

**To improve the Makespan of a Standard Job Shop Scheduling
problem incorporating GA by using Python**



By

SAMRA YOUSAF

MS-DME'SMME

(Reg No:400010)

ADVISOR

DR. SHAHID IKRAMULLAH BUTT

Department of Design and Manufacturing Engineering DME
School of Mechanical and Manufacturing Engineering SMME
National University of Sciences and Technology,
H-12 Islamabad, Pakistan

(2024)

**To improve the Makespan of a Standard Job Shop Scheduling problem
incorporating GA by using Python**



By

Samra Yousaf

(Reg No: 400010)

Department of Design and Manufacturing
Engineering

A thesis submitted to National University of Science and
Technology, Islamabad, in partial fulfillment of the requirements
for the degree of

Master of Science in
Design and Manufacturing Engineering

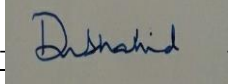
DR. SHAHID IKRAMULLAH BUTT

**Department of Design & Manufacturing Engineering
School of Mechanical & Manufacturing Engineering (SMME)**

**National University Of Sciences and Technology
H-12 Islamabad, Pakistan**

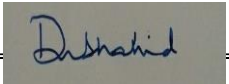
THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by **Regn No. 00000400010 Samra Yousaf** of **School of Mechanical & Manufacturing Engineering (SMME)** has been vetted by undersigned, found complete in all respects as per NUST Statues/Regulations, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis titled. **To improve the makespan of a Standard Job Shop Scheduling problem incorporating GA by using Python.**


Signature:  _____

Name of Supervisor: **Dr. Shahid Ikramullah Butt**

Date: 20 - Aug - 2024

Signature (HOD):  _____

Date: 20 - Aug - 2024

Signature (Principal):  _____

Date: 20 - Aug - 2024



National University of Sciences & Technology (NUST)

MASTER'S THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: Samra Yousaf (00000400010) Titled: To improve the makespan of a Standard Job Shop Scheduling problem incorporating GA by using Python. be accepted in partial fulfillment of the requirements for the award of MS in Design & Manufacturing Engineering degree.

Examination Committee Members

1. Name: Syed Hussain Imran Jaffery

Signature:

2. Name: Muhammad Rizwan Ul Haq

Signature:

Supervisor: Shahid Ikram Ullah Butt

Date: 20 - Aug - 2024

20 - Aug - 2024

Head of Department

Date

COUNTERSIGNED

20 - Aug - 2024

Date

Dean/Principal

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in this thesis, entitled “To Improve the Makespan of A Standard Job Shop Scheduling Problem incorporating GA by using Python” was conducted by Miss Samra Yousaf under the supervision of Dr Shahid Ikramullah.

No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the Department of Design and Manufacturing Engineering in partial fulfillment of the requirements for the degree of Master of Science in Field of Department of Design and Manufacturing Engineering by Department of Design and Manufacturing Engineering, National University of Sciences and Technology, Islamabad.

Student Name: Samra Yousaf

Signature: 

Examination Committee:-

a) External Examiner 1: Dr. Hussain Imran

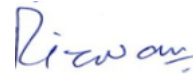
Signature:



(Asst Professor, DME SMME, NUST)

b) External Examiner 2: Dr. Muhammad Rizwan

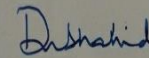
Signature:



(Asst Professor, DME SMME, NUST)

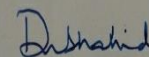
Supervisor Name: Prof. Dr. Shahid Ikramullah

Signature: _____



Name of Dean/HOD: Prof. Dr. Shahid Ikramullah

Signature: _____



AUTHOR'S DECLARATION

I Samra Yousaf hereby state that my MS thesis titled “To improve the Makespan of a Standard JobShop Scheduling problem incorporating GA in Python” is my own work and has not been submitted previously by me for taking any degree from National University of Sciences and Technology, Islamabad or anywhere else in the country/ world.

Name of Student: SAMRA YOUSAF

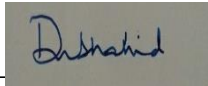
Date: 20-August-2024

Certificate for Plagiarism

It is certified that MS Thesis Titled “To Improve the Makespan of a Standard Job shop Scheduling problem incorporating GA by using Python” by Samra Yousaf, Regn. No. 00000400010 has been examined by us. We undertake the follows:

- a. Thesis has significant new work/knowledge as compared to already published or is under consideration to be published elsewhere. No sentence, equation, diagram, table, paragraph or section has been copied verbatim from previous work unless it is placed under quotation marks and duly referenced.
- b. The work presented is original and own work of the author (i.e. there is no plagiarism). No ideas, processes, results or words of others have been presented as Author own work.
- c. There is no fabrication of data or results which have been compiled/analyzed.
- d. There is no falsification by manipulating research materials, equipment or processes, or changing or omitting data or results such that the research is not accurately represented in the research record.
- e. The thesis has been checked using TURNITIN (copy of originality report attached) and found within the limits as per HEC Plagiarism Policy and instructions issued from time to time.

Signature of Supervisor _____



Name of Supervisor: Dr. Shahid Ikramullah Butt

**Dedicated to my Parents, my sister and my teachers
whose tremendous support and cooperation led me to
this wonderful accomplishment.**

ACKNOWLEDGEMENTS

"In the name of Allah, the Magnificent, the Merciful."

Research is a continuous quest, and I am blessed to have the support of my family and friends. Along this journey, I've faced numerous challenges, but as Einstein famously remarked, "Anyone who has never made a mistake has never tried anything new."

At each obstacle, Dr. Shahid Ikramullah has been a guiding light, illuminating my path. I am profoundly grateful to Dr. Shahid Ikramullah, my research advisor, for his unwavering support. Additionally, the encouragement and guidance from my Guidance and Examination Committee have been invaluable.

I extend my deepest gratitude to my parents and siblings, whose relentless support has been a cornerstone of my life. I am also thankful to my peers, whose insightful feedback and camaraderie have greatly enriched my journey.

To all who have been part of this journey, thank you. I wish you all success in your future

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	IX
TABLE OF CONTENTS	X
LIST OF FIGURES	XIII
LIST OF TABLES	XIV
ABSTRACT.....	1
CHAPTER 1: INTRODUCTION.....	2
1.1 Problem Statement.....	2
1.2 Purpose of Research	3
1.3 Manufacturing Layouts.....	3
1.3.1. Project production.....	4
1.3.2. Job-shop production.	4
1.3.3. Batch production.....	4
1.3.4. Mass production.	4
1.3.5. Flow/Process production	4
1.4 General types of facility layouts.....	5
1.4.1. Fixed Position Layout.....	5
1.4.2. Process Layout:.....	5
1.4.3. Cellular Layout	5
1.4.4. Product Layout	5
1.5 Machine Environments.....	5
1.6 Literature Review	7
CHAPTER 2: SCHEDULING.....	9
2.1 Classification of Job Scheduling	9
2.1.1 Single machine scheduling.....	9
2.1.2 Flow shop scheduling	11
2.1.3 Job shop scheduling.....	12

2.2 Assumptions for JSSP	12
2.3 Scheduling Algorithms	13
2.3.1 Classification of Scheduling Algorithms.....	13
2.4 Evolutionary Algorithms	16
2.5 Scheduling Rules	17
2.6 Classification scheme for scheduling problems	18
2.7 Scheduling Criteria	18
CHAPTER # 3: Introduction to Genetic Algorithm - GA	20
3.1 Fundamental Models	21
3.2 A Simple GA	22
3.3.1. Crossover (Recombination).....	23
3.3.2. Mutation.....	25
3.4 Local Optima	27
3.5 Development of a Genetic Algorithm for Job Shop Scheduling	28
3.5.1 Genetic Operators	29
3.5.2 Local search method:	31
3.5.3 Evolutionary Process	31
3.5.4 Experimental Setup and benchmarks	32
3.5.5 Basic Flowchart for GA.....	35
3.6 Manual Example:.....	35
3.7 Initial Makespan Calculator.....	38
3.7.1 How it works?.....	39
Chapter # 04: Python Libraries and GUI Development for Scheduling Optimization.....	41
4.1 Python Libraries	41
4.1.1 Random.....	41
4.1.2. Matplotlib (including matplotlib.patches)	42
4.1.3. Tkinter (including simpledialog, messagebox, Toplevel, Scrollbar, Frame, Canvas)	42
4.1.4. Matplotlib.backends.backend_tkagg (FigureCanvasTkAgg)	42
4.2 Interactive Flow Chart: Navigating the GUI-User Experience	43
4.2.1 Main Menu:	43
4.2.2 Other Benchmarks	46

4.3 Importance of Introducing Programming Languages like Python in Manufacturing	50
Chapter 5: GA performance Evaluation	52
5.1 Flowchart for GA working	52
5.2 Complete Python Pseudo Code	52
5.3 Results for Test problem.....	59
5.4 Test for MT06.....	60
5.5 Test for MT10.....	62
5.6 Test for MT20.....	64
5.7 GA result for dataset instances from Muth and Thomson.....	65
5.8 GA result for dataset instances from (Lawrence, 1984).....	65
5.8.1 LA01	65
5.9 GA result for dataset instances from (Lawrence, 1984).....	67
Chapter 6: Conclusion and Future Directions	68
6.1 Conclusion.....	68
6.2 Contributions	68
6.3 Implications	69
6.4 Future Directions	69
References.....	72

LIST OF FIGURES

Figure 1: Manufacturing Layouts	4
Figure 2:Flow Shop Layout	6
Figure 3: Open Shop Layout.....	6
Figure 4: Job Shop Layout.....	7
Figure 5: Classification of Scheduling Algorithms	13
Figure 6:Evolutionary Algorithms.....	17
Figure 7:Natural Evolution with GA	21
Figure 8:Simple GA search process.....	27
Figure 9: Initial Maespan Calc.....	37
Figure 10: Result Message.....	37
Figure 11: Visual Studio Code.....	41
Figure 12: Interactive Flow Chart: Navigating the GUI-User Experience	43
Figure 13: Main Menu for JSSP	44
Figure 14: Run Algorithm for MT06.....	45
Figure 15: Other Benchmarks	46
Figure 16: Start Algorithm Button for others	49
Figure 17: Flowchart for Beckend (GA).....	52
Figure 18: GANTT CHART for Test Problem.....	60
Figure 19: Fitness Graph for Test Problem.....	60
Figure 20: GANTT CHART for MT06	61
Figure 21: Fitness Graph for MT06	62
Figure 22: GANTT CHART for MT10	63
Figure 23: Fitness Graph for MT10	63
Figure 24: GANTT CHART for MT20	64
Figure 25: Fitness Graph for MT20.....	65
Figure 26: LA01 Benchmark	66
Figure 27: GANTT CHART for LA01	66
Figure 28: :Fitness Graph for LA01.....	67

LIST OF TABLES

Table 1: Contrast of Natural Evolution with Genetic Algorithm	21
Table 2: Experimental setup	32
Table 3: Processing Times for Manual Problem.....	35
Table 4: Initial Population:	36
Table 5: Start and Finish times for each job	36
Table 6: New Chromosomes After Crossover	37
Table 7: Mutation.....	38
Table 8: Test Problem	59
Table 9: MT06 problem	61
Table 10: MT10 problem	62
Table 11: MT20 problem	64
Table 12: GA Results for Muth and Thomson.....	65
Table 13: GA Results for Lawrence Instances	67

ABSTRACT

Effective job scheduling is crucial in industrial manufacturing planning, where each job, consisting of multiple operations, must be allocated to the machines that are available machines for processing. Each job has a specific interval, and every machine can only handle one operation at a time. Efficient job allocation is essential to minimise the makespan and reduce machine idle time. In Job Shop Scheduling (JSS), job operations follow a specified order. Genetic Algorithms (GA) have emerged as a popular heuristic for tackling various scheduling problems. This study introduces a Genetic Algorithm Integrating Python (GAIP) with feasibility-preserving solution representation, initialization, and operators tailored for the JSS problem. The proposed GAIP achieves the best-known results with high success rates on the Muth and Thomson and Lawrence benchmark datasets. Experimental results demonstrate the GA's rapid convergence towards optimal solutions. Incorporating GA with local search and two selection methods at the same time is done to further enhance solution quality and success rates.

Keywords: Job Shop Scheduling, Genetic Algorithm , Manufacturing Planning, Makespan Optimization, Python, , Initialization, Lawrence Datasets, Local Search, Hybrid Algorithms.

CHAPTER 1: INTRODUCTION

Effective scheduling is a cornerstone of operational success in various industries, particularly in manufacturing, where precision and efficiency directly impact productivity and profitability. The ability to optimise the allocation of resources, minimise delays, and ensure the smooth flow of operations is crucial in environments where a series of machines are available to process multiple jobs.

Advanced scheduling techniques are integral to addressing the complexities of modern manufacturing processes. These techniques help to minimise downtime, reduce operational costs, and improve overall throughput. In industries such as automotive, aerospace, and electronics, the coordination of intricate tasks and the synchronisation of machine operations are vital for maintaining competitive advantage and meeting customer demands.

The continuous evolution of manufacturing technologies and the increasing complexity of production processes have amplified the need for sophisticated scheduling algorithms. Traditional heuristic methods often fall short when dealing with the multifaceted nature of real-world scheduling problems. As a result, there is a growing demand for innovative solutions that can handle the dynamic and complex environment of modern manufacturing.

1.1 Problem Statement

Job shop scheduling presents significant challenges, primarily due to its combinatorial nature and the constraints involved in the sequencing and allocation of jobs. One of the most critical aspects of JSS is the optimization of the makespan, which is the whole time essential to complete all jobs.

The makespan is a crucial performance metric in manufacturing, as it directly affects production efficiency, lead times, and overall operational costs. Traditional methods for solving job shop scheduling problems, such as exact algorithms and heuristic methods, often struggle with scalability and computational efficiency, especially as the problem size increases. These methods could be computationally expensive and may not provide optimal solutions within a reasonable time frame. Hence, there is a need for more efficient and effective algorithms that can handle the complexities of job shop scheduling and deliver optimised makespan solutions.

Hence, there is a need for more efficient and effective algorithms that can handle the complexities of job shop scheduling and deliver optimised makespan solutions. Also, there is a need to introduce python programming language in manufacturing so that it can handle combinatorial problems in more efficient way and can also develop a user friendly interface where user can test for vaious scheduling problems.

1.2 Purpose of Research

- Main goal is to develop a robust and efficient genetic algorithm (GA) incorporating Python (GAIP) to improve the makespan in job shop scheduling problems. Specific aims include:
- Designing a GA using Python with customized solution representation, initialization, and operators tailored for JSSP.
- Incorporating local methods to improve the performance of the GA and ensure faster convergence towards optimal solutions.
- Validating the proposed GA against standard benchmark datasets, such as the Muth and Thomson datasets and Lawrence datasets, to assess its effectiveness and reliability. Also, there is an aim to design the system in such a way that user can test custom bencharks also.
- Developing a user-friendly graphical user interface (GUI) in Python to facilitate input, execution, and visualization of job shop scheduling results.

1.3 Manufacturing Layouts

"Manufacturing layouts" refer to the physical arrangement of machinery and equipment within a factory or production area. This layout determines the movement of materials and work through the production organisation. Effective layouts are crucial for optimising manufacturing efficiency and can directly impact production processes and their capacity, worker efficiency, and safety.

Production processes can be categorized into 5 in aspect of :

- continuousness,
- product diversity
- Volume of Production:

1.3.1. Project production (low volume,intricate process, , each project or product has unique processing sequence,high customizationand fixed position layout)..

1.3.2. Job-shop production (manufacturing a limited quantity of products, characterized by low volume and high variety. Each job has distinct technical requirements, necessitating highly expert operators and typically leading to the need for substantial inventory levels).

1.3.3. Batch production (comparatively shorter production runs, flexible plants and machines, lower manufacturing cost with lower lead compared to job-shop production).

1.3.4. Mass production (comparatively briefer production runs, flexible plants and machines, lesser manufacturing cost).

1.3.5. Flow/Process production (Manufacturing includes producing a huge volume of products with slight variety, often concentrating on just one type. The process uses specific machines organised in a fixed sequence).

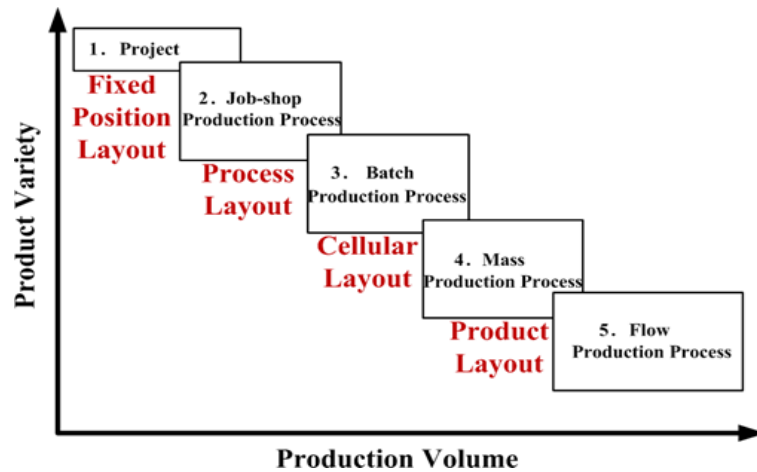


Figure 1: Manufacturing Layouts[2]

1.4 General types of facility layouts

Following are the general types of facility layouts

1.4.1. Fixed Position Layout: A Fixed Position Layout is typically employed when the product is too large, heavy, or difficult to move. This layout is common in industries like construction, shipbuilding, and the manufacturing of large industrial equipment. In this setup, the product remains in one location, while workers, materials, and machinery are brought to it as needed. While this approach reduces the need to move the product, it can result in higher costs and increased complexity in coordinating the movement of labor and materials

1.4.2. Process Layout: Also known as a job shop layout, it groups similar processes in the same area. It's used for operations that produce a variety of products in relatively low volumes. This layout is flexible and can handle a wide variety of work items but can have higher material handling costs and longer processing times. Typical applications include specialised machine shops, hospitals, and custom furniture makers.

1.4.3. Cellular Layout: Organises the production floor into cells, each of which is dedicated to a specific set of processes that are required to produce a family of similar products or components. This layout aims to join the process layout's flexibility with a product layout's efficiency. It's suitable for medium-volume production where families of products can be grouped. This can lead to reduced work-in-process inventory and shorter processing times.

1.4.4. Product Layout: Used for mass production of standardized products where the production volume is high enough to justify specialized workstations laid out in a line. This layout is highly efficient for large runs of production, reducing material handling and setup times but lacking flexibility for product changes. It's commonly used in automotive assembly lines, food processing, and electronics manufacturing.

1.5 Machine Environments

The term "machine environments" typically refers to the specific configuration and operational settings of machinery within a production process. This can involve

various types of machinery such as single machines, parallel machines, or a series of machines that are part of a production line. In the context of scheduling and operations research, different machine environments might have distinct characteristics or constraints, such as:

- **Single Machine:** Tasks are processed on a single machine.
- **Parallel Machines:** Multiple tasks are processed simultaneously on different machines that might have the same or varying capabilities.
- **Flow Shop:** A specific type of production process where tasks are processed in the same sequence on multiple machines.

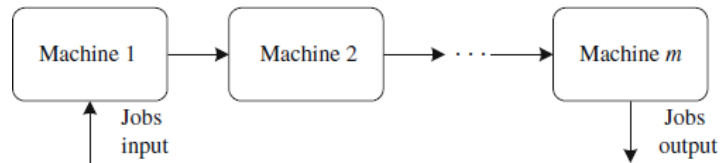


Figure 2:Flow Shop Layout

- **Open Shop Layout:** In open shop jobs must be processed for a certain amount of time at each sets of defined work centres irrespective of the proper sequence. No predetermined sequence is followed therefore jobs can be handled in any sequence

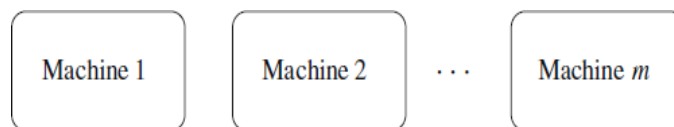


Figure 3: Open Shop Layout

- **Job Shop:** A job shop is a manufacturing setup designed to handle highly customised, small to medium batch production, where each product may require a distinct sequence of processes and operations. This type of environment is characterised by its significant flexibility, allowing for the manufacture of a diverse range of products. Machines in a job shop are typically organized by function rather than by specific product lines, which contrasts with flow shops where machinery is arranged in the order of the manufacturing process. This

functional grouping allows each job to take a unique route through the workshop, depending on the specific operations required. However, this flexibility comes at a cost, as job shops often face complex scheduling challenges.

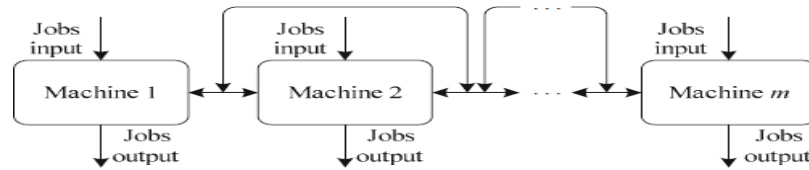


Figure 4: Job Shop Layout

1.6 Literature Review

The literature on Job Shop Scheduling (JSS) has seen significant contributions, particularly in the application of Genetic Algorithms (GA) and other optimization techniques.

Kalshetty et al. [1] introduced feasible operators in GA for solving JSS problems, focusing on feasibility and efficiency. However, the study lacks comprehensive hybridization with local search methods, which could enhance the optimization process. My work incorporates a robust local search heuristic, which improves the convergence rate and solution quality, filling this gap.

Ripon et al. [2] proposed an Improved Precedence Preservation Crossover (IPPX) for multi-objective JSS problems, which showed promise in maintaining solution integrity across multiple criteria. Nevertheless, the study did not fully explore the potential of integrating IPPX with advanced local search methods for makespan optimization. My research bridges this gap by integrating such methods, resulting in better convergence and optimized solutions.

Ikramullah and Hou-Fang [3] applied GAs to flexible JSS, incorporating rule-based approaches. Although effective, their method did not adequately address the issue of scalability and the complexities associated with real-world job shops. My work extends their approach by providing a more scalable solution that efficiently handles

larger, more complex scheduling problems through enhanced genetic operators and local search techniques.

Chang et al. [4] utilized a hybrid Taguchi-GA to optimize makespan in flexible JSS, achieving significant improvements. However, their approach may suffer from convergence issues due to the deterministic nature of the Taguchi method. My research mitigates this by employing stochastic elements in local search, ensuring a broader exploration of the solution space and avoiding premature convergence.

Adams et al. [5] and Applegate and Cook [6] focused on local search heuristics for JSS, which laid the foundation for subsequent research. However, these studies primarily addressed static job shops and did not fully integrate modern heuristic approaches. My work builds on their foundation by integrating advanced GA techniques with dynamic local search heuristics, suitable for both static and dynamic job shop environments.

Kundakcı and Kulak [7] explored hybrid GAs for dynamic JSS, focusing on makespan minimization. While their approach demonstrated efficacy, it lacked flexibility in handling diverse job shop configurations. My research introduces a more adaptable framework that can accommodate various job shop layouts and configurations, enhancing its applicability in real-world scenarios.

Barat's thesis [10] on job scheduling using GA provided valuable insights but was limited by its reliance on traditional GA operators without incorporating hybridization with local search. My work addresses this limitation by implementing a hybrid GA framework, which significantly improves solution quality and robustness.

CHAPTER 2: SCHEDULING

Scheduling and planning are the two key activities carried out in industries for manufacturing products. Planning involves estimating the necessary actions and identifying any constraints on how these actions should be performed. Scheduling, on the other hand, focuses on estimating the time and resources required for each activity and allocating the optimal resources for task execution. Additionally, scheduling determines the precedence relationships between activities and the associated constraints. For a plan to be fully executed, tasks and activities must be assigned a specific timeline. Effective scheduling offers several advantages. Improved on-time delivery

- Reduced inventory
- Cut lead times
- Increased bottleneck means utilisation

However, due to the combinatorial nature of scheduling problems, finding the optimal schedules can be challenging. Scheduling is considered NP-complete because determining the best schedule for 'a' number of jobs on 'b' number of machines requires optimal utilization of both resources and time. In the traditional Job Shop Scheduling Problem (JSSP), jobs are treated as activities, while machines are regarded as resources.

2.1 Classification of Job Scheduling

Job scheduling can be categorized into three types: single machine scheduling, flow shop scheduling, and job shop scheduling.

2.1.1 Single machine scheduling

In this type, several jobs are allocated to a single machine for processing. The machines involved in this process can be classified into two categories: dependent and independent. When the setup time for each job is not influenced by others, it is termed a single-machine scheduling problem with independent jobs. Conversely, if the setup time depends on the sequence of jobs, it is known as a single-machine scheduling problem with dependent jobs. The function is measured using the following metrics.

- **Mean Flow Time:** The average time a job spends in the system from arrival to completion.
- **Maximum Lateness:** The greatest delay of any job beyond its scheduled due date.
- **Total Tardiness:** The cumulative delay of all jobs beyond their respective due dates.
- **Number of Tardy Jobs:** The count of jobs completed after their due dates.

In single machine scheduling, multiple jobs are assigned to a single machine for execution. The machines used for job allocation can be classified into two types: dependent and independent. If the setup time for each job is independent of others, the problem is referred to as a single-machine scheduling problem with independent jobs. Conversely, if the setup time depends on the sequence of jobs, it is known as a single-machine scheduling problem with dependent jobs.[1].

1) Identical parallel machine scheduling problem: Here parallel machines function at matching speeds, and jobs assigned to them need the same processing time. A branch and bound algorithm is employed to lessen job tardiness within this framework. After processing a specified number of jobs, each machine requires a preventive preservation task.

2) Proportional parallel machine scheduling problem: Here parallel machines operate at varying speeds, with the first machine being the slowest and the last the fastest. This scheduling model addresses the challenge of scheduling jobs with similar due dates and proportionate early and tardy disadvantages across identical parallel machines. Sun & Wang analysed these issues and confirmed that the scheduling is an NP-hard problem. Dynamic programming is subsequently used to address the complexities involved in the scheduling process.

3) Unrelated parallel machine scheduling problem

In this scheduling model, the processing times of jobs on parallel machines are independent, reflecting variations in technology due to factors like differing machine capabilities and job characteristics. Abdelmaguid introduced an iterated greedy

algorithm for large-scale unrelated parallel machine scheduling. This algorithm iteratively refines solutions through a cycle of destruction and construction phases, offering optimal performance compared to traditional metaheuristics. Additionally, Torabi developed a Multi-Objective Particle Swarm Optimization (MOPSO) to estimate the optimal approximation of the Pareto frontier effectively. This approach leverages selection regimes to identify the best personal and global solutions, outperforming the conventional Multi-Objective Particle Swarm Optimization (CMOPSO) in terms of quality, diversity, and spacing.

2.1.2 Flow shop scheduling

In this scheduling approach, jobs are assigned to different machines, with each job adhering to a specific process sequence. All jobs follow the same process sequence. The effectiveness of flow shop scheduling is assessed using the following metrics

- Maximum lateness
- Total tardiness
- Number of tardy jobs
- Makespan

A Memetic algorithm called Opposition-based Differential Evolution (ODDE) is proposed to tackle the Permutation Flow Shop Problem (PFSSP). To adapt ODDE for PFSSP, the Largest-Ranked Value (LRV) rule is applied, which transforms the continuous positions generated by Differential Evolution (DE) into discrete job permutations. The Nawaz-Enscore-Ham (NEH) method is integrated with random population initialization to ensure a balance of quality and diversity. Additionally, the crossover rate is adjusted by leveraging DE's global optimization capabilities. By deploying the opposition based learning for the initialization and generation jumping for the global optimum solution enhancement, the convergence rate of the DE is enhanced. Individuals are improved with a certain probability through a rapid local search process. A pairwise-based local search technique is employed to refine the global optimum solution, helping to avoid the algorithm getting trapped in local

minima. To address the Distributed Permutation Flow-shop Scheduling Problem (DPFSP), an effective estimation of a Distributed Algorithm (EDA) is proposed. Optimal schedules are generated by applying completion factory rules, and the probability distribution of the solution space is represented through a probability model.

2.1.3 Job shop scheduling

Job shop scheduling (JSS) is a complex optimization problem within operations research and computer science that focuses on effectively allocating jobs to machines. The scheduling decisions in JSS are influenced by factors such as the routing sequence of jobs and their respective processing times. The classical JSS problem, a cornerstone in the study of operational strategies, involves planning the order in which jobs are processed on various machines to optimize specific objectives, such as minimizing the total completion time of all operations.

JSS can be classified into two types: static and dynamic. Static scheduling, which is the focus of this thesis, involves scenarios where all information about the jobs (including processing times and job sequences) is known in advance, and the schedule is determined before execution begins. Conversely, dynamic scheduling adapts to information that becomes available during the operations, such as machine breakdowns or changes in job priorities, requiring real-time adjustments to the schedule. The static approach is critical for environments where predictability and pre-planning mitigate potential disruptions and optimize workflow efficiency.

2.2 Assumptions for JSSP

Following are the assumptions [1], [17] that must be followed during static JSSP.

1. All jobs are ready to begin at time zero.
2. Each machine can handle only one operation at a time.
3. Every operation can be processed on only one machine at any given moment.
4. The operations for each job must be completed in a specified sequence.
5. All operations must be finished across the designated set of machines.

The goal of the scheduling task is to optimize a specific criterion, which serves as a performance measure for the schedule. One such criterion is the makespan, which refers to the total time required to complete all the jobs.

2.3 Scheduling Algorithms

An algorithm is a set of well-defined rules that are designed to perform a specific task or solve a particular problem. It's a step-by-step procedure, typically used in computing, to input data and produce a desired output or result. An Algorithm is fundamental to all aspects of the field of computer science and are used to automate, enhance, and streamline processes in various fields.

Scheduling Algorithms are a specific category of algorithms used to assign resources to tasks over time, aiming to optimise certain objectives like minimising total completion time, maximising resource utilisation, or balancing load among resources. These algorithms are crucial in fields ranging from manufacturing and logistics to computing and telecommunications. These algorithms help in completing tasks properly in an efficient manner and resources are used effectively.

2.3.1 Classification of Scheduling Algorithms

These algorithms can be classified as:

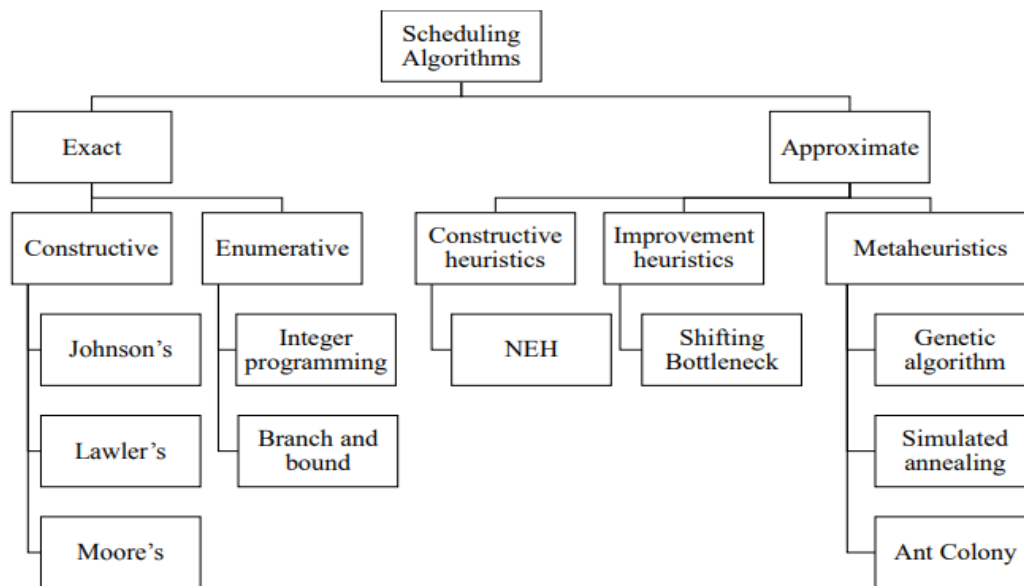


Figure 5: Classification of Scheduling Algorithms[3]

To address scheduling challenges, a variety of methods are categorized broadly into two types: Exact and Approximate methods.

Exact methods, though precise, are often impractical for large-scale Job Shop Scheduling Problems (JSSP) due to their computational intensity and time-consuming nature. As a result, heuristic methods, which provide approximate yet high-quality solutions, are preferred, particularly when real-time execution and feasible solutions are essential. These heuristic approaches are designed to approach optimal solutions closely within a considerably shorter timeframe compared to exact methods.

In practice, achieving an absolute optimal solution is seldom possible, making a robust approximate solution highly valuable. Heuristics are therefore a focal point for researchers aiming to develop algorithms that can efficiently converge near-optimal solutions under practical constraints. Approximate methods typically model production scheduling problems through a series of equality or inequality constraints, optimizing the target function within these defined parameters.

Hybrid methods [18], which combine two or more of the best-performing heuristic rules, are prominent in contemporary research, enhancing the effectiveness of traditional approaches. These methods are particularly useful not only in static but also in dynamic JSSP, where the complexity escalates with the increasing number of machines and job flows. As the complexity and the number of feasible solutions grow exponentially, approximate methods offer a pragmatic means to achieve optimal solutions within a reasonable timeframe. In smaller-scale industries, where job orders often arrive unpredictably and each batch has distinct due dates, scheduling must prioritise and complete jobs in alignment with these timelines. This necessitates a strategy that integrates priority dispatching rules, which will be further explored and detailed in subsequent sections, illustrating a practical JSSP scenario from a small-scale industrial setting.

1. Constructive Algorithms (Exact)

- **Johnson's Rule:** Efficient for two-machine sequencing problems where jobs must be done on both machines in a specific order.

- **Lawler's Algorithm:** Minimizes maximum lateness by sequencing jobs based on their deadlines and processing times.
- **Moore's Algorithm:** Focuses on minimizing the number of late jobs by reordering tasks based on their due dates.

2. Enumerative Algorithms (Exact)

- **Integer Programming:** Uses mathematical formulations to find the exact solution by exploring all possible combinations.
- **Branch and Bound:** Reduces the search space by logically eliminating sequences that do not lead to an optimal solution.

3. Constructive Heuristics (Approximate)

- **NEH (Nawaz-Enscore-Ham):** Prioritizes jobs based on their total processing times across all machines, building the schedule by adding one job at a time.

4.Improvement Heuristics (Approximate)

- **Shifting Bottleneck:** Addresses the most critical part of the production process first and sequentially adds other parts to improve the overall schedule.

5. Metaheuristics (Approximate)

- **Genetic Algorithm:** Simulates natural selection processes to iteratively evolve solutions to optimal or near-optimal levels.
- **Simulated Annealing:** Mimics the cooling process of materials, allowing solutions to 'cool' and stabilize into a minimal energy state.
- **Ant Colony Optimization:** Models the behavior of ants searching for food to find optimal paths through graphs, applicable to scheduling.

Metaheuristics and nature-inspired optimization algorithms have become increasingly prominent in addressing the complexities of Job Shop Scheduling Problems (JSSP). Extensive research has explored various methodologies that leverage these advanced optimization techniques to enhance engineering problem-solving capabilities. Among the notable approaches, alternative graph solution algorithms have

been tailored for general formulations of JSSP, accommodating unique constraints such as blocking and no-wait scenarios. These algorithms are designed with flexibility, allowing for easy modification to include new constraints and reconfiguration for multi-objective cases.

Further innovations include the adaptation of the Ant Colony algorithm, which has been refined to resolve JSSP efficiently within acceptable time frames. The integration of fuzzy reliability functions also plays a crucial role in assessing system reliability, particularly when faced with uncertain information. This approach underscores the adaptability of metaheuristic algorithms in managing not only the operational efficiency but also the reliability and robustness of systems.

Recent developments have seen the application of algorithms such as Grey Wolf Optimization and Cuckoo Search Algorithm aimed at minimizing costs while maximizing the availability and operational time of system components. These nature-inspired techniques contribute significantly to enhancing system reliability, productivity, and profitability.

Additionally, Particle Swarm Optimization (PSO) has gained recognition as one of the most effective algorithms for solving JSSP. Efforts to augment PSO include the introduction of hybrid methods that combine PSO with other optimization techniques, demonstrating a potent capability for tackling both constrained and unconstrained nonlinear optimization challenges.

These advancements highlight the efficacy of metaheuristic algorithms in solving complex real-world problems, positioning them as invaluable tools in the continuous evolution of optimization practices in job shop scheduling and beyond.

2.4 Evolutionary Algorithms

Evolutionary algorithms (EAs) are optimization approaches that draw inspiration from the principles of natural selection and the survival of the fittest. Unlike traditional optimization approaches, EAs work with a "population" of potential solutions rather than just one. Each iteration of an EA includes a competitive selection process that

removes less effective solutions. A basic structure of evolutionary algorithms is depicted in the figure.

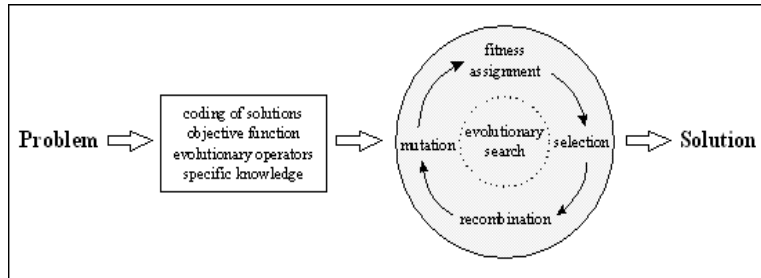


Figure 6:Evolutionary Algorithms[3]

2.5 Scheduling Rules

In scheduling, specific strategies or rules(like FCFS, SJF, EDF) are used to prioritise and manage tasks efficiently across various systems. These strategies are fundamental to ensuring that tasks are completed in an optimal manner, respecting constraints such as deadlines, resource availability, and operational efficiency. These can be used within both exact and approximate frameworks to handle specific scheduling needs in various systems [17].

- **First-Come, First-Served (FCFS):** Simple, treats all tasks with equal priority as they come.
- **Shortest Job First (SJF):** Optimizes waiting time by handling the shortest tasks first.
- **Priority Scheduling:** Manages tasks based on predefined priorities.
- **Round Robin (RR):** Ensures all tasks receive equal time slices, commonly used in computing.
- **Earliest Deadline First (EDF):** Critical for systems where tasks must meet deadlines.
- **Least Slack Time First (LSTF):** Prioritizes tasks with the least slack time to avoid delays.

2.6 Classification scheme for scheduling problems

Classification system for scheduling problems is as follows:

1) Requirement Generation: Manufacturing shops are considered as either closed or open founded on how they handle order contentment. In a closed shop, orders are fulfilled using current inventory. In contrast, an open shop functions on a build-to-order basis, meaning no inventory is kept in stock.

2) Processing Complexity: It the number of steps of processing and the machines or workstations related to production process.

The single-stage, single-processor and single-stage, multiple-processor scenarios involve tasks that need to be completed using either one or several possessions, respectively. In the multi-stage flow shop scenario, each job is composed of multiple tasks that must be processed by different resources, following a standardized route for all jobs. Conversely, in the multi-stage job shop environment, jobs may utilize alternative sets of resources and follow different routes, enabling the manufacturing of various part types

2.7 Scheduling Criteria

It tells the preferred objectives to be happened which can be many, intricate and regularly differing. Some of the most common scheduling criteria contain the following:

- a. Lessen whole tardiness
- b. Lessen the number of late jobs
- c. Maximize system/resource use
- d. Minimize in-process record
- e. Balance resource usage
- f. Maximize production rate

Parameter Variability: This term describes the level of uncertainty in scheduling problem parameters. When uncertainty is minimal (for example, when the

variance in processing times is significantly smaller than the expected durations), the scheduling is considered deterministic, such as when an expected processing time is six hours with only a minute of variance. Conversely, if uncertainties are substantial, the scheduling is deemed stochastic.

Scheduling Environment: Well-defined by whether the scheduling context is static or dynamic. In static scheduling, all job information, including the number and readiness of jobs, is predetermined. Dynamic scheduling, on the other hand, involves fluctuating job numbers and characteristics over time. In many manufacturing settings, scheduling is often manually conducted by experienced personnel using tools like pencil, paper, Gantt charts, and possibly an industrial database, making it a complex and challenging task, particularly in environments characterized by low volume and high variety.

CHAPTER # 3: Introduction to Genetic Algorithm - GA

Methods of scheduling are discussed in detail in first and second chapters, so the main emphasis in this chapter is on the selected technique of Genetic Algorithm (GA) and its application to generate schedule. Genetic algorithms (GAs) is related to the class of evolutionary algorithms and are inspired by the process of natural genetic evolution. The original concept of natural evolution was proposed by Charles Darwin [14], who introduced the idea that natural populations evolve through the natural selection process and its basis is "survival of the fittest "principle. Darwin's theory has since become a cornerstone in the field of biology, explaining how species adapt over generations to their environments. Building on this, John Holland conducted pioneering work on genetic algorithms in 1975, further extended by David Goldberg[15], who contributed significantly to their development and practical applications.

The analogy of giraffes evolving longer necks to reach higher leaves exemplifies the principle of survival of the fittest. Giraffes with longer necks could access more food, survive better, and pass on their advantageous traits to future generations. Similarly, genetic algorithms mimic this natural process of evolution to solve complex engineering problems. The elegance of GAs lies in their adaptive nature; they can dynamically adjust to changing environments, making them highly versatile and robust optimization tools.

At the core of a GA lies the concept of a gene, the fundamental unit of information. Genes are combined to form a chromosome, which is a possible solution of the scheduling problem. Each chromosome has a series of genes, and these genes encapsulate the state data in a structured format.

A chromosome, also referred to as an individual, embodies a single candidate solution within the search space. The collection of these individuals forms a population, representing a diverse set of potential solutions. The diversity within the population is crucial, as it provides a broad range of genetic material for the algorithm to work with, enhancing the ability to explore various regions of the solution space.

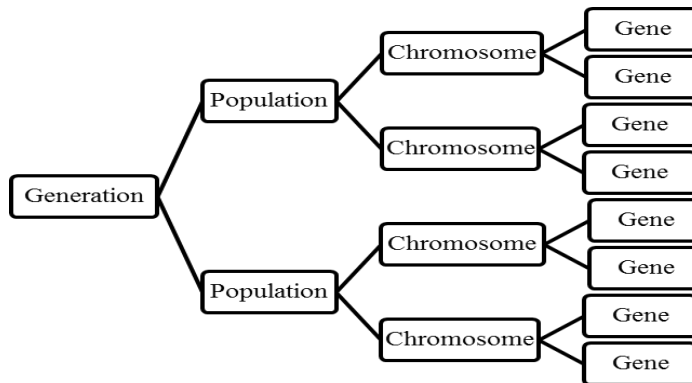


Figure 7:Natural Evolution with GA[3]

3.1 Fundamental Models

Genetic Algorithms borrow terminology from natural genetics. A group of individuals is referred to as a population. Each individual has two forms of representation: genotype and phenotype. The following table provides a list of common terms and their corresponding expressions used in genetics.

Table 1:Contrast of Natural Evolution with Genetic Algorithm[3]

Natural Evolution	Genetic Algorithm
Phenotype	Uncoded point
Genotype	Coded string
Gene	String position
Fitness	Objective function value
Allele	Value at a certain position
Chromosome	String

The phenotype straight represents a possible solution to the problem as it was primarily defined. Conversely, the genotype suggests an encoded form of this solution, designed as a chromosome. A chromosome consists of genes, which are linear sequences of characters, each gene influencing the inheritance of specific traits or features. In some literature, chromosomes are also called individuals. For example, a chromosome might consist of a sequence of 0s and 1s (a bit string), where the value at a specific position indicates whether a particular feature is active (value = 1) or inactive

(value = 0). Depending on the target problem, more complex forms, such as sequences of symbols or permutations of alphabets, may be used for chromosomes.

Each individual has a fitness level, which assesses how well it is modified to its environment. In line with Darwinian [14] theory, the individual most suitable to its environment within a population is more expected to last and reproduce, a concept known as "survival of the fittest." Within this context, the objective function of the optimization problem signifies the environment, while an individual's fitness reflects how efficiently the matching potential solution meets the original optimization standards. When the goal of optimization is to maximize the objective function, fitness may be directly equivalent to the objective function value. $F(x) = f(x)$, here x is the individual in the existing population P (Set of individuals or possible solutions

3.2 A Simple GA

The evolutionary process within a GA begins with an initial population, which is typically generated randomly or through heuristic methods to ensure diversity. Every individual in the population is assessed using a fitness function (i.e. Makespan, Due Date, Energy Consumption etc), which quantifies how fine it solves the problem at hand. This fitness evaluation is pivotal, as it directly influences the selection process. For reproduction, the individuals with higher fitness scores are more expected to be selected, reflecting the principle of "survival of the fittest."

The selection process can employ various techniques, such as

- Roulette wheel selection,
- Tournament selection, or
- Rank-based selection,

to choose individuals that will contribute to the next generation.

Roulette Wheel Selection

Roulette wheel selection, or else fitness balanced selection, allocates each individual a segment of a virtual roulette wheel proportional to their fitness score.

A random number determines which individual is selected based on their fitness proportion. Advanced fitness individuals have a greater chance of being chosen. This method can lead to dominance by the fittest individuals, reducing genetic diversity and risking premature convergence.

Tournament Selection

Tournament selection includes randomly choosing a subset of individuals and selecting the one with the highest fitness among them.

A predefined number of individuals (the tournament size) are randomly selected. The fittest individual within this subset is chosen for the reproduction. This process is iterative until the selection of desired individuals. Larger tournaments favour fitter individuals, while smaller ones maintain more diversity.

Rank-Based Selection

Rank-based selection categories individuals by their suitability and assigns selection probabilities based on their ranks instead of raw fitness scores.

Selection probabilities are assigned based on these ranks, ensuring a balanced and equitable selection process. This prevents domination by highly fit individuals, maintaining genetic diversity but requiring additional computational effort.

3.3 Genetic Operators (Crossover and Mutation)

Reproduction in Genetic Algorithms (GAs) is a crucial phase where genetic operators—specifically crossover (recombination) and mutation play pivotal roles in generating new candidate solutions from existing ones. Here's a more detailed explanation of each operator

3.3.1. Crossover (Recombination)

Crossover is known as a genetic operator used to generate new offspring by combining the genetic information of two parent chromosomes. This process mimics biological reproduction, where offspring inherit traits from both parents, thereby promoting genetic diversity within the population. From the population pool, Two parent chromosomes get selected typically based on their fitness. A position within the chromosome is randomly chosen as the crossover point. The segments of the parent chromosomes beyond this point are

swapped to create two new offspring.

Consider two binary-encoded chromosomes:

- Parent 1: 11010
- Parent 2: 00101

If the crossover point is afterward the third gene, the new offspring generated would be:

- Offspring 1: 110|01
- Offspring 2: 001|10

There are several types of crossover methods,

1. Single-Point Crossover

In single-point crossover, 1 crossover point is nominated on the parent chromosomes. All data outside that point in either chromosome is exchanged between the 2 parent chromosomes. This results in two offspring, each carrying genes from both parents.

Example:

- Parent 1: 110|110
- Parent 2: 001|001

Crossover point after the third gene.

- Offspring 1: 110|001
- Offspring 2: 001|110

2. Two-Point Crossover

Two-point crossover involves two places being chosen on the parent chromosomes. The genes between these two points are exchanged between the parent creatures, creating more diversity than single-point.

Example:

- Parent 1: 110|110|1
- Parent 2: 001|001|0

Crossover points between the third and sixth genes.

- Offspring 1: 110|001|1
- Offspring 2: 001|110|0

3. Uniform Crossover

In uniform crossover, each gene is considered separately. For each gene, a coin is flipped to decide whether or not it will be swapped. This type of crossover does not rely on a specific

segment of the chromosome, but rather mixes the genes at a more granular level.

Example:

- Parent 1: 110110
- Parent 2: 001001

Offspring might have genes from either parent at each position based on a random coin flip.

3.3.2. Mutation

Mutation presents random deviations to individual genes of a chromosome, serving to prevent the genetic search from becoming too homogeneous and potentially trapped in local optima. It ensures the investigation of the genetic search space by introducing new genetic variations.

A chromosome is chosen, often randomly or based on a particular strategy. One or more positions in the chromosome are selected randomly. The genes at these positions are altered. In a binary encoding, this typically means flipping a bit (changing a 1 to a 0, or vice versa).

Consider a binary-encoded chromosome:

- Original: 11010

Suppose the mutation occurs at the second and fifth positions:

- Mutated: 10011 (the second gene is flipped from 1 to 0 and the fifth gene from 0 to 1)

There are several types of mutation techniques, each suitable for different kinds of encoding and problem requirements.

1. Swap Mutation

This mutation type contains selecting two genes at random and exchanging their locations. It is commonly used in permutation-based encodings, such as those found in scheduling and routing problems.

Example:

- Original chromosome (sequence): [1, 2, 3, 4, 5]
- After mutation (swapping positions of 2 and 5): [1, 5, 3, 4, 2]

2. Bit Flip Mutation

This is the most common type of mutation used with binary encoded chromosomes. In

bit flip mutation, each bit in a chromosome has a small chance to be changed from 0 to 1 or from 1 to 0).

Example:

- Original chromosome: 1011001
- After mutation (flipping the third and sixth bits): 1001000

3. Random Resetting

Random resetting is a mutation technique used for integer or real-valued chromosomes. It involves selecting a gene at random and setting it to a random value inside the possible array of values.

Example:

- Original chromosome: [4, 12, 7, 9]
- After mutation (resetting the second value): [4, 6, 7, 9]

Through these iterative processes of selection, crossover, and mutation, populations evolve over successive generations. Each new generation ideally brings the population closer to optimal or near-optimal solutions, as beneficial traits are propagated and refined. The series of evaluation, selection, reproduction, and replacement continues till a end point already fixed is met, such as achieving a satisfactory fitness level or reaching a predefined number of generations.

This iterative and adaptive process allows genetic algorithms to effectively search complex and vast solution spaces, they are the powerful tools for solving many optimization problems across various domains.

Conditions for Ending can be:

- A predetermined number of generations has been completed.
- An individual meets the minimum required criteria.
- The top individual's fitness has reached a level where further iterations no longer yield improvements.
- Manual review or inspection.
- May require the ability to pause and resume

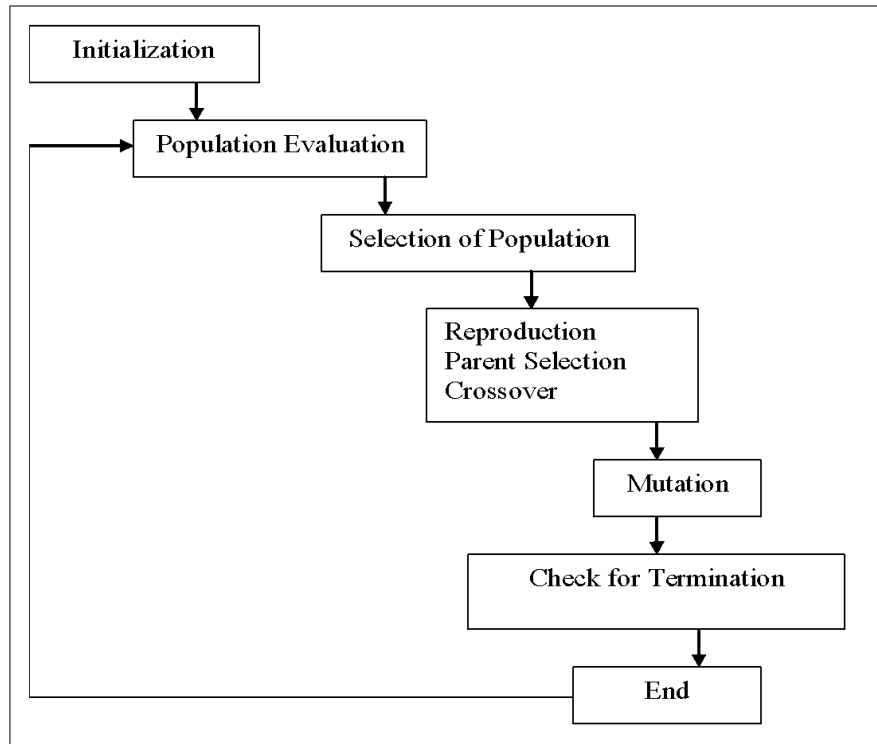


Figure 8: Simple GA search process[1]

3.4 Local Optima

Local optima refer to solutions that are best within a definite neighbourhood of the search space but might not be the globally best solution for the whole problem. These solutions can appear when the algorithm converges prematurely to a suboptimal solution due to the nature of the fitness landscape.

In genetic algorithms (GA), local optima typically arise when the selection pressures and genetic operators (such as crossover and mutation) favor certain regions of the solution space over others. As the algorithm progresses, individuals with advanced fitness in the current population may dominate, leading to a concentration around these local optima and reducing exploration of potentially better solutions elsewhere in the space.

Several strategies can help mitigate the issue of local optima in genetic algorithms:

- **Diversity Maintenance:** Introduce mechanisms to maintain genetic diversity within the population. This can include using diverse selection methods (e.g., tournament selection with varied tournament sizes), promoting mutation rates, or implementing elitism to preserve the best individuals across generations.

- **Adaptive Parameters:** Implement adaptive techniques for adjusting parameters such as mutation rates, crossover probabilities, or population size dynamically during the algorithm's execution. This helps to balance exploration (diversification) and exploitation (intensification) of the solution space.
- **Advanced Selection Mechanisms:** Use advanced selection mechanisms like niching methods or fitness sharing, which encourage exploration by penalizing solutions in crowded regions of the search space. These methods promote the discovery of diverse and potentially better solutions beyond local optima.
- **Hybrid Approaches:** Associate genetic algorithms with other optimization methods, such as local search heuristics or metaheuristics like simulated annealing or tabu search. Hybrid approaches can leverage the strengths of different methods to overcome local optima and improve overall solution quality.

3.5 Development of a Genetic Algorithm for Job Shop Scheduling

The core of my thesis is centered on the development of an advanced genetic algorithm GA incorporating Python (GAIP) specifically tailored to address the complexities inherent in job shop scheduling problems. This algorithm seeks to optimize scheduling efficiency by strategically minimizing the makespan. The makespan, defined as the whole time essential to complete all jobs, serves as the fitness function for the genetic algorithm. This choice for this fitness function ensures that the GA focuses on solutions that reduce the total operational time or makespan, thereby enhancing overall productivity. The makespan serves as the fitness function for the genetic algorithm, meaning that potential solutions are evaluated based on how effectively they reduce this total operational time. The genetic algorithm get started by generating a pool of the initial population of potential solutions, where each solution represents a unique sequence of job operations across multiple machines.

To identify promising solutions, two **selection methods** (Tournament and Roulette wheel selection) are employed simultaneously. This method selects individuals from the population based on their initial makespan, favoring those with shorter makespans. This ensures that the most efficient solutions are retained and used to produce the next generation. The genetic algorithm then applies genetic operators (crossover and mutation) to these selected individuals. These genetic operations introduce variation and help explore new

potential solutions within the search space.

Over successive generations, the algorithm iteratively evaluates and selects the best-performing individuals. By consistently focusing on solutions that minimize the makespan, the GA gradually converges towards an optimal or near-optimal solution. This iterative refinement process ensures that the algorithm continuously improves scheduling sequences, enhancing overall productivity and operational efficiency.

Tournament selection and Roulette wheel selection (Selection methods) are crucial in the genetic algorithm for choosing individuals for crossover. In this method, a subset of the population (a tournament) is randomly selected, and the individual with the best fitness in this subset is chosen. This is repeated until the desired number of individuals is selected.

In my implementation, the tournament size is set to 5. This size strikes a balance between selecting the fittest individuals, which speeds up convergence, and maintaining genetic diversity, which prevents premature convergence. Increasing the tournament size would favor fitter individuals more, accelerating convergence but reducing diversity. Decreasing the size would enhance diversity but slow down convergence. Setting the size to 5 ensures a balanced approach for effective optimization. If the selection probability is more than 0.8, then a roulette wheel selection method will be applied.

3.5.1 Genetic Operators

For recombination, a two-point crossover method is employed. This technique involves selecting two random points within the chromosome structure of the parents to define a segment of genes that will be swapped between them, generating new offspring.

The decision to execute the crossover operation is governed by a probability P_c , which is dynamically determined constructed on the fitness standards of the parents involved in the mating process. Specifically, P_c is calculated as follows [3]:

$$P_c = \begin{cases} K_1 & \text{if } f_1 \geq f_{av} \\ K_2 & \text{if } f_1 < f_{av} \end{cases}$$

where f_1 is the higher fitness value of the two parents, and f_{av} is the average fitness value of the population. The constants k_1 and k_2 are parameters that can be tuned based on experimental findings to optimize the algorithm's performance. In scenarios where the fitness of a parent is above the population average, a higher probability K_1 encourages the retention

of beneficial traits, while k_2 ensures diversity by facilitating crossover even when parent fitness is below average.

In my implementation, the crossover probability was set to 0.8. This relatively high probability was chosen to intensively explore the genetic landscape of the population by frequently mixing genetic material, which is crucial for avoiding local optima and ensuring diverse genetic combinations.

This selection of crossover rate is open but DeJong and Grefenstette have introduced some guidelines for the selection of crossover rate as well as mutation rate.

Crossover rate = 0.6 (For large population size; 100)

Crossover rate = 0.9 (For small population size; 30)

Based on this guideline

Crossover rates k_1 and k_2 were selected as follows:

Crossover Rate $k_1 = 0.90$

Crossover Rate $k_2 = 0.85$

Alongside crossover, swap mutation is implemented as the mutation strategy. In swap mutation, two genes within a chromosome are selected randomly and their positions are swapped. This mutation type is mainly suited for scheduling problems as it subtly alters the sequence of operations, allowing the exploration of new schedules that might yield a shorter makespan without drastically changing the overall structure of the solution.

Mutation occurs with a probability P_m [3], defined as:

$$P_m = \begin{cases} K_3 & \text{if } f_2 \geq f_{av} \\ K_4 & \text{if } f_2 < f_{av} \end{cases}$$

where f_2 is the fitness value of the mutating chromosome. The constants k_3 and k_4 are used to modulate the mutation rate depending on whether the fitness of the chromosome is above or below the population average, facilitating a balance between preserving beneficial genetic material and introducing variability.

In my genetic algorithm, the mutation rate was set to a more moderate value of 0.1. Setting the mutation rate at 0.1 strikes a balance between introducing necessary genetic variations and maintaining the integrity of advantageous traits within the population. This moderate rate ensures that the algorithm does not overly perturb the solution space, thus

allowing a more focused and effective convergence on optimal solutions. It serves to occasionally escape local minima by subtle shifts in the genetic structure, thereby enriching the search process without compromising the established good performance of existing solutions.

Guidelines by DeJong and Grefenstette for the selection of mutation rate are as follows:

Mutation rate = 0.001 (For large population size; 100)

Mutation rate = 0.01 (For small population size; 30)

Based on this guideline Mutation rates k_3 and k_4 were selected as follows:

Crossover Rate $k_3 = 0.2$

Crossover Rate $k_4 = 0.1$

3.5.2 Local search method:

To enhance the refinement of solutions, a neighborhood search method is utilized. This approach involves iterating over each job and machine pairing within the schedule, making incremental adjustments to discover configurations that yield a reduced makespan. Unlike Tabu search, which maintains a list of recently visited solutions to prevent cycling back to them, the neighborhood search method focuses on exploring new areas in the solution space without such a list. This method complements the genetic operations by providing an extra layer of optimization, significantly improving the algorithm's overall effectiveness in finding optimal solutions.

3.5.3 Evolutionary Process

The evolutionary process of the GA involves iteratively applying the selection, crossover, and mutation operations to evolve the population over multiple generations. The fitness of the population is monitored in each generation, with the best solutions being carried forward through elitism.

Elitism: This approach focuses on preserving number of the fittest chromosomes, while the remainder of the population is produced using any of the chosen selection methods. This technique guarantees that the best solutions persist in the population, while also promoting diversity by selecting chromosomes from across the whole solution space. This guarantees that the best solutions found so far are conserved and not lost in subsequent generations.

The parameters for the genetic algorithm are carefully chosen to balance exploration and exploitation. The population size, number of generations, crossover probability, and mutation probability are set to values that provide a decent trade-off among finding high-quality solutions and computational effectiveness.

3.5.4 Experimental Setup and benchmarks

The experimental setup for the genetic algorithm involves running the algorithm on benchmark problems, such as MT06, MT10 and MT20, as well as custom scheduling problems. The data for benchmarks was taken from [OR Library](#) In other words, the developed genetic algorithm GA is tested against the benchmarks to test how accurate the genetic algorithm is and how well genetic operators are doing their job. Everything is done in the PYTHON (Programming language) and it is named as genetic algorithm incorporated Python (GAIP) . Below is experimental setup:

Table 2: Experimental setup

Operator/Parameter	Name/Value
Population size	1000 (user can change)
Elitism	5
Selection operator	Tournament/Roulette wheel
Tournament size	5
Crossover operator	Two-point
Mutation operator	swap
Iterations	100 (User can change)
Crossover probability	0.8
Mutation probability	0.1

Benchmark problems are standardized instances or datasets used to evaluate the performance of algorithms, including genetic algorithms (GAs) in job shop scheduling. These benchmarks serve several purposes:

- **Standardization:** They provide a common ground for comparing different algorithms objectively. By using the same set of problems, researchers can assess which algorithm performs better under similar conditions.

- **Complexity Variation:** Benchmarks come in various sizes and complexities, ranging from small to large-scale problems with different numbers of jobs and machines. This variation helps in understanding algorithm scalability and robustness.
- **Real-world Relevance:** Many benchmarks are derived from real-world scheduling problems, ensuring that algorithmic solutions can be applied effectively in practical scenarios.
- **Algorithm Testing:** They allow researchers to test algorithm efficiency, accuracy, and scalability under controlled conditions, providing insights into strengths and weaknesses.

MT06, MT10 and MT20 are specific benchmark datasets commonly used in research on job shop scheduling problems. They are designed to simulate different scheduling scenarios with varying complexities:

MT06 (6 Jobs, 6 Machines):

- MT06 contains instances where there are 6 jobs and 6 machines.
- Respectively job requires processing on respectively machine in a predefined sequence with specific processing times.
- The goal is to schedule these jobs on machines to lessen the makespan, which is the total time mandatory to complete all jobs.

Table 3:MT06 benchmark

Process	1		2		3		4		5		6	
Jobs	m	t	m	t	m	t	m	t	m	t	m	T
1	3	1	1	3	2	6	4	7	6	3	5	6
2	2	8	3	5	5	10	6	10	1	10	4	4
3	3	5	4	4	6	8	1	9	2	1	5	7
4	2	5	1	5	3	5	4	3	5	8	6	9
5	3	9	2	3	5	5	6	4	1	3	4	1
6	2	3	4	3	6	9	1	10	5	4	3	1

MT10 (10 Jobs, 10 Machines):

- MT10 involves instances with 10 jobs and 10 machines.
- Similar to MT06, each job needs to be handled on each machine following a given sequence and processing times.

- The objective remains the same: to find an best schedule that minimizes the makespan.

Table 4: MT10 Benchmark

Process	1		2		3		4		5		6		7		8		9		10	
Jobs	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t
1	1	29	2	78	3	9	4	36	5	49	6	11	7	62	8	56	9	44	10	21
2	1	43	3	90	5	75	10	11	4	69	2	28	7	46	6	46	8	72	9	30
3	2	91	1	85	4	39	3	74	9	90	6	10	8	12	7	89	10	45	5	33
4	2	81	3	95	1	71	5	99	7	9	9	52	8	85	4	98	10	22	6	43
5	3	14	1	6	2	22	6	61	4	26	5	69	9	21	8	49	10	72	7	53
6	3	84	2	2	6	52	4	95	9	48	10	72	1	47	7	65	5	6	8	25
7	2	46	1	37	4	61	3	13	7	32	6	21	10	32	9	89	8	30	5	55
8	3	31	1	86	2	46	6	74	5	32	7	88	9	19	10	48	8	36	4	79
9	1	76	2	69	4	76	6	51	3	85	10	11	7	40	8	89	5	26	9	74
10	2	85	1	13	3	61	7	7	9	64	10	76	6	47	4	52	5	90	8	45

MT20 (20 Jobs, 5 Machines)

- MT20 involves problems with 20 jobs and 5 machines.
- Similar to MT06 and MT10, apiece job needs to be processed on each machine following a given sequence and processing times.
- The objective remains the same: to find an ideal schedule that minimizes the makespan

Table 4: MT10 Benchmark

Process	1		2		3		4		5	
Jobs	m	t	m	t	m	t	m	t	m	t
1	1	29	2	9	3	49	4	62	5	44
2	1	43	2	75	4	69	3	46	5	72
3	2	91	1	39	3	90	5	12	4	45
4	2	81	1	71	5	9	3	85	4	22
5	3	14	2	22	1	26	4	21	5	72
6	3	84	2	52	5	48	1	47	4	6
7	2	46	1	61	3	32	4	32	5	30
8	3	31	2	46	1	32	4	19	5	36
9	1	76	4	76	3	85	2	40	5	26
10	2	85	3	61	1	64	4	47	5	90
11	2	78	4	36	1	11	5	56	3	21
12	3	90	1	11	2	28	4	46	5	30
13	1	85	3	74	2	10	4	89	5	33
14	3	95	1	99	2	52	4	98	5	43
15	1	6	2	61	5	69	3	49	4	53
16	2	2	1	95	4	72	5	65	3	25
17	1	37	3	13	2	21	4	89	5	55
18	1	86	2	74	5	88	3	48	4	79
19	2	69	3	51	1	11	4	89	5	74
20	1	13	2	7	3	76	4	52	5	45

These benchmarks are widely used in the research community to assess the effectiveness and efficiency of scheduling algorithms, including genetic algorithms. Researchers often test their algorithms on these benchmarks to compare performance against other methods and to demonstrate the scalability and robustness of their approaches in handling different problem sizes and complexities.

3.5.5 Basic Flowchart for GA

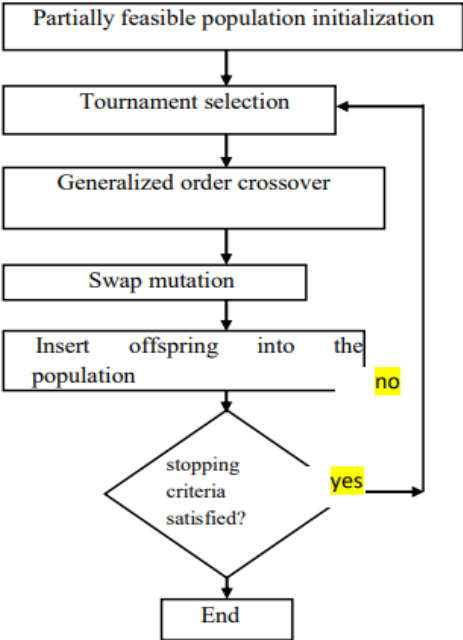


Figure 9:Basic Flowchart for GA[1]

3.6 Manual Example

Jobs (J): J0, J1, J2, J3

Machines (M): M0, M1, M2, M3

Processing Times Matrix:

Table 5:Processing Times for Manual Problem

Job \ Machine	M0	M1	M2	M3
J0	7	2	5	3
J1	1	9	3	5
J2	4	3	6	2
J3	2	6	4	8

Step 1: Initialization

Generate an initial population of four chromosomes, each representing a different machine order for the jobs.

Table 6:Initial Population

Chromosome	Job Orders (J0 to J3)
C1	[0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3]
C2	[0, 2, 1, 3], [1, 0, 3, 2], [1, 2, 0, 3], [2, 0, 3, 1]
C3	[1, 0, 3, 2], [2, 1, 0, 3], [0, 3, 2, 1], [1, 2, 3, 0]
C4	[2, 3, 1, 0], [3, 1, 0, 2], [1, 0, 3, 2], [0, 2, 1, 3]

Step 2: Calculate Initial Makespan

The makespan for each chromosome is calculated by simulating the execution of the jobs on the workstations according to the specified demand. Here, I will detail the makespan calculation for **Chromosome 1 (C1)** as an example.

Makespan Calculation for Chromosome 1 (C1):

Using a table to show the start and finish times for each job at each machine

Table 7:Start and Finish times for each job

Job \ Machine	M0	M1	M2	M3
J0	0-7	7-9	9-14	14-17
J1	7-8	9-18	18-21	21-26
J2	8-12	18-21	21-27	27-29
J3	12-14	21-27	27-31	31-39

Lets proof whether this makespan is correct by using ‘initial makespan calculator’ which is designed to find the makespan for various problems.

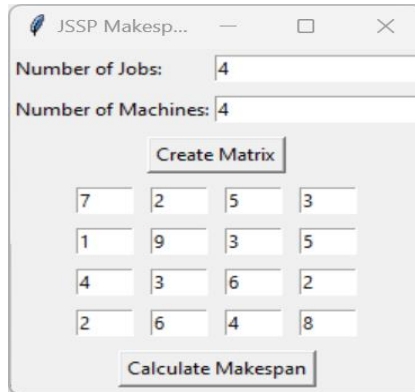


Figure 9: Initial Maespan Calc

Result Message:

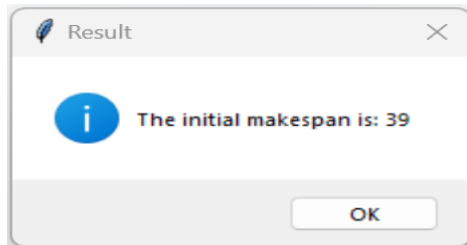


Figure 10: Result Message

So, it shows that the initial makespan is 39 and it's correct. Initial Makespan for C1: 39

Step 3: Selection for Crossover

Select the two best chromosomes based on their makespans. Suppose C1 and C2 are selected for their shorter makespans.

Step 4: Crossover

Apply a two-point crossover between C1 and C2:

Crossover Point: After the first job.

Table 8: New Chromosomes After Crossover

Resulting Chromosome	Job Orders After Crossover
C1'	C1's first job, C2's other jobs
C2'	C2's first job, C1's other jobs

Resulting Chromosome (C1’): The first job sequence was retained from C1, and the sequences for the remaining jobs were taken from C2, resulting in a new chromosome: [[0, 1, 2, 3], [1, 0, 3, 2], [1, 2, 0, 3], [2, 0, 3, 1]].

Step 5: Mutation

Randomly mutate C1’ by swapping two machines in J2’s order:

Table 9:Mutation

Chromosome	Job Orders Before Mutation	Job Orders After Mutation
C1'	[0, 1, 2, 3] in J2	[0, 2, 1, 3] in J2 (swap M1 and M2)

Post-Mutation Chromosome (C1’): [[0, 1, 2, 3], [1, 0, 3, 2], [1, 0, 2, 3], [2, 0, 3, 1]].

Step 6: Evaluate and Iterate

Calculate the new makespans for the mutated chromosomes. Assume that this leads to an improved makespan of 35 for C1’.

Step 7: Convergence and Final Result

After repeated iterations and genetic operations, the algorithm converges to an optimised solution:

- **Optimized Makespan:** 35
- **Optimal Chromosome (Final C1’):** [[0, 1, 2, 3], [1, 0, 3, 2], [1, 0, 2, 3], [2, 0, 3, 1]]

3.7 Initial Makespan Calculator

In the realm of Job Shop Scheduling (JSS), accurately determining the initial makespan is crucial for benchmarking and comparing the efficacy of scheduling algorithms. To facilitate this, I developed an "Initial Makespan Calculator" implemented in Python using the Tkinter graphical user interface (GUI) framework. This tool enables users to input the number of jobs and machines, along with specific processing times, to calculate the initial makespan quickly and efficiently.

Once hitting run button the software starts, asking user to enter the total number of jobs and machines involved in the scheduling process. Based on these inputs, a dynamic input grid generated where the user can fill in the processing times for each job on each machine. This approach not only ensures flexibility in handling different job shop configurations but also simplifies data entry, making it user-friendly.

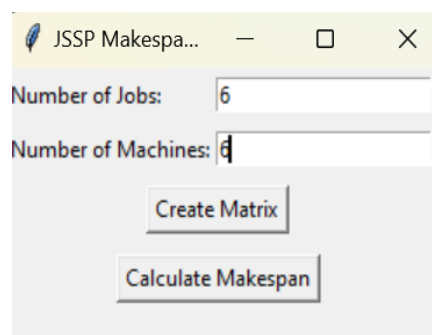
Once the data is inputted, the makespan calculation is triggered by a dedicated button within the GUI. The calculation itself is executed through a function that iterates over each job and machine, determining the earliest possible start time for each task based on the readiness of the machine and the conclusion of the previous tasks. The maximum end time across all machines, which represents the initial makespan, is then displayed to the user in a message box. This immediate feedback is valuable for operations managers and researchers who need to assess and optimize their production schedules.

Moreover, the tool is equipped with error handling to ensure that only valid integers are entered, enhancing its robustness and reliability. This calculator not only serves as a standalone tool for quick makespan assessments but also as a complementary utility for researchers and practitioners who are testing and comparing different scheduling algorithms' performance.

Through this development, I aimed to provide a practical tool that bridges the hole between hypothetical scheduling models and real applications, allowing for a more interactive and accessible approach to understanding and improving job shop scheduling outcomes.

3.7.1 How it works?

Step 1: Enter Job Equipment like Jobs and Machines numbers.



Step 2: Create a Matrix and add Job Info i.e Processing Times for each job

Job \ Machine	1	2	3	4	5	6
1	1	3	6	7	3	6
2	8	5	10	10	10	4
3	5	4	8	9	1	7
4	5	5	5	3	8	9
5	9	3	5	4	3	1
6	3	3	9	10	4	1

Step 3: Click “Calculate Makespan’ to get results.

Result

i The initial makespan is: 66

OK

Chapter # 04: Python Libraries and GUI Development for Scheduling Optimization

Using advanced programming languages like Python in manufacturing is beneficial due to the increasing complication of production arrangements and the need for greater efficiency and customization in manufacturing processes. I used Visual studio code as a platform for Python.

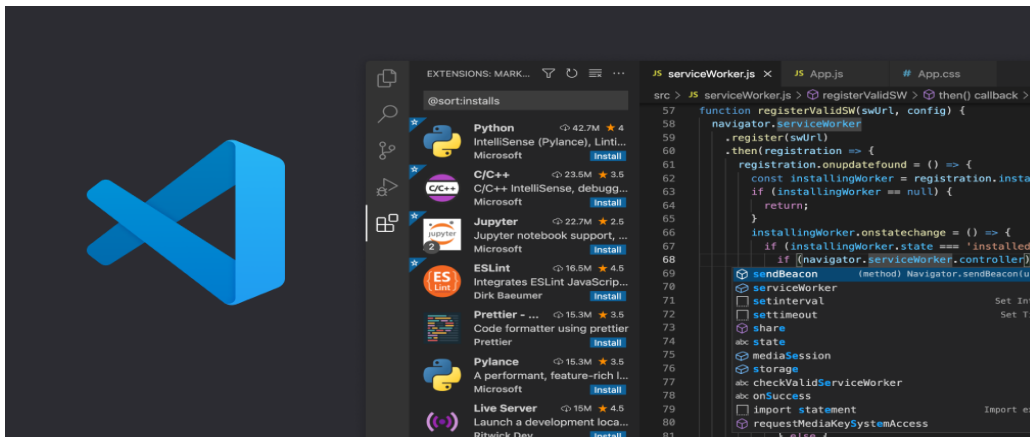


Figure 11: Visual Studio Code

My Python code utilizes several powerful libraries, each playing a significant part in the functioning of genetic algorithm for job shop scheduling optimization. Here's a breakdown of these libraries and their contributions to my project:

4.1 Python Libraries

4.1.1 Random

The random library is essential for implementing stochastic processes within your genetic algorithm. It enables the generation of random numbers, which are crucial for initializing solutions, performing genetic operations like crossover and mutation, and randomly selecting individuals for various genetic operations.

The use of randomness introduces variability and helps in exploring a broader solution space, which is vital for avoiding local minima and enhancing the algorithm's ability to find near-optimal solutions.

4.1.2. Matplotlib (including matplotlib.patches)

Matplotlib is a Python library used for generating interactive visuals in Python. Within your project, it is specifically employed to generate Gantt charts, which are integral for visualizing the scheduling results. The Gantt charts clearly depict the start and end times of different jobs on various machines, allowing for an intuitive understanding of the scheduling efficiency and resource allocation. The matplotlib.patches module is particularly useful for adding shapes (like rectangles for each task in the Gantt chart), enhancing the visual appeal and readability of the charts.

4.1.3. Tkinter (including simpledialog, messagebox, Toplevel, Scrollbar, Frame, Canvas)

Tkinter is Python's standard GUI toolkit and is used extensively in your project to create a user-friendly interface. This toolkit allows you to build windows with buttons, dialogs, and other graphical elements, making the algorithm accessible to users who may not be familiar with command-line interactions. simpledialog is used for inputting parameters like population size and number of iterations, messagebox for displaying alerts and information, Toplevel for creating additional windows, Scrollbar for adding scrolling capability, Frame for structuring the layout, and Canvas for more complex graphical tasks like drawing the Gantt chart within the GUI.

4.1.4. Matplotlib.backends.backend_tkagg (FigureCanvasTkAgg)

This module integrates Matplotlib with Tkinter by providing a specific backend to handle rendering figures in a Tkinter canvas. FigureCanvasTkAgg takes a Matplotlib figure and converts it into a format that can be used as a Tkinter widget. This integration is crucial for embedding Matplotlib graphs (such as your Gantt charts) directly into the graphical interface, providing a seamless user experience.

In this project, Python's versatile capabilities were leveraged to design a user-friendly graphical user interface (GUI), allowing users to seamlessly interact with the genetic algorithm for job shop scheduling. This interface provides intuitive controls and visual feedback, making the complex process of scheduling optimization accessible and manageable for users regardless of their technical expertise.

4.2 Interactive Flow Chart: Navigating the GUI-User Experience

A flow chart is shown for better understanding of how users can interact with algorithm.

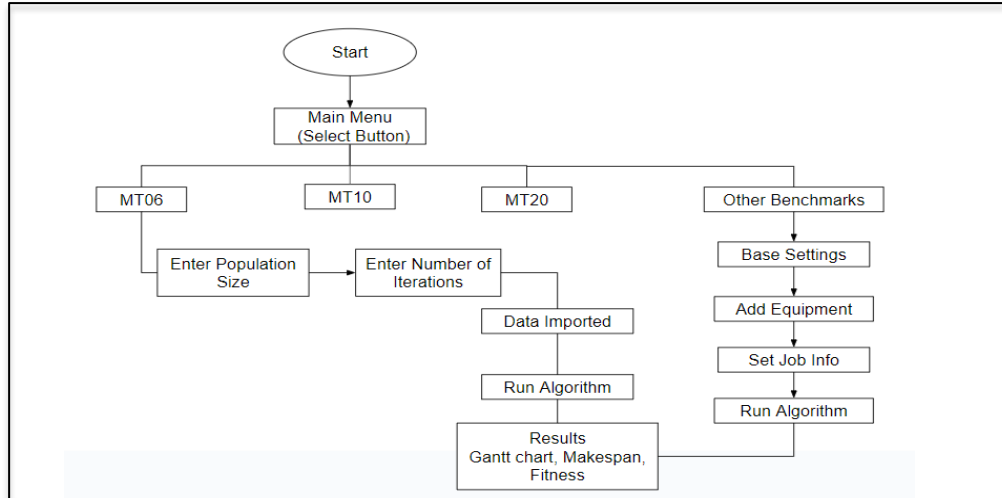


Figure 12: Interactive Flow Chart: Navigating the GUI-User Experience

4.2.1 Main Menu:

The main menu of the job shop scheduling application serves as the central hub for user interaction, providing a straightforward and intuitive interface that guides users through various functionalities of the system.

When the application is launched, the main menu is the first screen presented to the user. It serves as the gateway to all the functionalities of the application.

Step 1: The user clicks on the 'Import Data' button.

Step 2: A submenu appears, offering choices between MT06, MT10, and MT20 datasets. These benchmarks are already added in the code for user ease.

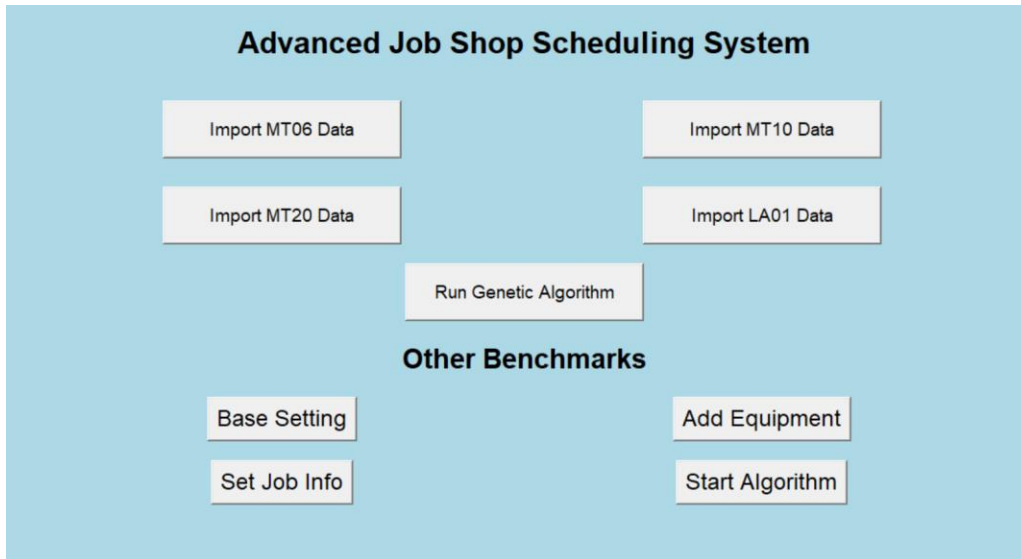
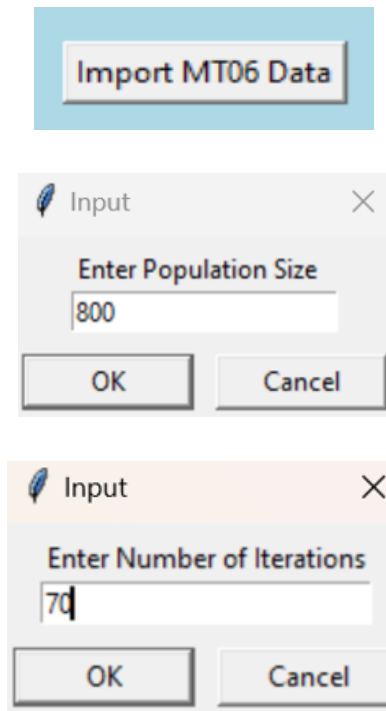
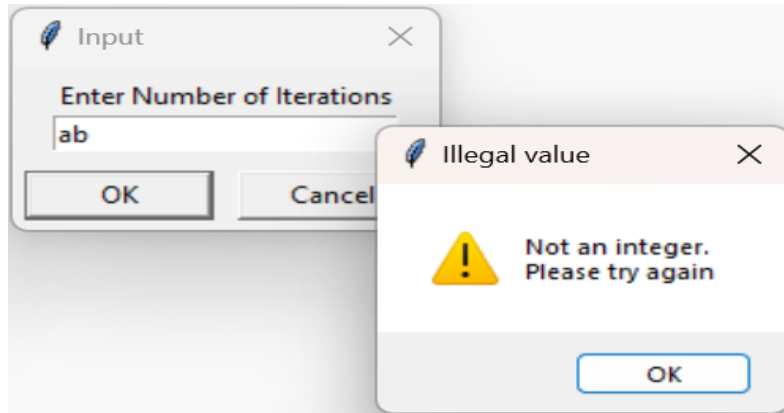


Figure 13: Main Menu for JSSP

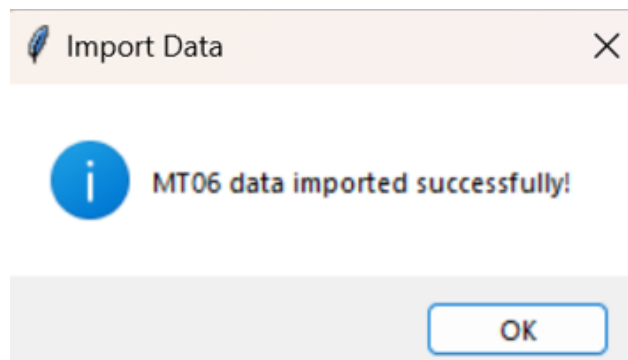
Step 3: The user selects the desired dataset. A new window appears asking for “Population Size”. Once Population is entered, it will lead to a new window asking for “Number of Iterations”.



Note: The data will be entered only in numeric form. If a user tries to enter string or alphabets, the system will generate an error to guide the user.



Step 4: Once all information is added, a message will appear saying “Data Imported Successfully”. Which means that system has successfully imported benchmarks and all parameters from the code.



Step 5: After this message, the user needs to hit the “Run genetic algorithm” button to start the algorithm.

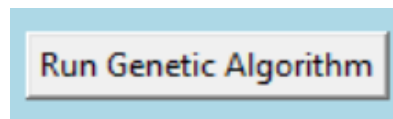


Figure 14: Run Algorithm for MT06

Step 6: After execution of algorithm, a window will appear with “Final Makespan” which is an optimized makespan much smaller than initial makespan, a Gantt chart representing the Job and machine schedule and fitness graph.

The **Gantt chart** is utilized to visualize the scheduling optimization performed by the genetic algorithm. Each bar on the Gantt chart represents a specific job being executed on a particular machine, the bar length showing the job duration.. The chart

effectively demonstrates how jobs are allocated to machines over time, helping identify bottlenecks or inefficiencies in the schedule. It allows users to quickly assess the makespan of the entire production process, which is crucial for optimizing operational efficiency in manufacturing environments.

The **fitness graph** is used to display the improvement in the algorithm's performance over time. Each point on the graph represents the fitness score of the best solution found in that generation, allowing users to visually track how the solution quality evolves as the algorithm runs. This is particularly important for diagnosing the behavior of the genetic algorithm, such as determining if and when it has converged to a solution or if it is stuck in a local optimum. The fitness graph helps in validating the effectiveness of the algorithmic tweaks and settings adjustments made through the GUI.

4.2.2 Other Benchmarks

The 'Other Benchmark' feature in the graphical user interface (GUI) of the job shop scheduling application is designed to enhance the flexibility and usability of the system. Its primary purpose is to allow users to load custom datasets or benchmarks that are not included in the predefined options (like MT06, MT10, or MT20). This feature caters to users who need to test the algorithm with specific scheduling scenarios or data reflective of their unique operational environments. It not only broadens the application's applicability across different operational and educational contexts but also empowers users by providing them the tools needed to customize their experience and results. This feature is a critical component in ensuring that the application can meet the evolving needs of end users, to foster a continuous environment of learning and improvement.

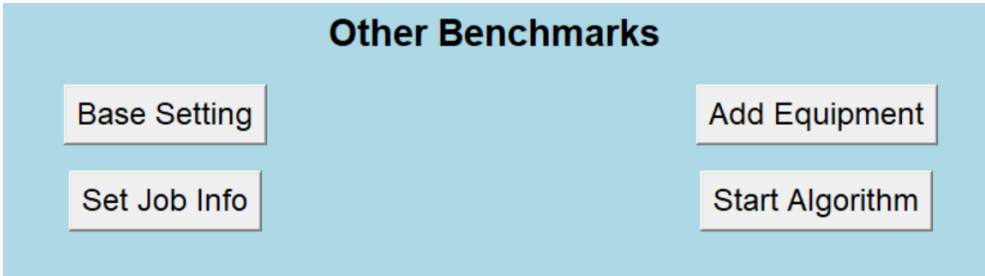
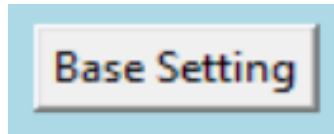
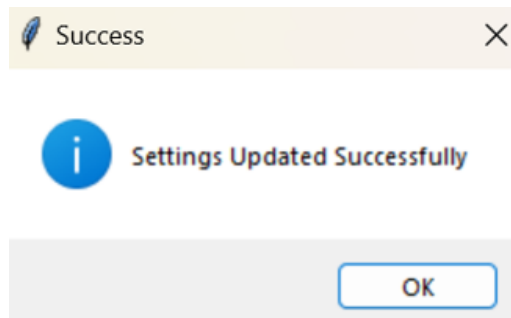


Figure 15: Other Benchmarks

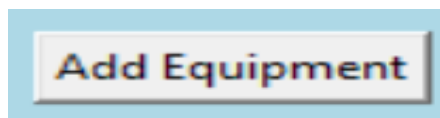
Step 1: The first step is “base settings”.



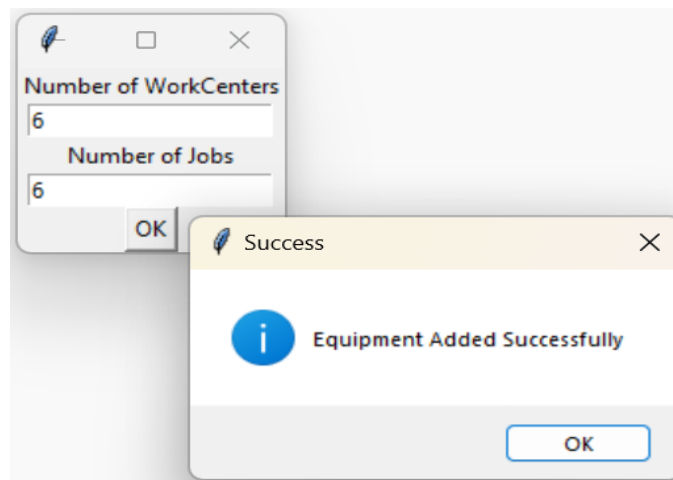
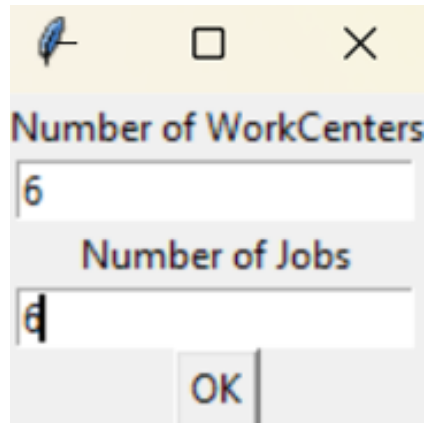
Once user hits this button, it will lead to a window asking for “Population Size” and “Number of Iterations”.

A small dialog box with a yellow header bar containing a feather icon, a maximize button, and a close button. Below the header, there are two input fields. The first is labeled "Population" and contains the value "1000". The second is labeled "Generation" and contains the value "100". At the bottom right of the dialog is a "Set" button.

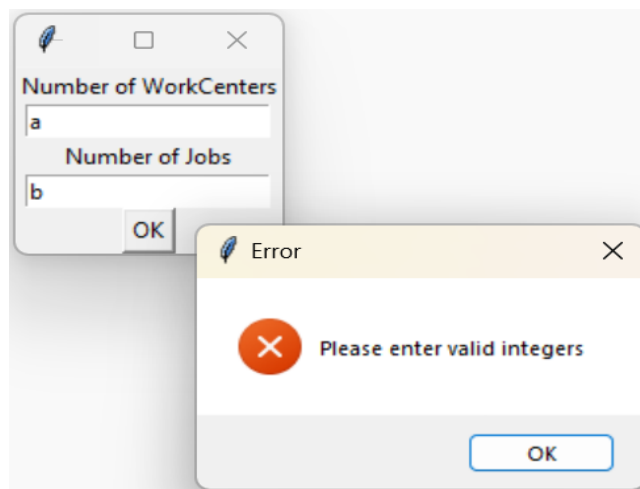
Step 2: Once base setting is done, the next step is “Add Equipment” which means setting the “Number of Jobs” and “Number of machines”.



This is very crucial step because on the basis of number of workcentres/machines and number of jobs , the next window will be created where user will have to give machine (IDs-processing time).



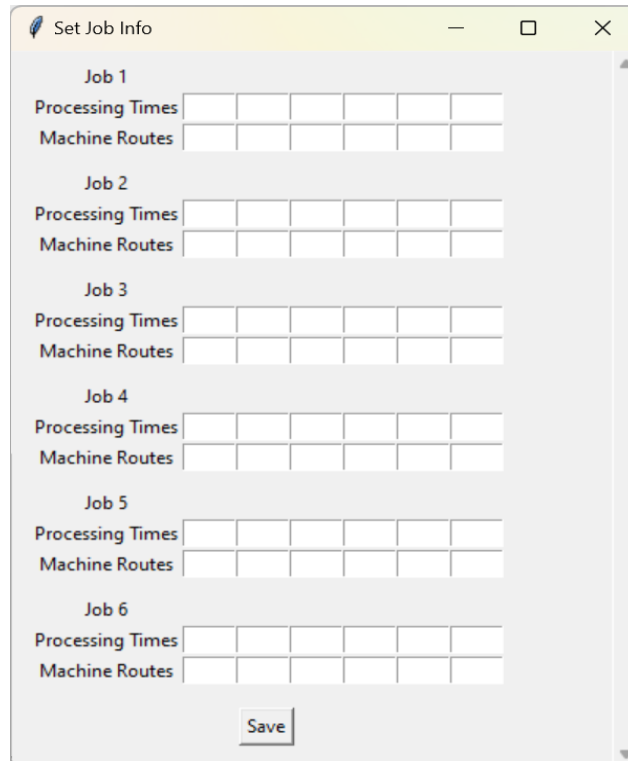
Note: If user fails to give input in numeric form, system will generate an error message.



Step 3: The next step is to “Set Job Info”.



Users need to add Machine IDs in front of Machine row and Times in front of Processing times row below.

A dialog box titled "Set Job Info" with a yellow header bar. It contains six job entries, each with "Processing Times" and "Machine Routes" rows. Each row consists of five empty input boxes. A "Save" button is located at the bottom center of the dialog box.

Job	Processing Times	Machine Routes
Job 1	<input type="text"/>	<input type="text"/>
Job 2	<input type="text"/>	<input type="text"/>
Job 3	<input type="text"/>	<input type="text"/>
Job 4	<input type="text"/>	<input type="text"/>
Job 5	<input type="text"/>	<input type="text"/>
Job 6	<input type="text"/>	<input type="text"/>

Step 4 : Hit “Start Genetic Algorithm” to get optimised makespan, gantt chart and fitness graph.

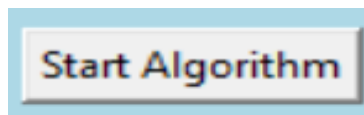


Figure 16: Start Algorithm Button for others

4.3 Importance of Introducing Modern Programming Languages like Python in Manufacturing

My project exemplifies the best of what Python can offer in a manufacturing context. Introducing modern programming languages like Python into manufacturing is essential for several compelling reasons, all of which revolve around enhancing efficiency, adaptability, and innovation within the sector:

Using Python in manufacturing, especially for complex scheduling tasks, brings several advantages:

- **Efficiency and Speed:** Python's extensive libraries and simple syntax allow for rapid development and testing of complex algorithms, significantly speeding up the iterative process of solution enhancement.
- **Frontend Integration**

Python isn't just suitable for writing scripts and automating tasks—it's also a robust language for developing complete applications, especially when combined with various libraries and frameworks tailored to app development. For applications requiring a graphical user interface, Python's **Tkinter** library allows for the development of desktop-based GUIs. For web-based applications, Python can serve data to frontend technologies using templates or through APIs. Libraries like **Jinja2** work well with Flask and Django to generate HTML content dynamically. For more interactive frontends, you can develop RESTful APIs using frameworks like **Django REST framework** or **Flask-RESTful**, which can serve data to frontends built with JavaScript frameworks like React or Angular.
- **Flexibility:** Python can easily integrate with other systems and handle various data formats, making it suitable for the heterogeneous environments typically found in manufacturing.

- **Scalability:** Python's ability to scale with increased data sizes and complexity makes it ideal for industrial applications where vast amounts of data are processed.
- **Accessibility:** With its widespread use and supportive community, Python offers extensive resources for troubleshooting and development, reducing barriers to technology adoption in manufacturing settings.

Chapter 5: GA performance Evaluation

In this chapter, we delve into the detailed evaluation of the genetic algorithm's (GA) performance as applied to job shop scheduling problems within our developed application. The primary focus is on analyzing how effectively the GA optimizes scheduling tasks, its efficiency in navigating all the solutions, and the solutions quality it generates. This assessment is crucial not only for validating the effectiveness of the algorithm but also for identifying potential areas for enhancement.

5.1 Flowchart for GA working

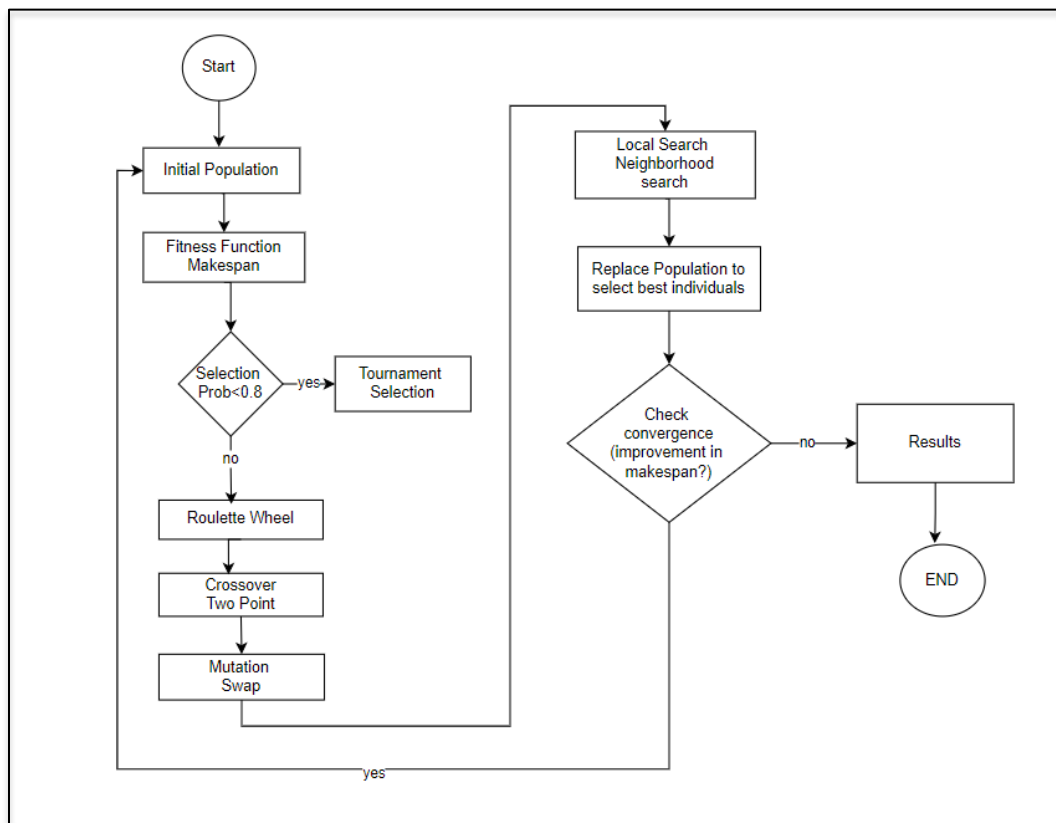


Figure 17: Flowchart for Beckend (GA)

5.2 Complete Python Pseudo Code

```
#Import necessary libraries IMPORT random # For generating
random numbers
IMPORT matplotlib.pyplot AS plt # For plotting charts
```

```

    IMPORT matplotlib.patches AS patches # For drawing shapes
on charts
    IMPORT tkinter AS tk # For creating the GUI window and
elements
    FROM tkinter IMPORT Toplevel, Scrollbar, Frame, Canvas #
For advanced GUI elements FROM tkinter IMPORT simplifiedialog,
messagebox # For dialog boxes and message pop-ups
    FROM matplotlib.backends.backend_tkagg IMPORT
FigureCanvasTkAgg # For embedding matplotlib charts in
tkinter

# Define benchmark data for various job shop scheduling
problems INITIALIZE mt06_data AS 2D Array # Contains
processing times and machine assignments for MT06 benchmark
    INITIALIZE mt10_data AS 2D Array # Contains processing
times and machine assignments for MT10 benchmark
    INITIALIZE mt20_data AS 2D Array # Contains processing
times and machine assignments for MT20 benchmark
    INITIALIZE la01_data AS 2D Array # Contains processing
times and machine assignments for LA01 benchmark

# Function to prompt user for integer input through a
dialog box FUNCTION custom_askinteger(title, prompt,
parent=None):
    CREATE a new dialog window
    SET the title of the dialog window
    DISPLAY the prompt message in the dialog
    ALLOW the user to input an integer value
    VALIDATE the input; if invalid, show an error message
    RETURN the integer value entered by the user

# Function to display an informational message to the user

```

```

FUNCTION custom_showinfo(title, message, parent=None):
    CREATE a new dialog window
    SET the title of the dialog window
    DISPLAY the message to the user
    WAIT for the user to acknowledge the message by
clicking 'OK'

# Function to initialize a population of job schedules
FUNCTION initialize_population(population_size,
num_jobs,num_machines):
    CREATE an empty list to hold the population
    FOR each individual in the population:
        CREATE a random job schedule (sequence of machines
for each job)      ADD the schedule to the population list
    RETURN the list of job schedules (population)

# Function to calculate the makespan (total time) of a
given schedule FUNCTION calculate_makespan(schedule,
processing_times):
    INITIALIZE machine_end_times AS a list of zeros (length
equals number of machines)
    INITIALIZE job_end_times AS a list of zeros (length equals
number of jobs)
    CREATE dictionaries to hold the job and machine schedules

    FOR each job in the schedule:
        FOR each machine in the job's route:
            DETERMINE the start time for the operation as the
maximum of the      machine's available time and the job's
current time
            CALCULATE the end time by adding the processing time

```

```
UPDATE machine_end_times and job_end_times with the
new end time RECORD the operation in job_schedule and
machine_schedule
```

```
RETURN the maximum value in job_end_times as the makespan,
along with the job_schedule and machine_schedule
```

```
# Function to perform crossover between two parent
chromosomes FUNCTION crossover(parent1, parent2, rand_int,
rand_float, avg_fitness, max_fitness):
```

```
SELECT two random positions in the chromosome for
crossover points SWAP the segments between the two
parents to create two new child chromosomes
```

```
RETURN the two child chromosomes
```

```
# Function to mutate chromosomes in the population
FUNCTION mutation(rand_int, rand_float, fitness,
population):
```

```
CREATE an empty list to hold the mutated population
```

```
FOR each chromosome in the population:
```

```
IF a random number is less than the mutation rate:
```

```
SELECT two random positions in the chromosome
```

```
SWAP the genes at these positions
```

```
ADD the (potentially) mutated chromosome to the new
population RETURN the mutated population
```

```
# Function to improve a schedule through local search
```

```
FUNCTION local_search(schedule, processing_times):
```

```
SET the best_schedule AS the current schedule
```

```
CALCULATE the makespan for the current best_schedule
```

```
FOR each job and machine in the schedule:
```

```

        MODIFY the schedule slightly by changing the
machine assignment    CALCULATE the makespan for the
modified schedule

    IF the modified schedule has a better makespan:
        UPDATE best_schedule to the modified schedule
    RETURN the best_schedule found during the search

# Function to perform tournament selection
FUNCTION tournament_selection(population, tournament_size,
processing_times):
    RANDOMLY select a subset of chromosomes from the
population    CALCULATE the fitness (makespan) for each
selected chromosome RETURN the chromosome with the best
fitness

# Function to perform roulette wheel selection
FUNCTION roulette_wheel_selection(population,
fitness_values):    CALCULATE the total fitness of the
population
    PICK a random value between 0 and the total fitness
    LOOP through the population, accumulating fitness, until
the random value is reached
    RETURN the selected chromosome

# Main function to run the genetic algorithm with custom
crossover FUNCTION
genetic_algorithm_with_custom_crossover(num_machines,
num_jobs, processing_times, population_size=1500,
elitism=20, iterations=100, tournament_size=10):
    INITIALIZE the population using the initialize_population
function

```

```

    SET best_solution AS None and best_makespan AS
infinity
    CREATE an empty list to store fitness over time

    FOR each generation (iteration):
    CALCULATE the fitness (makespan) for each chromosome in
the population
    UPDATE the best_solution if a new best_makespan is found
    STORE the current best makespan in the fitness_over_time
list

    CREATE a new population using elitism (preserving the top
solutions) FILL the rest of the population using crossover
and mutation

    APPLY local search to improve each chromosome in the new
population UPDATE the population with the new population

    RETURN the best_solution, best_makespan, and
fitness_over_time

# Function to plot a Gantt chart representing the job
schedule on machines
FUNCTION plot_gantt_chart(job_schedule, machine_schedule,
num_jobs, num_machines, master):
    CREATE a new figure and axis using matplotlib
    FOR each machine in machine_schedule:
        FOR each job operation on the machine:
            DRAW a rectangle representing the operation on the
Gantt chart LABEL the rectangle with the job and
operation number
    SET axis limits and labels

```

```

DISPLAY the Gantt chart in the tkinter window

# Function to plot fitness over generations
FUNCTION plot_fitness_over_time(fitness_over_time,
master):
    CREATE a new figure and axis using matplotlib
    PLOT the fitness values over time
    SET axis labels and title
    DISPLAY the plot in the tkinter window

# Class to handle the graphical user interface (GUI) for
the genetic algorithm
CLASS GeneticAlgorithmGUI:
    FUNCTION __init__(self, master):
        SETUP the main tkinter window with a title and size
        CREATE and layout buttons for importing data, running the
algorithm, and setting job information
        INITIALIZE variables to store job and machine
information

FUNCTION import_mt06_data(self):
    PROMPT the user for population size and generations
    SET the number of jobs and machines for MT06 benchmark
    LOAD the processing times from mt06_data

FUNCTION import_mt20_data(self):
    PROMPT the user for population size and generations
    SET the number of jobs and machines for MT20 benchmark
    LOAD the processing times from mt20_data

FUNCTION import_la01_data(self):
    PROMPT the user for population size and generations

```



```

SET the number of jobs and machines for LA01 benchmark
LOAD the processing times from la01_data

FUNCTION run_algorithm(self):
    IF job information is not set:
        DISPLAY an error message
    ELSE:
        RUN the genetic algorithm using the specified
settings
        DISPLAY the results, including a Gantt chart and
fitness plot

# Main program execution IF __name__ == "__main__":
CREATE a tkinter root window
CREATE an instance of GeneticAlgorithmGUI with the root
window    START the tkinter main event loop

```

5.3 Results for Test problem

Example Problem

Let us consider an example problem with the following specifications, assuming that set up times are included in the processing times:

Table 10: Test Problem

Process	1		2		3		4	
Jobs	m	t	m	t	m	t	m	t
1	1	21	0	53	4	95	3	55
2	0	21	3	52	4	16	2	26
3	3	39	4	98	1	42	2	31
4	1	77	0	55	4	79	2	66

- **Best Obtained Makespan:** 277
- **Final Chromosome:** [[1, 2, 3, 4], [3, 2, 4, 1], [2, 3, 1, 4], [1, 4, 2, 3]]

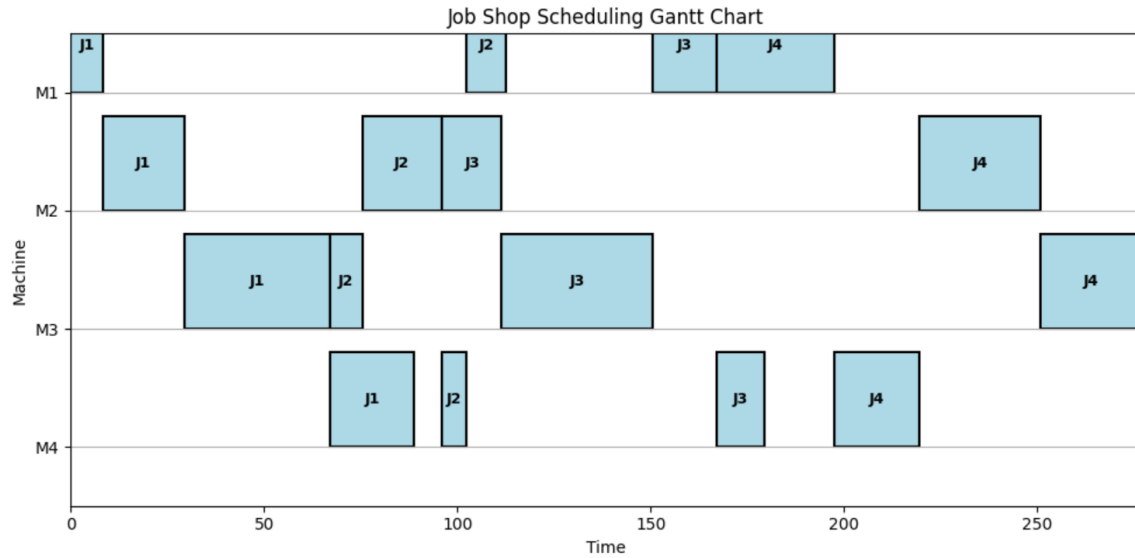


Figure 18: GANTT CHART for Test Problem

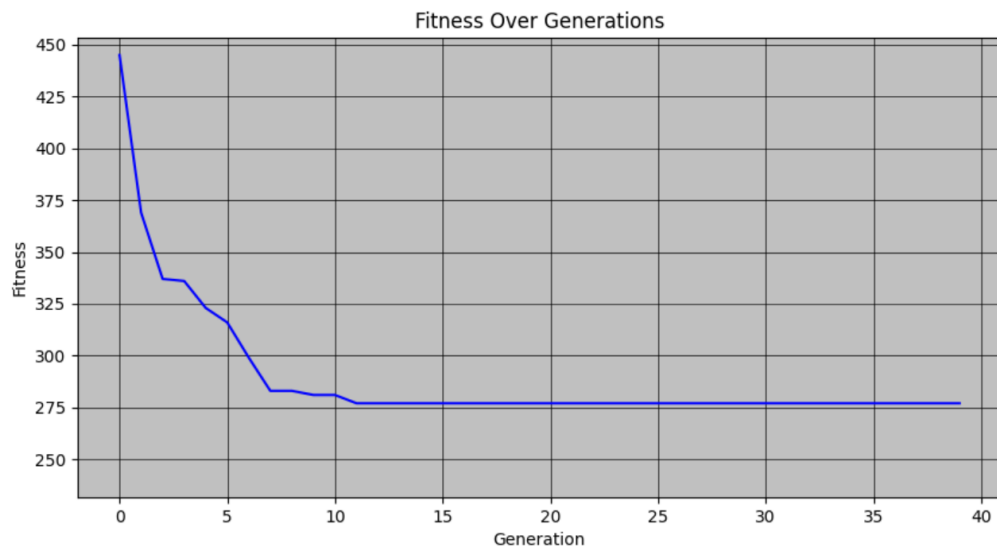


Figure 19: Fitness Graph for Test Problem

5.4 Test for MT06

Muth and Thompson Job Shop Problem - MT06

m= Machine ID

t= Processing time

Table 11 :MT06 problem

Process	1		2		3		4		5		6	
Jobs	m	t	m	t	m	t	m	t	m	t	m	T
1	3	1	1	3	2	6	4	7	6	3	5	6
2	2	8	3	5	5	10	6	10	1	10	4	4
3	3	5	4	4	6	8	1	9	2	1	5	7
4	2	5	1	5	3	5	4	3	5	8	6	9
5	3	9	2	3	5	5	6	4	1	3	4	1
6	2	3	4	3	6	9	1	10	5	4	3	1

For MT06 [24], the results are below. Also my benchmarks was the makespan obtained by Ripon et al. [2]. I have tried to compare my results with their results for mt06, mt10 and mt20 and also with best results already available and calculated by Muth and Thomson

- **Result of Benchmark: 55**
- **Best Known Makespan: 55**
- **Best Obtained Makespan by GAIP: 47**
- **Final_Chromosome: [[1,2,4,5,6,3],[5,2,1,4,6,3],[4,1,2,6,3,5],[4,1,3,5,6,2],[4,1,3,5,6,2],[3,5,4,1,2,6]]**

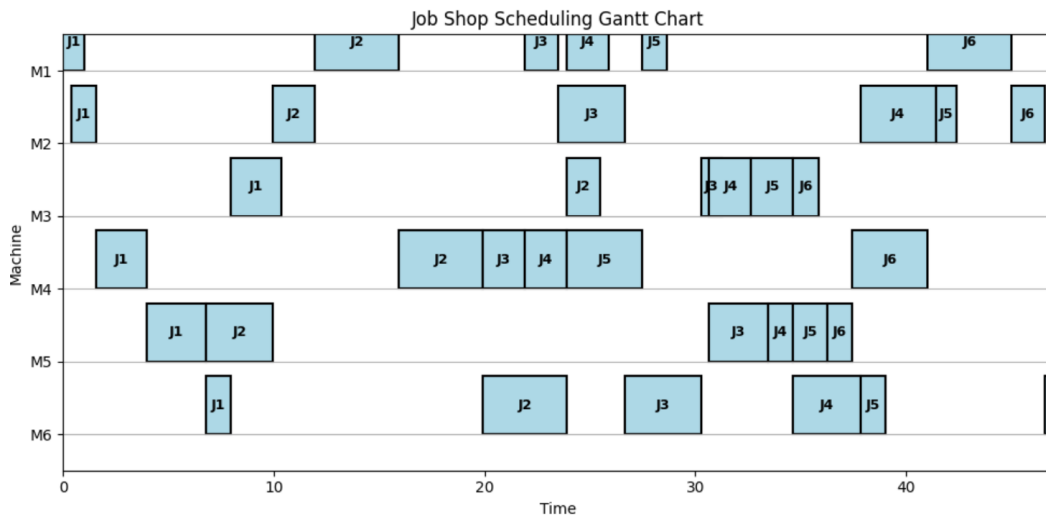


Figure 20: GANTT CHART for MT06

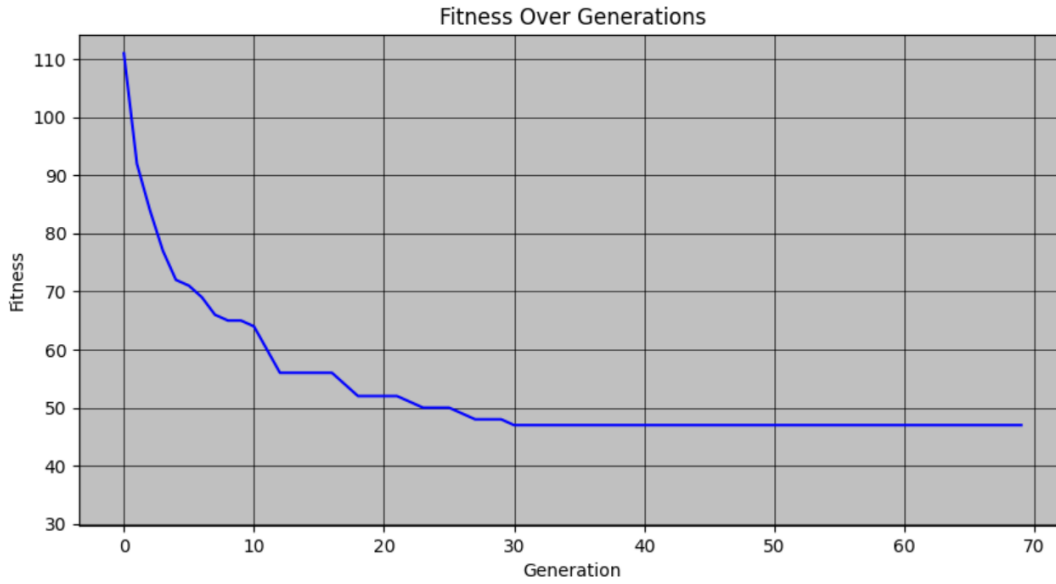


Figure 21: Fitness Graph for MT06

5. 5 Test for MT10

Muth and Thompson Job Shop Problem – MT10

Table11:MT10 problem

Process	1		2		3		4		5		6		7		8		9		10	
Jobs	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t
1	1	29	2	78	3	9	4	36	5	49	6	11	7	62	8	56	9	44	10	21
2	1	43	3	90	5	75	10	11	4	69	2	28	7	46	6	46	8	72	9	30
3	2	91	1	85	4	39	3	74	9	90	6	10	8	12	7	89	10	45	5	33
4	2	81	3	95	1	71	5	99	7	9	9	52	8	85	4	98	10	22	6	43
5	3	14	1	6	2	22	6	61	4	26	5	69	9	21	8	49	10	72	7	53
6	3	84	2	2	6	52	4	95	9	48	10	72	1	47	7	65	5	6	8	25
7	2	46	1	37	4	61	3	13	7	32	6	21	10	32	9	89	8	30	5	55
8	3	31	1	86	2	46	6	74	5	32	7	88	9	19	10	48	8	36	4	79
9	1	76	2	69	4	76	6	51	3	85	10	11	7	40	8	89	5	26	9	74
10	2	85	1	13	3	61	7	7	9	64	10	76	6	47	4	52	5	90	8	45

- **Result of Benchmark:** 930
- **Best Known Makespan:** 930
- **Best Obtained Makespan by GAIP:** 996

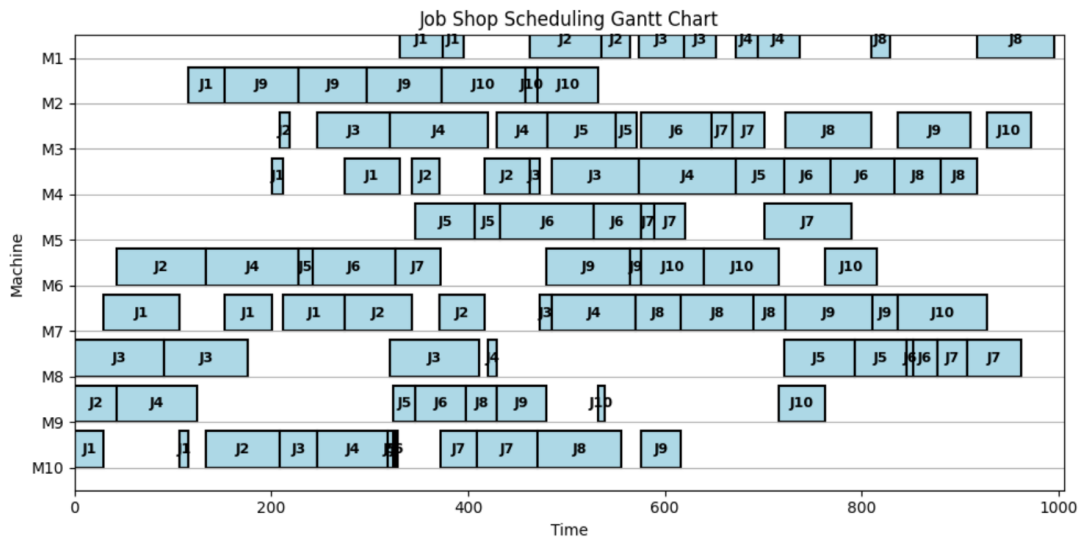


Figure 22: GANTT CHART for MT10

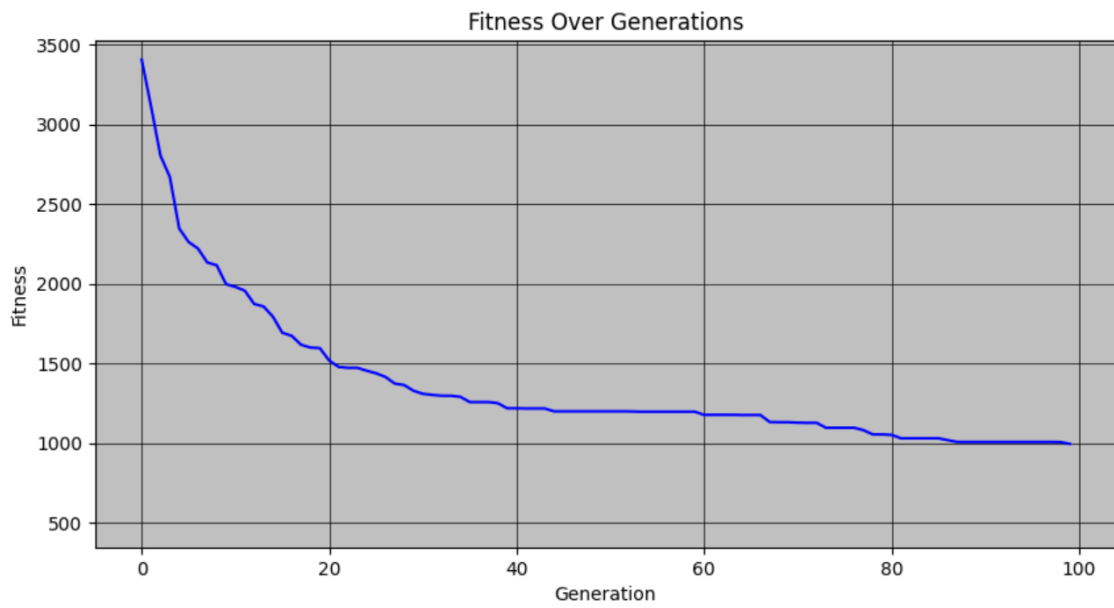


Figure 23: Fitness Graph for MT10

5. 6 Test for MT20

Muth and Thompson Job Shop Problem – MT20

Table 12:MT20 problem

Process	1		2		3		4		5	
	m	t	m	t	m	t	m	t	m	t
1	1	29	2	9	3	49	4	62	5	44
2	1	43	2	75	4	69	3	46	5	72
3	2	91	1	39	3	90	5	12	4	45
4	2	81	1	71	5	9	3	85	4	22
5	3	14	2	22	1	26	4	21	5	72
6	3	84	2	52	5	48	1	47	4	6
7	2	46	1	61	3	32	4	32	5	30
8	3	31	2	46	1	32	4	19	5	36
9	1	76	4	76	3	85	2	40	5	26
10	2	85	3	61	1	64	4	47	5	90
11	2	78	4	36	1	11	5	56	3	21
12	3	90	1	11	2	28	4	46	5	30
13	1	85	3	74	2	10	4	89	5	33
14	3	95	1	99	2	52	4	98	5	43
15	1	6	2	61	5	69	3	49	4	53
16	2	2	1	95	4	72	5	65	3	25
17	1	37	3	13	2	21	4	89	5	55
18	1	86	2	74	5	88	3	48	4	79
19	2	69	3	51	1	11	4	89	5	74
20	1	13	2	7	3	76	4	52	5	45

- **Result of Benchmarks: 1194**
- **Best Known Makespan: 1165**
- **Best Obtained Makespan by GAIP: 1131**

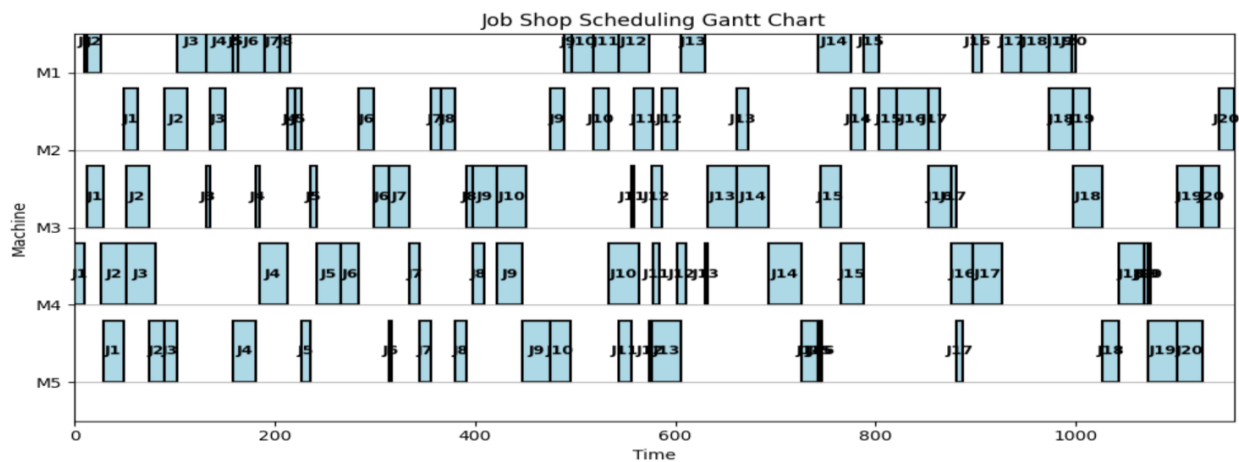


Figure 24: GANTT CHART for MT20

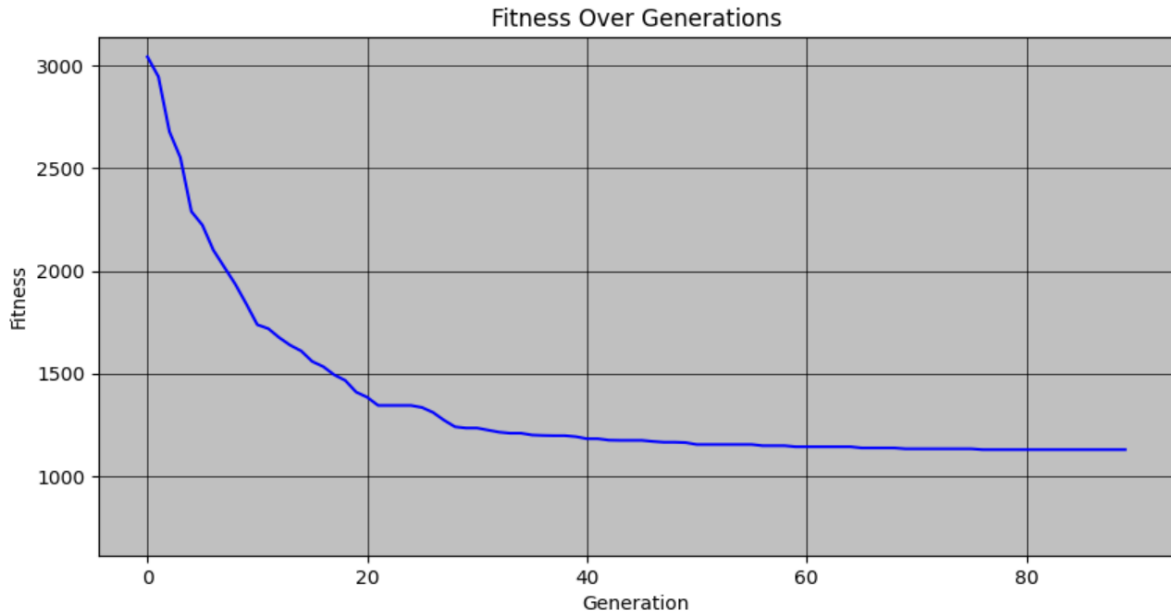


Figure 25: Fitness Graph for MT20

5.7 GA result for dataset instances from Muth and Thomson

Below are results for MT06, MT10 and MT20

Table 13:GA Results for Muth and Thomson

Instance	Ripon et al. [2]	Best Known	GAIP Results	Difference	Deviation (%)	Efficiency (estimate)
MT06	55	55	47	-8	-14	100
MT10	930	930	996	66	7.10	95
MT20	1194	1165	1131	-34	-2.92	100

5.8 GA result for dataset instances from (Lawrence, 1984)

5.8.1 LA01

Number of Machines: 5

Number of Jobs: 10

This benchmark outlines the job shop scheduling configuration where each

entry corresponds to a Machine ID followed by its associated processing time. I used Kalshetty et al. [1] as a benchmark to compare my results for Lawrence instances

10	5								
1	21	0	53	4	95	3	55	2	34
0	21	3	52	4	16	2	26	1	71
3	39	4	98	1	42	2	31	0	12
1	77	0	55	4	79	2	66	3	77
0	83	3	34	2	64	1	19	4	37
1	54	2	43	4	79	0	92	3	62
3	69	4	77	1	87	2	87	0	93
2	38	0	60	1	41	3	24	4	83
3	17	1	49	4	25	0	44	2	98
4	77	3	79	2	43	1	75	0	96

Figure 26: LA01 Benchmark

- **Results obtained by benchmark:** 1050
- **Best Known Makespan:** 666
- **Best obtained Makespan GAIP:** 640
- **Final Chromosome:** [[5,1,2,3,4], [2,5,3,4,1], [3,5,2,4,1], [1,4,2,5,3] [3,4,2,5,1], [2,1,5,4,3], [1,4,3,2,5], [2,1,3,5,4], [2,4,3,5,1], [1,4,5,3,2]]

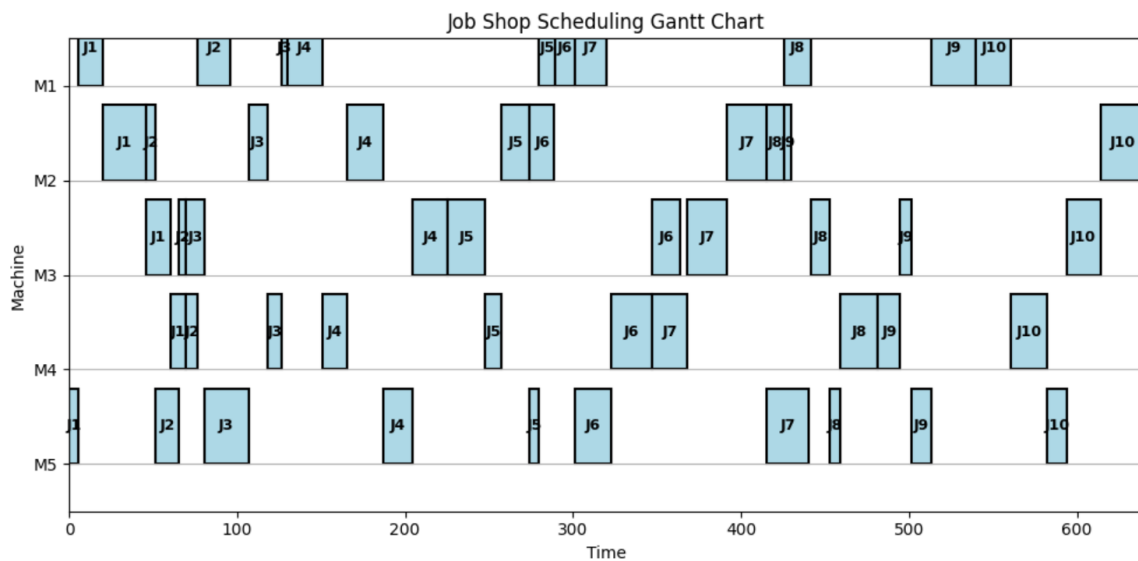


Figure 27: GANTT CHART for LA01

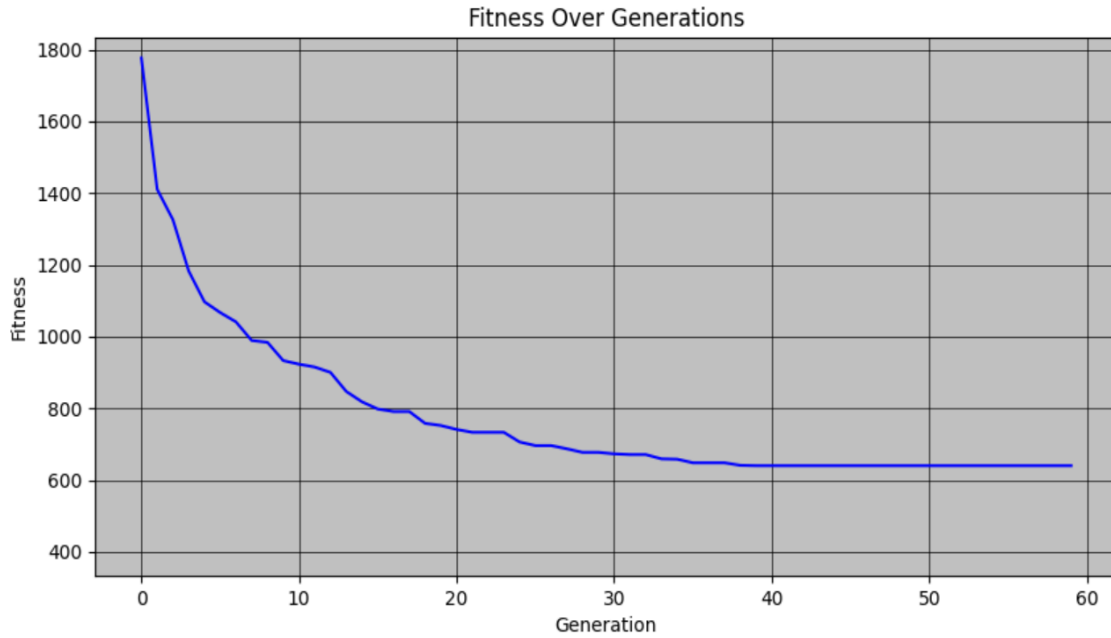


Figure 28: :Fitness Graph for LA01

5.9 GA result for dataset instances from (Lawrence, 1984)

Below are some of Lawrence [22] Instances which are tested by genetic algorithm

Table 14:GA Results for Lawrence Instances

Instance	Kalshetty et al. [1]	Best Known Makespan	Best Obtained By GAIP	Difference	Deviation	Efficiency (Approx%)
LA01	666	666	650	-16	-2.40	100
LA10	958	958	876	-82	-8.56	100
LA15	1207	1207	1162	-45	-3.7	100
LA20	902	1046	1070	24	2.29	80
LA25	935	977	1270	293	29.99	60

Chapter 6: Conclusion and Future Directions

6.1 Conclusion

This thesis presented a detailed analysis of job shop scheduling problems, with a particular focus on the application of genetic algorithms (GAs) incorporating Python (GAIP) to optimize scheduling efficiency in various industrial settings. The study utilized benchmark datasets from Muth and Thompson as well as extended instances like LA benchmarks to assess the performance of the planned genetic algorithms. Also, the project has facilitated the users to test custom benchmarks other than Muth and Thomson and Lawrence by manually entering the data.

Key findings demonstrated that genetic algorithms, enhanced with customized crossover and mutation strategies, significantly improve the scheduling effectiveness by reducing makespan compared to traditional methods. For instance, the experimental results on the MT06 instance showed an improvement over the best-known solutions, highlighting the sturdiness and efficiency of the genetic approach in handling intricate scheduling scenarios.

The application of genetic algorithms was further explored through the integration with graphical user interfaces (GUIs), which facilitated the visualization of scheduling outcomes, such as Gantt charts and fitness over generations. This not only provided a user-friendly platform for observing the algorithm's performance but also served as a tool for further analysis and refinement.

6.2 Contributions

The thesis contributed to the field of operations research in several ways:

Implementation in Python: My research utilizes Python to develop GA for job shop scheduling, addressing the gap in detailed programming implementations and optimization within this popular language.

Combination of Selection Methods: By integrating both tournament and roulette wheel selection methods, innovatively enhances the efficiency and

effectiveness of genetic algorithms, a combination not extensively explored in existing literature.

Enhancement of Genetic Algorithms: By integrating advanced genetic operators and local search techniques, the study advanced the understanding and capabilities of GAs in solving job shop problems.

Benchmarking and Analysis: Extensive testing on well-known benchmarks provided valued understandings into the strengths and limitations of the planned methods, contributing to ongoing discussions in the academic community.

Practical Application and Tool Development: The development of a GUI for the GA implementation offered a practical tool for researchers and practitioners to simulate and modify scheduling parameters dynamically and to test not only specific benchmarks but custom benchmarks also by using “Other Benchmarks” option.

6.3 Implications

The implications of this research are twofold. Practically, the findings support the adoption of genetic algorithms in industries where job shop scheduling plays a significant role, such as manufacturing and logistics.. Academically, the research fills a gap in the study of hybrid genetic algorithms, presenting a new avenue for exploring algorithmic interactions with user-driven interfaces.

6.4 Future Directions

Looking ahead, the scope for advancing this research is substantial and can be directed along several promising avenues:

- **Dynamic Scheduling Environments:**

Future research could focus on dynamic versions of the JSSP where job arrivals, processing times, and machine readiness are not acknowledged a preceding. Incorporating real-time data and adaptive algorithms could potentially reduce wait times and improve throughput in manufacturing systems.

- **Advanced Machine Learning Models:**

Exploring more complex machine learning models, such as deep learning and reinforcement learning, could provide new ways to approach the non-linearity and high dimensionality of advanced scheduling problems. These models could evolve from learned experiences, improving their decision-making processes over time.

- **Cross-Disciplinary Applications:**

Applying findings from this research to other domains such as healthcare, where scheduling of operating rooms and medical staff is crucial, could be explored. Similar strategies could optimize logistics in transportation or streamline project management in software development.

- **Sustainability and Resource Optimization:**

Future studies could incorporate sustainability metrics into the scheduling algorithms to optimize energy consumption and minimize waste. This approach would align industrial practices with global sustainability goals, offering economic and environmental benefits.

- **Quantum Computing:**

Investigating the potential of quantum computing to solve complex JSSP could break the barriers of computational times and solution quality. Early research into quantum algorithms for optimization problems shows promising results that could be pivotal for scheduling problems.

- **Customization for Small to Medium Enterprises (SMEs):**

Developing scalable and customizable solutions that can be easily implemented in SMEs could democratize access to advanced scheduling tools, enabling smaller manufacturers to compete more effectively on a global scale.

Creating interactive applications and simulation-based learning tools to train personnel in optimization and scheduling could help bridge the gap between theoretical research and practical, on-the-ground application.

Each of these areas not only extends the current research frontier but also invites interdisciplinary collaboration and technological innovation, ensuring that the field of scheduling remains responsive to the evolving industrial landscape and emerging technological trends.

This comprehensive forward-looking view sets the stage for ongoing contributions to the field of operations research and beyond, promising to address the pressing challenges of efficiency and complexity in diverse scheduling environments.

This thesis not only enhances the existing body of knowledge on job shop scheduling but also provides scalable, efficient solutions adaptable to various industrial needs. The foundations laid by this research surface the method for innovative coming studies, potentially revolutionizing the approaches to operational scheduling problems.

References

- [1] Y. R. Kalshetty, A. C. Adamuthe, and S. Phani Kumar, "Genetic Algorithms with Feasible Operators for Solving Job Shop Scheduling Problem," Dept. of CSE, GITAM University, Hyderabad, India, and Dept. of CS & IT, RIT, Rajaramnagar, Sangli, MS, India.
- [2] K. S. N. Ripon, N. H. Siddique, and J. Torresen, "Improved precedence preservation crossover for multi-objective job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 23, no. 6, pp. 2065-2079, Dec. 2010. DOI: 10.1007/s10845-010-0475-0.
- [3] S. Ikramullah and S. Hou-Fang, "Application of genetic algorithm and rules in the scheduling of flexible job shop," 2007
- [4] H. C. Chang, Y. P. Chen, T. K. Liu, and J. H. Chou, "Solving the flexible job shop scheduling problem with makespan optimization by using a hybrid Taguchi-genetic algorithm," *IEEE Access*, vol. 3, pp. 1740-1754, 2015.
- [5] W. P. Adams, M. Balas, and H. W. Morton, "A local search heuristic for job shop scheduling," *Operations Research*, vol. 36, no. 2, pp. 258-272, 1988.
- [6] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149-156, 1991.
- [7] N. Kundakcı and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," Department of Business Administration, Pamukkale University, 20070 Denizli, Turkey; Department of Industrial Engineering, Pamukkale University, 20070 Denizli, Turkey.
- [8] OR Library, Available: <http://mscmga.ms.ic.ac.uk>. Accessed: Jul. 12, 2009.
- [9] S. H. Niu, S. K. Ong, and A. Y. C. Nee, "An improved intelligent water drops algorithm for solving multi-objective job shop scheduling," Mechanical Engineering Department, National University of Singapore, 9 Engineering Drive 1, Singapore 117576, Singapore.
- [10] D. Barat, "JOB SCHEDULING WITH GENETIC ALGORITHM," M.S. thesis, Dept. Computer Science, North Dakota State Univ., Fargo, ND, January 2013.
- [11] S. Verma, M. Jain, and D. Choudhary, "Solving the Job-Shop Scheduling Problem by using Genetic Algorithm," Department of Applied Mathematics & Computational Science, S.G.S. Institute of Technology & Science, Indore, M.P., India.

- [12] P. Pongchairerks, "A two-level metaheuristic algorithm for the job-shop scheduling problem," *Complexity*, pp. 1-11, 2019.
- [13] P. S. Narayanan et al., "Job Shop Scheduling Using Heuristics Through Python Programming and Excel Interface," Department of Mechanical Engineering, Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India and Maryland Health Benefit Exchange, Baltimore, MD, USA.
- [14] C. Darwin, *On the Origin of Species by Means of Natural Selection*, ed. J. Carroll, Toronto: Broadview, 2003.
- [15] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1989.
- [16] M. A. Vonderembse and G. P. White, *Operations Management: Concepts, Methods, and Strategies*.
- [17] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial Scheduling*, vol. 3, no. 2, pp. 225-251, 1963.
- [18] Z. Wang et al., "A hybrid algorithm for order acceptance and scheduling problem in make-to-stock/make-to-order industries," *Computers & Industrial Engineering*, vol. 127, no. 46, pp. 841-852, 2019.
- [19] P. Pongchairerks, "A two-level metaheuristic algorithm for the job-shop scheduling problem," *Complexity*, pp. 1-11, 2019.
- [20] Y. Gao et al., "Job-shop scheduling considering rescheduling in uncertain dynamic environment," in *Proceedings of the 16th International Conference on Management Science & Engineering*, Moscow, Russia, 2009, pp. 380–384.
- [21] H. Ge et al., "A hybrid algorithm based on particle swarm optimization and simulated annealing for job shop scheduling," in *Proceedings of the 3rd International Conference on Natural Computation*, Haikou, China, 2007, pp. 715–719.
- [22] S. Lawrence, "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement)," Graduate School of Industrial Administration, Carnegie-Mellon University, 1984.
- [23] J. F. Muth and G. L. Thompson, *Industrial Scheduling*. Prentice-Hall, 1963.