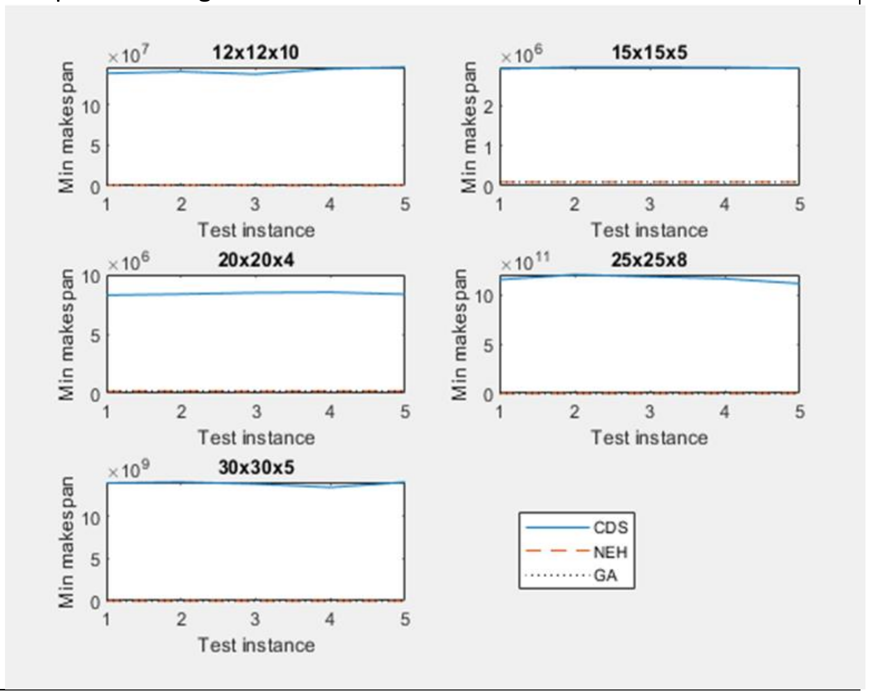
Same data range	Unique data range
	

Figure 25: Minimum makespan plots for large problems with scenario 3

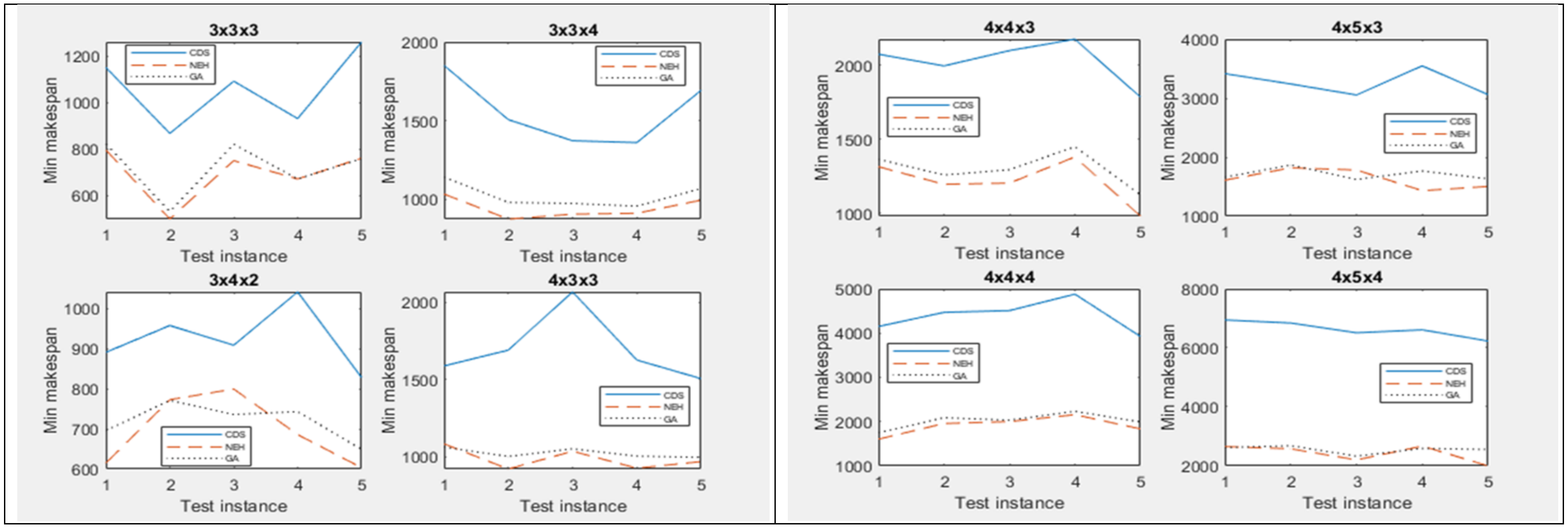


Figure 26: Minimum makespan plots for small problems and same data range with scenario 4

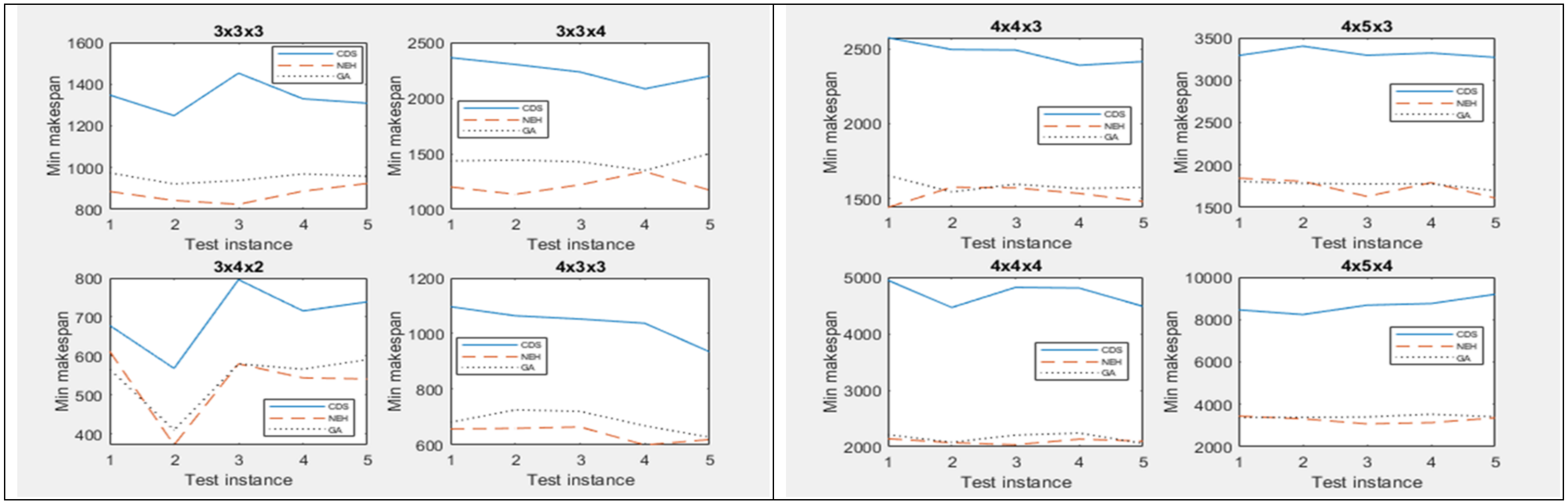


Figure 27: Minimum makespan plots for small problems and unique data range with scenario 4

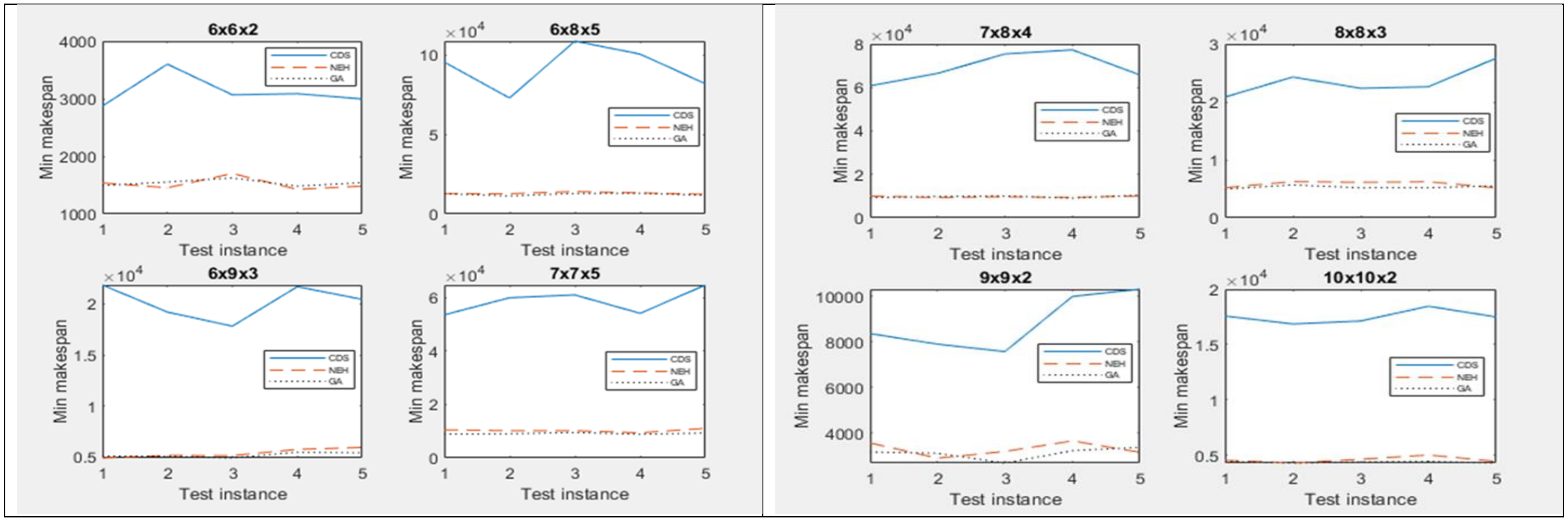


Figure 28: Minimum makespan plots for medium problems and same data range with scenario 4

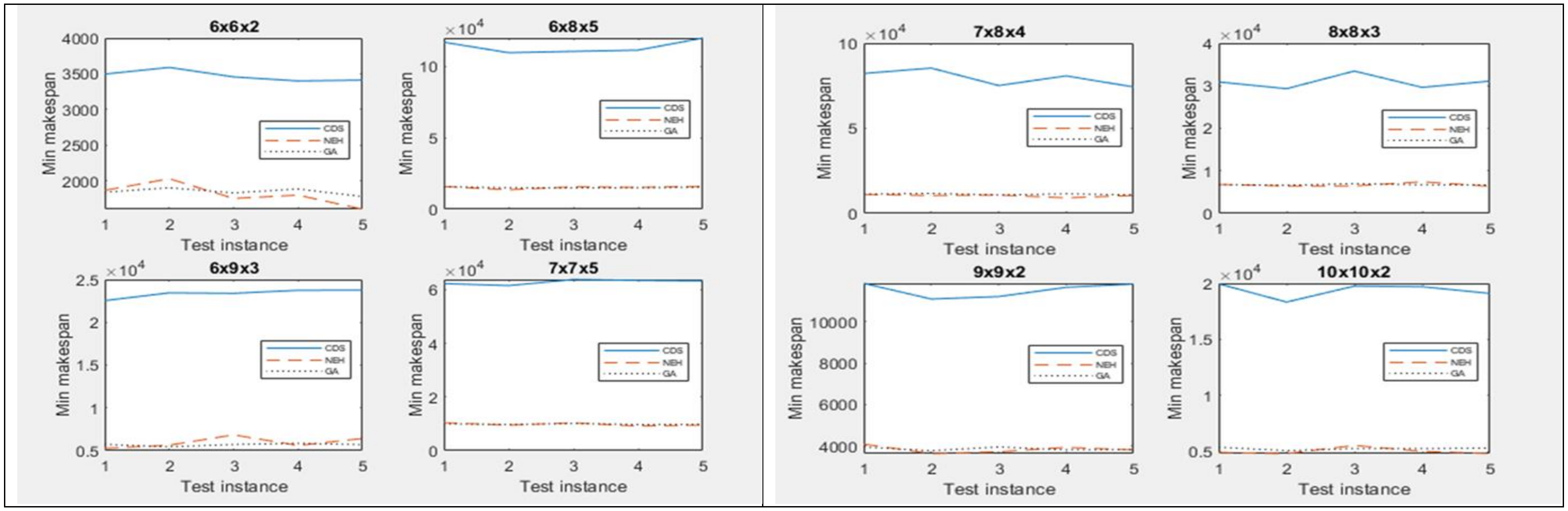


Figure 29: Minimum makespan plots for medium problems and unique data range with scenario 4

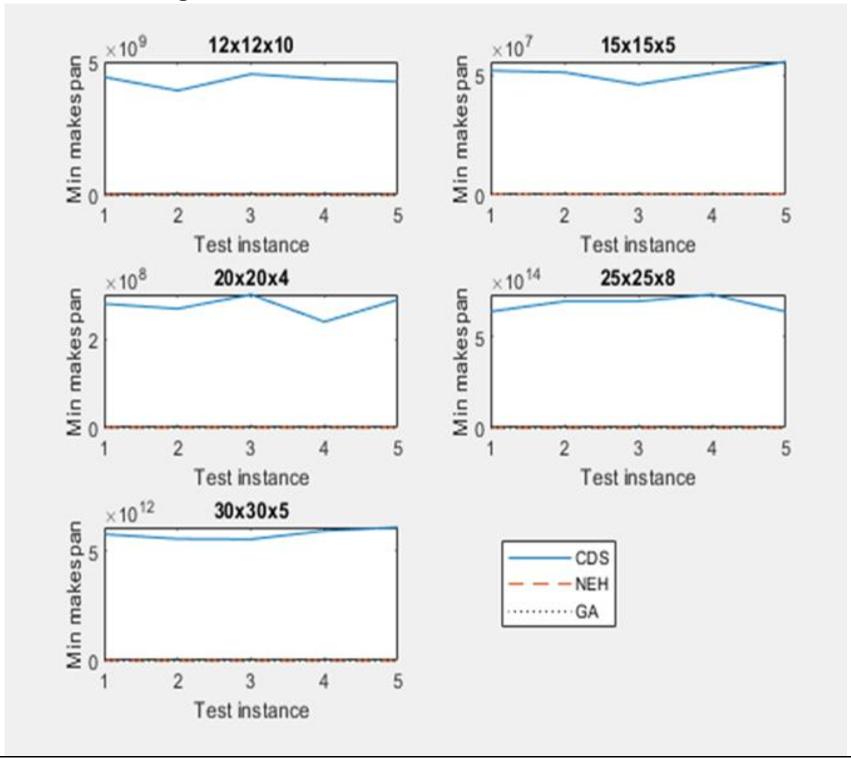
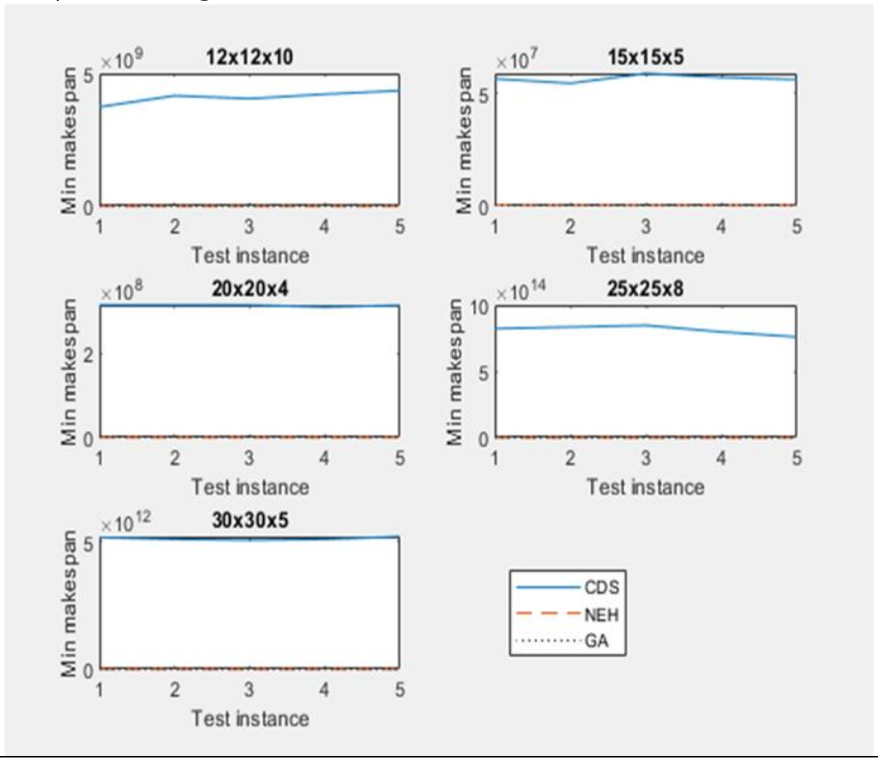
Same data range	Unique data range
	

Figure 30: Minimum makespan plots for large problems with scenario 4

Annexure B: Makespan analysis

Table 17: Count of minimum makespan for small problems with SDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
3x3x3	3	5	5	1	5	3	0	4	2	0	4	2
3x3x4	5	5	5	4	5	4	0	3	3	0	5	0
3x4x2	2	5	5	2	5	5	0	5	0	0	3	2
4x3x3	0	3	5	1	2	4	0	5	1	0	4	1
4x4x3	1	5	4	0	5	2	0	3	2	0	5	0
4x5x3	1	5	5	1	5	3	0	4	1	0	4	1
4x4x4	1	4	5	1	4	4	0	5	0	0	5	0
4x5x4	1	4	5	1	4	4	0	4	1	0	3	2
TOTAL	**14**	**36**	**39**	**11**	**35**	**29**	**0**	**33**	**10**	**0**	**33**	**8**

Table 18: Count of minimum makespan for small problems with UDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
3x3x3	0	3	3	0	4	4	0	5	0	0	5	0
3x3x4	1	4	3	1	4	5	0	5	0	0	5	0
3x4x2	2	5	5	1	4	5	0	5	0	0	4	2
4x3x3	0	5	5	0	5	3	0	5	0	0	5	0
4x4x3	3	4	4	2	3	2	0	5	0	0	4	1
4x5x3	0	5	5	0	5	4	0	5	0	0	2	3
4x4x4	1	3	5	1	4	4	0	5	3	0	4	1
4x5x4	1	5	5	0	3	3	0	5	0	0	4	1
TOTAL	**8**	**34**	**35**	**5**	**32**	**30**	**0**	**40**	**3**	**0**	**33**	**8**

Table 19: Count of minimum makespan for medium problems with SDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
6x6x2	0	2	5	0	2	5	0	4	1	0	3	2
6x8x5	0	3	5	0	2	4	0	3	2	0	0	5
6x9x3	0	4	5	0	4	3	0	4	1	0	1	4
7x7x5	0	1	5	0	2	4	0	3	2	0	0	5
7x8x4	0	2	5	0	2	3	0	4	1	0	3	2
8x8x3	0	2	4	0	1	5	0	4	1	0	1	4
9x9x2	0	4	2	0	3	3	0	3	2	0	2	3
10x10x2	0	2	3	0	0	5	0	2	3	0	0	5
TOTAL	**0**	**20**	**34**	**0**	**16**	**32**	**0**	**27**	**13**	**0**	**10**	**30**

Table 20: Count of minimum makespan for medium problems with UDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
6x6x2	0	3	5	0	3	3	0	4	1	0	3	2
6x8x5	0	1	5	0	1	5	0	4	1	0	1	4
6x9x3	1	2	5	1	3	5	0	4	2	0	2	3
7x7x5	0	2	4	0	3	2	0	5	0	0	3	2
7x8x4	0	0	5	0	2	5	0	4	1	0	4	1
8x8x3	0	0	5	0	1	4	0	3	2	0	3	2
9x9x2	0	1	5	0	1	5	0	3	2	0	3	2
10x10x2	0	2	5	0	2	4	0	5	0	0	4	1
TOTAL	**1**	**11**	**39**	**1**	**16**	**33**	**0**	**32**	**9**	**0**	**23**	**17**

Table 21: Count of minimum makespan for large problems with SDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
12x12x10	0	1	4	0	2	3	0	3	2	0	2	3
15x15x5	0	2	3	0	2	3	0	1	4	0	2	3
20x20x4	0	3	2	0	0	5	0	0	5	0	4	1
25x25x8	0	4	1	0	0	5	0	0	5	0	0	5
30x30x5	0	5	0	0	0	5	0	4	1	0	2	3
TOTAL	**0**	**15**	**10**	**0**	**4**	**21**	**0**	**8**	**17**	**0**	**10**	**15**

Table 22: Count of minimum makespan for large problems with UDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
12x12x10	0	4	1	0	2	4	0	3	2	0	4	1
15x15x5	0	3	3	0	3	2	0	3	2	0	4	1
20x20x4	0	4	1	0	5	0	0	5	0	0	5	0
25x25x8	0	5	0	0	3	2	0	5	0	0	0	5
30x30x5	0	3	2	0	0	5	0	2	3	0	2	3
TOTAL	**0**	**19**	**7**	**0**	**13**	**13**	**0**	**18**	**7**	**0**	**15**	**10**

Table 23: Average proportion above minimum makespan for small problems with SDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
3x3x3	0.289	0.267	0.267	0.295	0.267	0.269	1.042	0.396	0.446	1.124	0.394	0.445
3x3x4	0.139	0.139	0.139	0.140	0.139	0.140	0.609	0.108	0.137	0.777	0.079	0.171
3x4x2	0.151	0.114	0.114	0.151	0.116	0.116	0.552	0.161	0.573	0.533	0.152	0.192
4x3x3	0.126	0.048	0.044	0.127	0.075	0.049	0.570	0.067	0.097	0.837	0.072	0.111
4x4x3	0.112	0.061	0.064	0.115	0.071	0.078	0.964	0.256	0.308	1.064	0.238	0.323
4x5x3	0.111	0.052	0.052	0.112	0.056	0.059	0.874	0.085	0.125	1.280	0.138	0.194
4x4x4	0.219	0.167	0.164	0.220	0.168	0.164	1.027	0.156	0.190	1.737	0.192	0.259
4x5x4	0.112	0.060	0.057	0.114	0.062	0.060	1.509	0.196	0.279	2.321	0.211	0.279
Average	**0.157**	**0.114**	**0.113**	**0.159**	**0.119**	**0.117**	**0.893**	**0.178**	**0.269**	**1.209**	**0.184**	**0.247**

Table 24: Average proportion above minimum makespan for small problems with UDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
3x3x3	0.089	0.063	0.055	0.089	0.063	0.062	0.487	0.053	0.106	0.623	0.059	0.156
3x3x4	0.080	0.053	0.059	0.080	0.053	0.048	0.712	0.061	0.187	0.970	0.069	0.260
3x4x2	0.461	0.406	0.406	0.449	0.399	0.397	0.893	0.375	2.491	0.877	0.423	0.458
4x3x3	0.151	0.088	0.088	0.151	0.087	0.093	0.500	0.047	0.088	0.729	0.067	0.142
4x4x3	0.029	0.022	0.024	0.084	0.089	0.098	0.665	0.052	0.121	0.718	0.055	0.102
4x5x3	0.040	0.021	0.021	0.033	0.016	0.017	0.731	0.037	0.076	1.060	0.079	0.099
4x4x4	0.042	0.033	0.029	0.042	0.037	0.036	0.792	0.039	0.075	1.310	0.030	0.061
4x5x4	0.034	0.029	0.029	0.049	0.029	0.028	1.128	0.035	0.102	1.813	0.061	0.112
Average	**0.116**	**0.089**	**0.089**	**0.122**	**0.097**	**0.097**	**0.738**	**0.088**	**0.406**	**1.013**	**0.105**	**0.174**

Table 25: Average proportion above minimum makespan for medium problems with SDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
6x6x2	0.114	0.055	0.048	0.119	0.057	0.051	0.751	0.059	0.075	1.191	0.067	2.041
6x8x5	0.109	0.057	0.049	0.108	0.060	0.052	3.854	0.043	0.079	7.130	0.154	0.093
6x9x3	0.113	0.070	0.068	0.103	0.060	0.061	2.088	0.059	0.110	3.119	0.095	0.057
7x7x5	0.124	0.075	0.057	0.107	0.060	0.045	3.281	0.061	0.068	5.630	0.159	0.031
7x8x4	0.105	0.044	0.037	0.110	0.052	0.054	2.797	0.063	0.086	6.620	0.068	0.071
8x8x3	0.129	0.088	0.083	0.125	0.079	0.073	1.663	0.021	0.031	3.744	0.167	0.067
9x9x2	0.208	0.154	0.152	0.211	0.162	0.154	1.520	0.169	0.179	2.296	0.227	0.160
10x10x2	0.067	0.038	0.038	0.105	0.074	0.052	1.557	0.083	0.054	3.046	0.067	0.018
Average	**0.121**	**0.073**	**0.067**	**0.124**	**0.076**	**0.068**	**2.189**	**0.070**	**0.085**	**4.097**	**0.125**	**0.317**

Table 26: Average proportion above minimum makespan for medium problems with UDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
6x6x2	0.025	0.010	0.009	0.030	0.015	0.013	0.653	0.065	0.088	1.162	0.129	0.153
6x8x5	0.038	0.033	0.029	0.038	0.033	0.029	4.099	0.069	0.103	7.361	0.123	0.120
6x9x3	0.042	0.024	0.022	0.043	0.024	0.022	1.930	0.053	0.069	3.437	0.130	0.080
7x7x5	0.067	0.030	0.029	0.068	0.037	0.037	3.154	0.040	0.070	5.849	0.068	0.078
7x8x4	0.026	0.014	0.013	0.026	0.014	0.013	2.585	0.039	0.049	7.711	0.141	0.231
8x8x3	0.043	0.018	0.015	0.034	0.026	0.017	1.838	0.087	0.100	3.844	0.051	0.060
9x9x2	0.031	0.033	0.019	0.034	0.037	0.022	1.288	0.055	0.071	2.171	0.056	0.063
10x10x2	0.032	0.020	0.016	0.052	0.039	0.038	1.447	0.032	0.051	3.027	0.043	0.092
Average	**0.038**	**0.023**	**0.019**	**0.041**	**0.028**	**0.024**	**2.124**	**0.055**	**0.075**	**4.320**	**0.093**	**0.109**

Table 27: Average proportion above minimum makespan for large problems with SDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
12x12x10	0.078	0.037	0.035	0.077	0.035	0.032	235	0.044	0.027	1501	0.149	0.093
15x15x5	0.080	0.021	0.017	0.078	0.025	0.028	33	0.100	0.065	180	0.133	0.084
20x20x4	0.085	0.019	0.021	0.090	0.054	0.027	52	0.104	0.094	383	0.136	0.204
25x25x8	0.053	0.008	0.014	0.057	0.040	0.019	16257	0.204	0.085	452975	0.338	0.060
30x30x5	0.085	0.018	0.032	0.244	0.137	0.048	14746	4.692	7.752	43578	0.219	0.057
Average	**0.076**	**0.021**	**0.024**	**0.109**	**0.058**	**0.031**	**6264**	**1.029**	**1.605**	**99723**	**0.195**	**0.100**

Table 28: Average proportion above minimum makespan for large problems with UDR data set

Problem size	Scenario 1			Scenario 2			Scenario 3			Scenario 4		
	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA	CDS	NEH	GA
12x12x10	0.043	0.017	0.019	0.044	0.018	0.016	227	0.029	0.035	1539	0.120	0.200
15x15x5	0.033	0.004	0.005	0.028	0.005	0.007	32	0.024	0.036	195	0.145	0.182
20x20x4	0.035	0.009	0.010	0.055	0.023	0.038	51	0.030	0.067	502	0.157	0.333
25x25x8	0.032	0.007	0.012	0.107	0.038	0.044	14156	0.095	0.148	457061	0.303	0.016
30x30x5	0.036	0.010	0.010	0.149	0.124	0.042	1855	0.073	0.058	41973	0.259	0.177
Average	**0.036**	**0.010**	**0.011**	**0.077**	**0.042**	**0.029**	**3264**	**0.050**	**0.069**	**100254**	**0.197**	**0.182**

Annexure C: Computation times

Table 29: Average computation times for small problems with scenario 1

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
3x3x3	0.0044	0.0003	0.0130	0.0004	0.0281	0.0298
3x3x4	0.0393	0.0080	0.0153	0.0003	0.0269	0.0252
3x4x2	0.0003	0.0003	0.0003	0.0002	0.0257	0.0269
4x3x3	0.0003	0.0003	0.0003	0.0004	0.0276	0.0292
4x4x3	0.0290	0.0004	0.0004	0.0004	0.2008	0.0426
4x5x3	0.0162	0.0005	0.0005	0.0004	0.0405	0.0361
4x4x4	0.0003	0.0004	0.0005	0.0004	0.0336	0.0312
4x5x4	0.0005	0.0005	0.0005	0.0005	0.0423	0.0283

Table 30: Average computation times for medium problems with scenario 1

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
6x6x2	0.0584	0.0005	0.0007	0.0006	0.0993	0.1330
6x8x5	0.0010	0.0011	0.0042	0.0017	0.2586	0.2425
6x9x3	0.0034	0.0009	0.0011	0.0075	0.2919	0.2625
7x7x5	0.0010	0.0011	0.0044	0.0017	0.2724	0.2295
7x8x4	0.1402	0.0319	0.0038	0.0014	0.4973	0.3452
8x8x3	0.0013	0.0009	0.0017	0.0016	0.3150	0.2888
9x9x2	0.0009	0.0008	0.0016	0.0024	0.4771	0.4534
10x10x2	0.0667	0.0279	0.0026	0.0026	0.8568	0.8794

Table 31: Average computation times for large problems with scenario 1

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
12x12x10	0.0047	0.1017	0.5409	0.0157	38.8601	38.8154
15x15x5	0.0044	0.0043	0.0202	0.0186	39.5828	38.2834
20x20x4	0.0072	0.0079	0.0402	0.0470	178.4229	173.2750
25x25x8	0.0836	0.0272	0.2105	0.2008	609.1398	541.6924
30x30x5	0.0274	0.0252	0.3879	0.3440	865.3643	872.3942

Table 32: Average computation times for small problems with scenario 2

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
3x3x3	0.0460	0.0003	0.0268	0.0003	0.2414	0.0286
3x3x4	0.0197	0.0003	0.0138	0.0003	0.0277	0.0362
3x4x2	0.0890	0.0003	0.0282	0.0003	0.0263	0.0279
4x3x3	0.0003	0.0003	0.0005	0.0004	0.1929	0.3108
4x4x3	0.0005	0.0004	0.0005	0.0006	0.0281	0.0251
4x5x3	0.0544	0.0004	0.0005	0.0005	0.0466	0.0368
4x4x4	0.0004	0.0005	0.0005	0.0005	0.0621	0.0332
4x5x4	0.0004	0.0005	0.0005	0.0005	0.0371	0.0389

Table 33: Average computation times for medium problems with scenario 2

Matrix size	**CDS**		**NEH**		**GA**	
	SD	UD	SD	UD	SD	UD
6x6x2	0.0008	0.0005	0.0020	0.0009	0.1589	0.1715
6x8x5	0.0513	0.0011	0.0015	0.0016	0.2710	0.2951
6x9x3	0.0537	0.0008	0.0027	0.0011	0.4115	0.3682
7x7x5	0.0009	0.0007	0.0017	0.0019	0.3820	0.3531
7x8x4	0.0010	0.0009	0.0234	0.0016	0.3935	0.3686
8x8x3	0.0650	0.0012	0.0024	0.0022	0.7371	0.7125
9x9x2	0.0010	0.0010	0.0022	0.0023	2.5197	2.5222
10x10x2	0.0014	0.0013	0.0029	0.0025	7.4478	6.3323

Table 34: Average computation times for large problems with scenario 2

Matrix size	**CDS**		**NEH**		**GA**	
	SD	UD	SD	UD	SD	UD
12x12x10	0.0955	0.0107	0.0467	0.0166	42.5583	42.1651
15x15x5	0.0265	0.0049	0.0205	0.0197	69.3522	120.4441
20x20x4	0.0079	0.0044	0.0462	0.0614	230.1981	259.2478
25x25x8	0.0239	0.0993	0.1972	0.1954	768.6413	799.5785
30x30x5	0.0284	0.0330	0.2481	0.2468	915.1942	880.6319

Table 35: Average computation times for small problems with scenario 3

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
3x3x3	0.0376	0.0003	0.0004	0.0004	0.1700	0.0357
3x3x4	0.0229	0.0003	0.0004	0.0003	0.0283	0.0268
3x4x2	0.0003	0.0114	0.0003	0.0003	0.0283	0.0268
4x3x3	0.0003	0.0004	0.0558	0.0080	0.0292	0.0256
4x4x3	0.0005	0.0004	0.0005	0.0004	0.0242	0.0269
4x5x3	0.0432	0.0004	0.0004	0.0004	0.0744	0.0367
4x4x4	0.0004	0.0789	0.0005	0.0004	0.0382	0.0330
4x5x4	0.0016	0.0005	0.0005	0.0004	0.0383	0.0370

Table 36: Average computation times for medium problems with scenario 3

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
6x6x2	0.0005	0.0005	0.0008	0.0008	0.1348	0.1566
6x8x5	0.0384	0.0012	0.0013	0.0015	0.3576	0.3219
6x9x3	0.0010	0.0008	0.0011	0.0028	0.3577	0.3401
7x7x5	0.0455	0.0010	0.0995	0.0017	0.6113	0.3814
7x8x4	0.0495	0.0011	0.0017	0.0015	0.3265	0.3275
8x8x3	0.0009	0.0008	0.0017	0.0018	0.7582	0.7321
9x9x2	0.0008	0.0008	0.0556	0.0020	2.8332	2.6490
10x10x2	0.0010	0.0011	0.0033	0.0029	5.5785	5.6356

Table 37: Average computation times for large problems with scenario 3

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
12x12x10	0.0116	0.0203	0.0178	0.0181	40.2420	40.3960
15x15x5	0.0134	0.0051	0.0190	0.0196	68.5193	68.8014
20x20x4	0.0443	0.0085	0.0507	0.0475	286.6765	233.1291
25x25x8	0.0256	0.0296	0.2436	0.2278	750.8251	941.1693
30x30x5	0.0255	0.0828	0.2684	0.2942	970.1836	862.8904

Table 38: Average computation times for small problems with scenario 4

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
3x3x3	0.0435	0.0003	0.0041	0.0003	0.1935	0.0379
3x3x4	0.0468	0.0250	0.0071	0.0003	0.1007	0.0327
3x4x2	0.0003	0.0003	0.0004	0.0003	0.0569	0.0313
4x3x3	0.0137	0.0003	0.0005	0.0004	0.0352	0.0321
4x4x3	0.0068	0.0004	0.0073	0.0005	0.0315	0.0314
4x5x3	0.0004	0.0004	0.0005	0.0005	0.0427	0.0384
4x4x4	0.0004	0.0004	0.0005	0.0005	0.0386	0.0408
4x5x4	0.0005	0.0005	0.0005	0.0005	0.0431	0.0358

Table 39: Average computation times for medium problems with scenario 4

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
6x6x2	0.0639	0.0005	0.0009	0.0007	0.1735	0.2254
6x8x5	0.0011	0.0227	0.0014	0.0015	0.3685	0.4032
6x9x3	0.0009	0.0009	0.0011	0.0010	0.3834	0.3674
7x7x5	0.1554	0.0010	0.0024	0.0018	0.3639	0.3279
7x8x4	0.0010	0.0380	0.0016	0.0018	0.2642	0.3407
8x8x3	0.0008	0.0009	0.0059	0.0017	0.7296	0.6878
9x9x2	0.0498	0.0027	0.0025	0.0020	2.8699	3.4503
10x10x2	0.0009	0.0012	0.0030	0.0031	7.3924	7.4732

Table 40: Average computation times for large problems with scenario 4

Problem size	CDS		NEH		GA	
	SDR	UDR	SDR	UDR	SDR	UDR
12x12x10	0.0798	0.0054	0.0460	0.0174	46.6065	45.5247
15x15x5	0.0051	0.0055	0.0235	0.0245	81.8296	80.6558
20x20x4	0.0110	0.0091	0.0478	0.0477	255.6345	235.4703
25x25x8	0.1385	0.0287	0.2311	0.2422	798.6494	761.6909
30x30x5	0.0322	0.0306	0.3294	0.2889	867.2491	874.3069

Annexure D: GA parameter variations

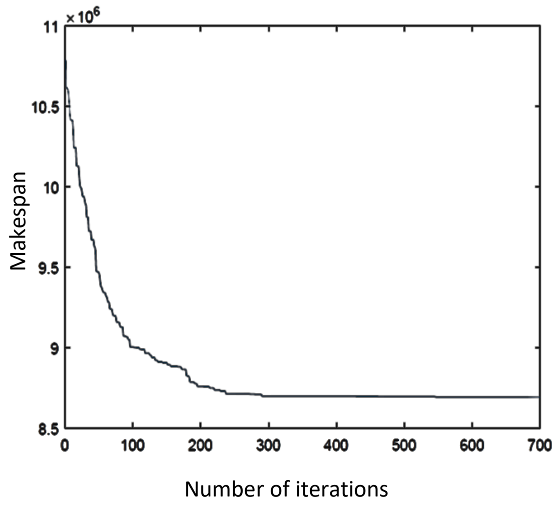
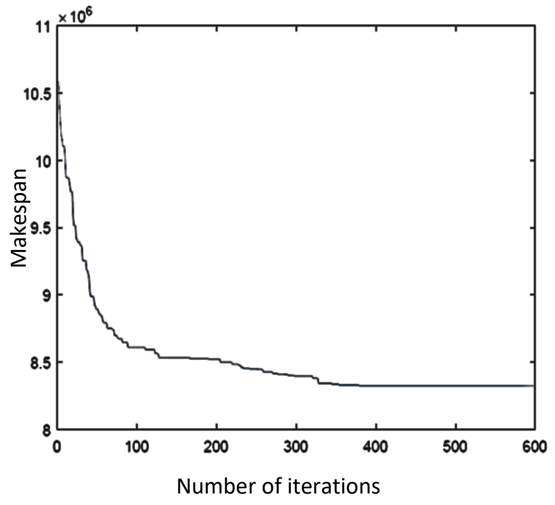
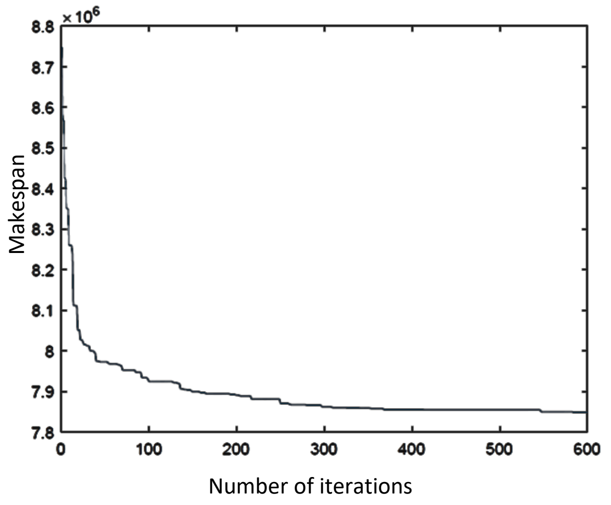
Population size = 1500	Population size = 750	Population size = 375
		

Figure 31: GA solution convergence for a 30x30x5 matrix with scenario 3 for various population sizes

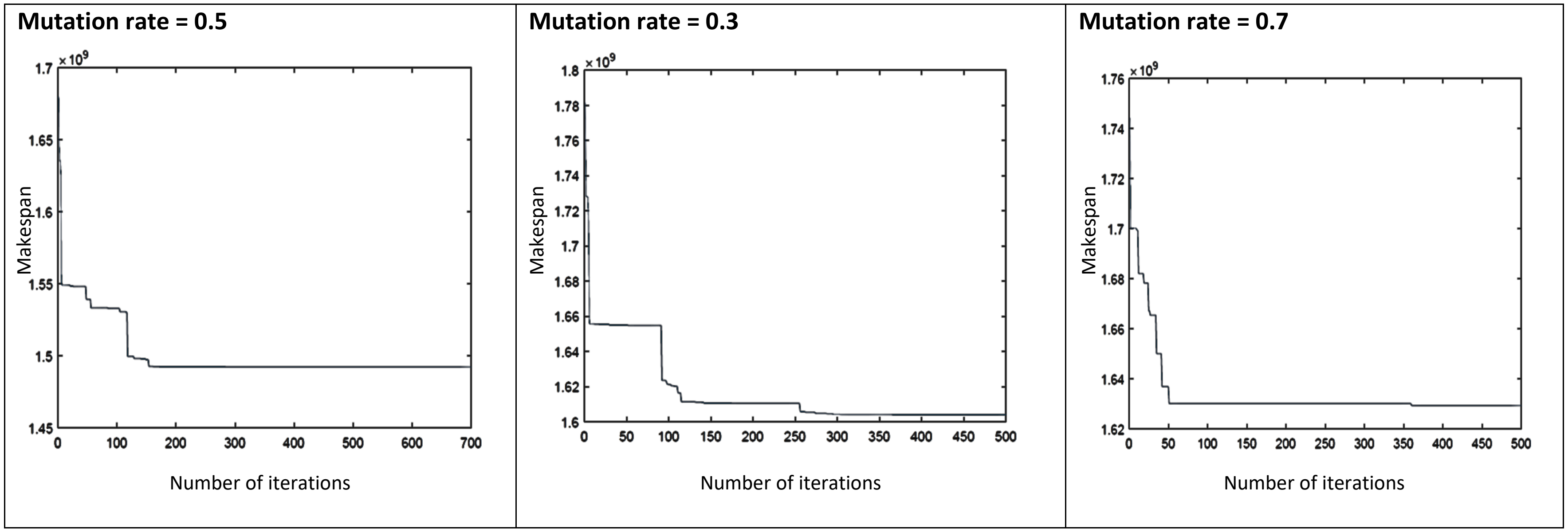


Figure 32: GA solution convergence for a 25x25x8 matrix with scenario 4 for various mutation rates