

分类号:

学校代码:10363

密 级:

学 号:



安徽工程大学

Anhui Polytechnic University

硕士学位论文

题 目

Hybrid Artificial Bee Colony Algorithm and Its
Application in Multi-Objective Flexible Job Shop
Scheduling Problem

论 文 作 者

MOHAMED EL-ATTAOUY

指 导 教 师

LI MING

学 科 (专 业)

Control Science and Engineering

研 究 方 向

Swarm Intelligence and Based Optimization

论文提交日期: 2025 年 03 月 23 日

分类号：
密级：公开

单位代码：10363
学 号：5220310102

Hybrid Artificial Bee Colony Algorithm and its Application in Multi-objective Flexible Job Shop Scheduling Problem

学生姓名 (Student Name):	MOHAMED EL-ATTAOUY
指导教师 (Supervisor):	Li Ming (Professor)
专业 (Specialty):	CONTROL SCIENCE AND ENGINEERING
研究方向 (Research Area):	SWARM INTELLIGENCE AND BASED OPTIMIZATION

论文答辩日期：2025 年 5 月 10 日

安徽工程大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律后果由本人承担。

学位论文作者签名 (Author Signature) : MOHAMED EL-ATTAOUY



日期 (Date): 2025 年 5 月 15 日

安徽工程大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权安徽工程大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密 ☐，在_____年解密后适用本版权书。

本学位论文属于

不保密 ☒。

学位论文作者签名：MOHAMED EL-ATTAOUY

日期： 2025 年 5 月 15 日



指导教师签名：李明

日期： 2025 年 5 月 15 日



HYBRID ARTIFICIAL BEE COLONY ALGORITHM AND ITS APPLICATION IN MULTI-OBJECTIVE FLEXIBLE JOB SHOP SCHEDULING

ABSTRACT

This study proposes the Hybrid Artificial Bee Colony algorithm in the Multi-objective Flexible Job Shop Scheduling Problem. The NP-hard problem plays a crucial role in contemporary manufacturing systems. FJSP entails allocating and sequencing a set of operations, each belonging to different jobs, onto machines with different capabilities. The primary goal of the research is to minimize three important performance metrics simultaneously: the total completion time (makespan), the maximum workload on any machine, and the overall workload on all machines. These goals typically have conflicting interests, for which a developed multi-objective optimization framework is needed to address trade-offs.

To solve such difficulties, this study combines the Artificial Bee Colony (ABC) metaheuristic model inspired by the natural foraging behavior of the honeybee with Pareto-based multi-objective optimization methods. The hybrid approach uses the Artificial Bee Colony (ABC) algorithm's power to explore large solution spaces while utilizing Pareto principles to keep a diversified archive of non-dominated solutions. The hybridization guarantees that the algorithm efficiently solves the routing (machine assignment) and sequencing (operation order) sub-problems characteristic of the Flexible Job Shop Problem (FJSP). Besides, the algorithm uses sophisticated local search operators to improve candidate solutions and supplement the exploration of promising areas in the solution space, thus speeding up the convergence to optimal or near-optimal schedules.

Comprehensive experimental tests were conducted based on well-known benchmark problems from the literature, such as those reported by Kacem and Brandimarte. The algorithm was experimented with over varying problem sizes, ranging from smaller scale (e.g., 4×5 instances) to more complicated situations with

8×8 and 10×10 instances. The key conclusions show that our proposed method can yield good results over the objective values in every case studied, and it is better than the classical algorithms. It results from minimizing the makespan and optimizing the workload between machines, decreasing idle time, and improving resource utilization. One of the experiments' main results is the algorithm's ability to create applicable scheduling solutions. For instance, it determined 14 different schedules for a 4×5 instance, 82 for an 8×8 instance, and 3,555 different solutions for a 10×10 instance. Such diversity is necessary for decision-makers, as it offers various options for varying operational limitations.

The cardinal findings of the research are dual. To begin with, the amalgamation of the Artificial Bee Colony (ABC) metaheuristic with the Pareto-based optimization techniques is a solid and scalable framework that efficiently solves the complex problems in multi-objective FJSP. The technique not only provides better results in terms of performance criteria but also has a good diversity to handle the problems related to the changing production systems, which are dynamic by nature. Secondly, the modified algorithm considerably lowers the computation cost compared to the primacy methods, so it can be considered a practical tool for industrial applications. This article finds that the possible upgrades beyond the parallel processing method added with more complex local search strategies would enhance the algorithm's performance.

The present investigation offers a considerable boost to the understanding of using swarm methods to solve complex planning difficulties within versatile production setups. The proposed method's capability, as it works to better many competing aims and offers assorted answers, significantly improves operations research with industrial applications. This work forms the basis for future investigations on flexible and effective arranging systems. Such systems can adjust to the changing demands of current manufacturing environments.

KEYWORDS: Production Program, FJSP, Makespan, Pareto, NP-hard Problems, Multi-objective Optimization.

Contents

CHAPTER 1 INTRODUCTION.....	1
1.1 Motivation/Justification.....	2
1.2 Objectives	2
1.3 Organization of Work.....	3
CHAPTER 2 THEORETICAL FOUNDATION/BASIC CONCEPTS.....	5
2.1 Most Common Production Scheduling Problems.....	5
2.2 Definition of FJSP.....	6
2.3 Performance Criteria.....	8
2.4 Optimization of multi-objective.....	9
2.4.1 Techniques for Handling Multi-Objectives in FJS Problems.....	9
2.4.2 Approaches to Production Scheduling Problems.....	10
2.5 Biological Basis of the Used Metaheuristic.....	11
2.5.1 Artificial Bee Colony Algorithm.....	12
2.5.2 Local Search Algorithms.....	12
CHAPTER 3 LITERATURE REVIEW.....	16
3.1 Solving FJSP with a Hybrid Artificial Bee Colony Approach.....	16
3.2 Solving FJSP Comparing Various Heuristics.....	17
3.3 Solving FJSP with BCO and Search Strategies.....	19
3.4 Solving FJSP with Artificial Bee Colony and SPO.....	21
3.5 Solving Machine Scheduling Problems with Sequence Dependent Using Hybrid Artificial Bee Colony.....	22
3.6 Hybrid Algorithm Between Artificial Bee Colony, Tabu Search (TS), and Pareto Applied to Multi-Objective FJSP.....	25
3.7 Hybrid Artificial Bee Colony Algorithm Aiming to Present Solution Diversity.....	27
3.8 Genetic Algorithm with Bee Algorithm Applied to Multi-Objective FJSP.....	28
3.9 Hybridization with Other Metaheuristics.....	30

3.10 Final Considerations of the Chapter	30
CHAPTER 4 WORK PROPOSAL	32
4.1 Description of methods and strategies used	35
4.1.1 Configuration of Code Adjustment Parameters	36
4.1.2 Creation of Initial Solutions	39
4.1.3 Function Fitness	39
4.1.4 Aggressiveness of Changes by Employed Bees (Exploitation vs. Exploration).....	40
4.1.5 Solution Representation	40
4.1.6 Employed Bee Phase	41
4.1.7 Onlooker Bee Phase	42
4.1.8 Pareto Archive	45
4.1.9 Scout Bee Phase	46
4.1.10 Strategy for Identifying Multiple Solutions for the Same Objective Values.....	47
4.1.11 Results Evaluation	48
4.1.12 Result Visualization Methods	49
4.1.13 Validation of Found Results	49
CHAPTER 5 Results and Experimental Analysis	50
5.1 Kacem, Hammadi, and Borne (2002) Instances	50
5.2 Instances of Brandimarte (1993)	66
CHAPTER 6 CONCLUSION AND FINAL CONSIDERATIONS	76
6.1 Future Work	77
References	79
Appendix	83
Catalog of academic papers and achievements published during the degree study period.....	87
Acknowledgments	86

List of Figures

Figure 2-1	Instance Barter.....	13
Figure 2-2	Instance Reverse 1.....	13
Figure 2-3	Instance Reverse 2.....	13
Figure 2-4	Instance insert	14
Figure 2-5	Instance in verse 2.....	14
Figure 2-6	Instance of crossing genetics	15
Figure 2-7	Instance of Mutation.....	15
Figure 3-1	Representative Diagram of the algorithm.....	22
Figure 3-2	Layers Representation.....	23
Figure 3-3	Number of Pareto-Optimal Solutions.....	24
Figure 4-1	General Flowchart of the Code, Representing the Main Function.....	35
Figure 4-2	Flowchart of Solutions Perturbation Process.....	38
Figure 4-3	Flowchart of representation from the stage of the bee employees	41
Figure 4-4	Illustrative Probabilistic Roulette Wheel	42
Figure 4-5	Illustrative Roulette Wheel for Onlooker Bee Search	44
Figure 4-6	Flowchart from the stage of creation of file of solutions Pareto	46
Figure 5-1	All results for the Objective Set 11,32,10 in the 4x5 instance.....	54
Figure 5-2	All results for the Objective Set 12,32,8 in the 4x5 instance.....	55
Figure 5-3	All results for the Objective Set 13,33,7 in the 4x5 instance	55
Figure 5-4	results for the Objective Set 11,34,9 in the 4x5 instance.....	56

Figure 5-5	Pareto frontier for the 4x5 instance.....	56
Figure 5-6	Number of solutions identified per result.....	57
Figure 5-7	Two solutions identified for the 8x8 instance.	57
Figure 5-8	Results identified for the 8x8 instance.	58
Figure 5-9	Results identified for the 8x8 instance.	59
Figure 5-10	Results Similar found for the 8x8 instance.	60
Figure 5-11	Pareto frontier for the 8x8 instance.	60
Figure 5-12	Results found for the 10x10 instance.	61
Figure 5-13	Result found for the 10x10 instance.	62
Figure 5-14	Diversity of values found for the 10x10 instance.	63
Figure 5-15	Pareto frontier for the 10x10 instance.	63
Figure 5-16	Diversity found for the 15x10 instance.	64
Figure 5-17	Result found for the 15x10 instance.	65
Figure 5-18	Pareto frontier for the 15x10 instance.	66
Figure 5-19	Pareto frontier for the Brandimarte mk01 instance.	70
Figure 5-20	Pareto frontier for the Brandimarte mk02 instance.	71
Figure 5-21	Pareto frontier for the Brandimarte mk03 instance.	71
Figure 5-22	Pareto frontier for the Brandimarte mk04 instance.	72
Figure 5-23	Pareto frontier for the Brandimarte mk05 instance.	73
Figure 5-24	Diversity for the Brandimarte mk05 instance.	73
Figure 5-25	Pareto frontier for the Brandimarte mk07 instance.	74
Figure 5-26	Diversity for the Brandimarte mk07 instance.	74
Figure 5-27	Pareto frontier for the mk08 instance.	75
Figure 5-28	Pareto frontier for the mk09 instance.	75

List of Tables

Table 3-1	Comparison of the results	19
Table 3-2	Results of Kacem's instances.....	26
Table 3-3	Results for Kacem's instances.....	27
Table 3-4	Results for two Kacem instances.....	28
Table 3-5	Results achieved	29
Table 4-1	Import of processing times from the 4x5 instance.....	36
Table 4-2	Complete vector of a solution.....	40
Table 5-1	Presentation of configuration values for the Kacem instances	51
Table 5-2	Comparison of results from other authors with the Proposed Algorithm....	53
Table 5-3	Configurations of the Brandimarte instances.....	67
Table 5-4	Comparison of the results obtained with those from the literature for the mk01, mk02, mk03, and mk04 instances of Brandimarte.....	68
Table 5-5	Comparison of the obtained results with those from the literature for the mk05, mk07, mk08, and mk09 instances of Brandimarte.....	69

CHAPTER 1 INTRODUCTION

Production systems have become increasingly flexible to meet the evolving demands of a dynamic market^[1]. With this market trend towards flexibilization of manufacturing processes more and more researchers publish works related to Flexible Job Shop Problem (FJSP).

The Flexible Job Shop Scheduling Problem (FJSP) has received considerable attention recently in computer science and information technology, and many heuristic algorithms, such as genetic algorithms, Swarm Particle optimization, Artificial Bee Colony (ABC) algorithms, and ACO, have been proposed to solve it^[2]. The proposed works are being developed considering multi-objectives, as decision-making is a process of evaluating available alternatives and selecting an appropriate action from a set of other options, which becomes more challenging when the process involves multiple criteria rather than a single one. Moreover, most decisions related to production control in industrial environments are made under multiple criteria, which can sometimes be conflicting, to satisfy various needs simultaneously.

The Flexible Job Shop Scheduling Problem (FJSP) is defined by a set of “j” jobs, each job consisting of “o” operations. These operations are each processed by a machine m that is part of the production environment's set of machines. In this type of system, each operation within the jobs can be processed on a different machine, and what makes the process flexible is that all machines are capable of processing all operations.

This type of problem can be conceptually defined; however, it falls into the class of NP-Hard problems, meaning those problems in computer science that cannot be deterministically solved in polynomial time^[3]. Systems developed for solving the FJSP are programmed so that the methods and heuristics work towards one or more objectives simultaneously, such as minimizing the makespan (the time elapsed until the last operation is completed) or minimizing idle time (the time machines remain idle). In the literature, the criterion of reducing makespan is the most commonly used for applying the FJSP, and with the addition of more objectives/performance criteria, the difficulty in programming and

computational costs increase, requiring more computational power and optimization methods. Consequently, this topic is increasingly used in this area, and solution methods are becoming more capable of solving real-world manufacturing problems.

Since the scheduling problem in FJSP systems has computational costs and characteristics that are typical of such problems, several authors propose heuristic and meta-heuristic approaches to deal with them. In recent years, works using techniques such as investigation have been done. (IA)^[4], Thermal-Based Search Method (TBSM)^[5]Genetic Algorithms (GA), Ant Colony Optimization (ACO)^[6], Artificial Bee Colony (ABC)^[7]Pareto, Bee Swarm Algorithms, among others, have been found. These techniques have shown promising results considering their application to the FJSP, especially the Artificial Bee Colony algorithm recently proposed by various authors such as Li, Pan, and Xie (2011), Gao et al. (2015), Wang et al. (2011), Li, Pan, and Tasgetiren (2014). Given the results presented by authors who used Artificial Bee Colony to address the multi-objective FJSP, an approach is proposed that aims to use the Artificial Bee Colony meta-heuristic with modified search methods and the Pareto method to perform the routing and sequencing of machines and operations in FJSP.

1.1 Motivation/Justification

This research is driven by the need to reduce operational costs in manufacturing industries while improving production effectiveness. Even though the industrial sector is equipped with sophisticated machinery, the machines' maximum potential is often ineffective when the production schedule is poorly managed.

1.2 Objectives

This work aims to present the Artificial Bee Colony (ABC) metaheuristic to address the multi-objective Flexible Job Shop Problem (FJSP), one of the challenges in production scheduling. Other heuristic-based methods will enhance the algorithm's efficiency in adapting to the FJSP and support the main metaheuristic.

Local and global search techniques are implemented to allow the algorithm to alternate between search modes, finding the expected results as quickly as possible while

discovering a wide diversity of reasonable solutions. In this context, diversity refers to different schedules for the same objective values intended to be minimized. For example, if a result of 20,32,14 is found 10 times with various schedules, it is understood that these values represent a diversity of 10 solutions for the result 20,32,14.

The work focuses on diversity, considering the difficulty of finding studies with this characteristic in the literature. Most existing works focus on solution quality and do not present the diversity of solutions found. The proposed algorithm will be directed towards multi-objective treatment, where three objectives will be addressed simultaneously, and it is expected that a high diversity of solutions will be obtained for each result presented.

The objectives explored are minimizing the total time required to complete all operations., minimizing the workload Assigned to the machine with the heaviest workload., and minimizing the workload total among all machines. Diversity will not be treated as an objective but will be considered when presenting results.

1.3 Organization of Work

The work is broken into seven chapters. In the first chapter, I quickly describe the problem, explain what drove me to choose this topic, and list the primary objectives I intend to achieve.

In the second chapter, I lay out the essential theoretical framework and introduce the key concepts that form the backbone of this study. It addresses the most common approaches to tackling the problem under investigation, provides detailed definitions of the problem and solution methods, and introduces the performance criteria that will be utilized.

Chapter three is a literature review. It isolates the key studies within the discipline that address the topic under consideration, highlighting the most influential research that has produced renowned findings with similar approaches employed in the present research.

The fourth chapter contains the development proposal, which includes implementation details, strategies, and the activity's operational methodology. This chapter explains in more detail the methods employed to produce the outcomes, as shown in the next chapter.

The fifth chapter includes the findings and focused comments based on the outcomes

of the numerous investigated cases. It compares the acquired values to those of other authors and uses visual aids (such as graphs) to demonstrate the quality of the solutions.

Chapter 6 offers an overview of the proposed approach and the results achieved. It also offers suggestions for improving the developed framework to guide future research.

CHAPTER 2 THEORETICAL FOUNDATION/BASIC CONCEPTS

This chapter presents the basic fundamentals required to be able to read this work. It states important principles and fundamentals and defines acronyms and terms utilized in the field.

The problem addressed herein concerns a critical component of manufacturing systems called the Flexible Job Shop problem (FJSP), researched by several authors, including some cited in Chapter 3. The FJSP is an established NP-hard problem and one of the most important machine scheduling challenges. It generalizes the classic Job Shop Problem (JSP), where one machine can perform each operation. In contrast, the FJSP, a generalization of the JSP, allows several machines to process the same operation, making the problem much more difficult to solve.

2.1 Most Common Production Scheduling Problems

The rules and constraints of the factory layout often cause various production scheduling problems in manufacturing. These are rules and restrictions regarding available machines/resources and the operations required to complete the work orders. Allahverdi, Ng et al. (2008) classify some production scheduling problems as scheduling operations across different plant layouts, ranging from one machine handling one operation to multiple machines with diverse operations and no precedence constraints.

Single Machine Scheduling: Only one machine processes the single operation required for each job.

Parallel Machine Scheduling: Each job has one operation, but multiple machines can process jobs simultaneously.

Flow Shop Scheduling: This problem involves a set of jobs and machines. All jobs are processed in the same order across all machines.

Flexible Flow Shop Scheduling: An extension of the traditional flow shop scheduling

approach., where operations are sequenced in stages, and each stage may have parallel machines handling the same task. Job Shop Scheduling: Unlike a flow shop, this allows for different processing sequences for each machine.

FJSP solves the job shop problem by allowing each task to be assigned to any machine from a predefined establishment. Machines might process the same operation differently at different times.

Open Shop Scheduling: Operations have no precedence constraints and can be processed in any order.

This work addresses the FJSP problem. The following section defines it formally.

2.2 Definition of FJSP

The Flexible Job Shop Problem (FJSP) is a crucial manufacturing system that has been the subject of thorough research and is significantly applicable in production scheduling and management. As stated by Suganthan, Gao et al. (2015), the incorporation of multiple machines into the same operation depending on the specifics of the problem makes the FJSP significantly more complex than the traditional Job Shop Problem (JSP) and, therefore, more challenging to solve.

The JSP-type scheduling problem can be formulated as described:

The counter z generically represents a maximum number of positions in a given set. This counter is used throughout the work to denote any arbitrary number.

- There is a limited but valuable collection of jobs available. (J_k), where $k=1, \dots, z$;
- There is a limited but valuable collection of machines (M_p), where $p=1, \dots, z$;
- Each Job comprises a finite number of operations o_{ij} , where $i=1, \dots, z$;
- Each operation $o_{j,i}$ of Job j_k requires processing by an advanced machine m from the set M of machines. Here, j corresponds to a given Job, and the symbol i refers to the i th operation of j ;

- C_m represented as makespan (total completion time).
- W_m represents the workload in time units (ut) of machine m , where $m=1, \dots, z$
- W_t represents the workload total in ut Among all machines, combined.

The FJSP is a set of jobs that must be done in a certain order but not simultaneously. Understanding this process is essential for optimal results and efficiency. For example, the first operation of each job ($O1,1$ $O1,1$) must be executed before subsequent operations ($O1,2$ $O1,2$, $O1,3$ $O1,3$, etc.). However, operations from different jobs can interleave. For instance, in a scenario involving three jobs, each job consists of three operations. the sequence: [$O1, 1$ | $O3, 1$ | $O1, 2$ | $O2, 1$ | $O3, 2$ | $O2, 2$ | $O2, 3$ | $O1, 3$ | $O3, 3$] is valid, the sequence of operations within each job is maintained. A machine processes each job operation at a specific time, and each machine processes only one operation at a time^[8].

Additional constraints are essential for defining and addressing the problem, as presented by Deng et al. (2017), Li et al. (2016), Li et al. (2011), Choo et al. (2016), and others. Operations cannot be interrupted once started; Machines can process only one operation at a time; An operation cannot be performed by multiple machines simultaneously; the movement between operations and the time required for machine setup is minimal; Machines operate independently of one another, and jobs also function independently from each other, among other factors.

- Operations cannot be interrupted once initiated;
- Machines may process only one operation at a time;
- No operation may be processed by multiple machines simultaneously;
- Movement between operations and machine setup times are deemed negligible;
- Machines operate independently of one another;
- Jobs are mutually independent.

Kacem, Hammadi, and Borne (2002) defined two types of FJSP: the partially flexible problem, which involves at least one machine unable to handle a specific operation, while the fully flexible model allows all machines to process every operation effortlessly! Thus, considering the defining rules of FJSP, regardless of whether the problem is partially or fully flexible, any viable solution must involve two fundamental stages: machine assignment and scheduling. A machine for processing a given operation is selected at the machine assignment stage. The order and timing of operations on assigned machines are determined in the scheduling stage.

2.3 Performance Criteria

Measures of process effectiveness are called performance criteria. Five key performance criteria are identified for this purpose by Slack, Chambers and Johnston (2009): Quality, Speed, Dependability, Flexibility and Cost. Indirectly related to quality is the production of goods or services without defects so that customers get goods or services that meet their requirements. Speed and dependability relate to the timely delivery of products or services as agreed in contracts. Flexibility means modifying plans to meet customer demands or dealing with unexpected circumstances to meet production demands. It involves cost-reducing production while maintaining proposed quality standards - to produce a satisfactory return on investment.

For the FJSP, performance criteria are typically used to measure the quality of schedules, and these criteria vary according to the objectives researchers aim to achieve. Three performance criteria, commonly referred to as Makespan (C_m), Workload Maximum (W_m), and Workload Total (W_t), were used to evaluate the results obtained in this work.

Makespan (C_m): This refers to production speed and the time needed to finish processing all occupations. It is usually directly related to production costs.

Maximum Workload (W_m) also depends on the processing speed of operations and is the time the machine spends working the most.

Workload Total (W_t) is the total time all machines are working, and is related to production costs: the fewer the machines work, the fewer resources they use.

These equations can represent the performance criteria:

Makespan:

$$\text{Minimize } C_m = \max \text{ where, } (1 \leq m \leq z) \quad (2-1)$$

So, in equation (2-1), z is the maximum number of operations on the problem.

Workload Maximum:

$$\text{Minimize } W_m = \max \text{ where, } (1 \leq m \leq z) \quad (2-2)$$

In equation (2-2), z denotes the maximum possible number of machines applicable to

the problem.

Workload Total:

$$\text{Minimize } \mathbf{wt} = \sum_{k=1}^m \mathbf{wt} \quad (2-3)$$

As in equation (2-3), m is the number of machines in the set of machines.

According to the reviewed literature, the criteria most commonly used by researchers dealing with FJSP-type problems are those presented above.

Treatment of multi-objective aspects is briefly reviewed next.

2.4 Optimization of multi-objective

Optimization of multi-objectives involves optimizing objective functions such as minimization or maximization. These must satisfy the rules and constraints applicable to each objective function for the function to be considered feasible.

2.4.1 Techniques for Handling Multi-Objectives in FJS Problems

Wang, Zhou et al. (2011) describe in their work a way of representing the multi-objective problem without compromising generality:

$$\text{Minimize } y = f(X) = (f_1(x), f_2(x), \dots, f_q(x)), \text{ where, } x \in \Omega, y \in \mathbb{R}^q \quad (2-4)$$

X represents the set of variables intended to be minimized within the solution space Ω , and y is the objective vector, containing all the objectives to be minimized, with $q > 1$ objective. THE representation $f_1(x)$ represents the objective one to be minimized.

Equation (2-5) lists previously presented objectives and can mathematically represent the three objectives to be minimized.

$$\text{Minimize } y = f(X) = (f_1(C_m), f_2(W_m), f_2(W_t)), \text{ where, } x \in \Omega, y \in \mathbb{R}^q \quad (2-5)$$

Wang, Zhou, et al. (2011) also present in their work some techniques used to treat the multi-objective problem, such as Pareto Dominance, Pareto optimal or nondominated solution, Pareto Front, Archive set, and Non-Dominated Selection.

Pareto Dominance: a solution A dominates a Solution B only if and only if $i \in \{1, 2, \dots, q\}$: $f_i(a) \leq f_i(b)$ and $\exists i \in \{1, 2, \dots, q\}$: $f_i(a) < f_i(b)$ (Wang, Zhou, et al., 2011).

So solution A is superior to solution B only if at least one attribute of a has a value smaller than the same attribute of solution b (considering minimization optimization). The remaining characteristics of solution a are at least equal to the corresponding attributes of solution b.

Optimal Pareto Solution or Non-Dominated: A solution is considered optimal if no other solution outperforms it.

Pareto Front: This is the set of all Pareto-optimal solutions, the collection of solutions that are not dominated.

Archive Set: This set stores only Pareto-optimal solutions. It is updated at each iteration. We continuously incorporate new Pareto-optimal solutions into our portfolio while decisively eliminating any dominated solutions, ensuring that our set remains robust and focused on the most effective choices.

Non-Dominated Sorting/Classification: This technique involves n sets of non-dominated solutions. The sets are ranked such that each subsequent set is inferior to the previous one and assigned a ranking value based on its degree of dominance. The first front has non-dominated solutions, and each item is ranked 1. The second front ranks solutions most strongly influenced by the first front and gives it a ranking value of 2. On the third front, solutions are largely dictated by the second front.

2.4.2 Approaches to Production Scheduling Problems

The approach to analyze the results of the studied problems is called multi-objective treatment. There are several possibilities for addressing the FJSP-type problem, which can be analyzed by comparing individual objectives or through a multi-objective approach, evaluating all objectives simultaneously.

Goldberg et al. (2016) write that the imitation of natural systems such as biomimetics has inspired computational algorithms to solve various problems. Other ways of solving computer science problems are listed under Natural Computing by Goldberg et al:

Bio-inspired computing: evolutionary computing, swarm intelligence, neural networks, organic computing, immune systems.

Physics- and chemistry-inspired computing, including simulated annealing, harmonic search, gravitational search, and chaotic search;

Natural material-based computing, covering DNA computing, quantum computing, and molecular computing;

Computational simulation of natural systems involving fractal geometry and artificial life.

Other than the above methods, many other approaches are available, and often hybridizations with other methods occur within each cited technique. Techniques to model animal behavior for algorithms are common within bio-inspired computing. Examples include the ACO, ABC, and SPO algorithms. The Artificial Bee Colony (ABC) algorithm was chosen for this task, and its biological basis will be discussed next.

2.5 Biological Basis of the Used Metaheuristic

A bee colony contains three kinds of bees - the queen (a female responsible for reproduction), the males and the worker bees. The queen reproduces, she does this through laying eggs, which can be fertilized or not. Drones have the duty of fertilization of future queens. They develop from eggs that have not been fertilized. They function to boost the DNA of their parent (the queen). Worker bees must maintain the colony. They do jobs such as cleaning, giving food, and putting supplies away. Queens and worker bees come from eggs that were fertilized. These get specific food as larvae after hatching^[9].

Several critical tasks exist within a bee colony, including mating, locating a new nesting site, and foraging. The biological foundation for the method examined in this study is this. A bee that has a job finds a flower, gathers nectar, puts it in its stomach, and then returns to the hive. After putting nectar into previously empty cells, the bee tells other bees about where the food can be located. It shares information about the direction and distance next to the quantity of available nectar. Each bee performs a specific dance in a designated colony area to relay this information. every onlooker bee selects a food source randomly, using the information relayed through dances, until the source is exhausted; then the onlooker bees become scout bees and discover a brand-new food source to exploit; then the

cycle repeats itself.

2.5.1 Artificial Bee Colony Algorithm

The method adopted to solve the FJSP is the Artificial Bee Colony (ABC) algorithm that Karaboga (2005) proposed for solving continuous optimization problems. The algorithm was built around three bee groups.

The employed bees choose food sources randomly. Onlookers hear this information in a flight vibration called the dance. Watchful bees remember the solutions and search nearby (i.e., in the neighborhood) for better solutions until they run out. Depleted solutions are abandoned by the bees, and the employed bee that worked on that food source becomes a scout bee that searches for a new food source (solution) randomly.

2.5.2 Local Search Algorithms

Achieving efficiency and efficacy in solving large-scale NP-hard optimization problems can be almost impossible. Efficiency gain may reduce effectiveness, and the reverse is also true. According to Goldberg and Luna (2016) a true balance between using known knowledge and finding new possibilities must be struck when designing a good search algorithm. The two main ideas are how to use what is known and seek out new information. Utilizing known knowledge allows searching within parts of the solution space. That is done through the problem variables. Rather than exploring, exploration means that the algorithm can find promising areas of the solution space without traversing the entire domain but just focusing on selected regions.

To enhance local search capabilities, local or neighborhood search mechanisms have been developed to assist metaheuristics in addressing complex problems. Below are some local search mechanisms.

Instance Barter: This operator exchanges randomly placed positions in a solution vector. A simple but effective mechanism is shown in Figure 2-1: two positions in the solution vector are chosen at random and their contents are exchanged.

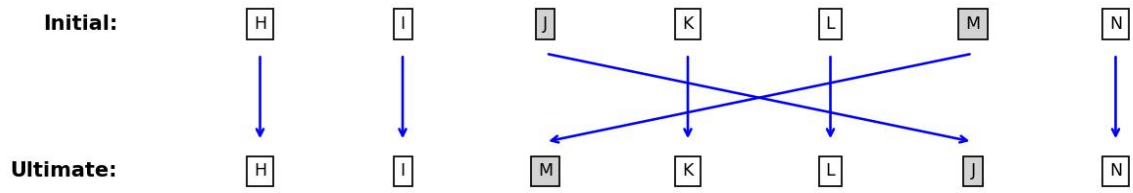


Figure 2-1 Instance Barter

The Reverse function switches positions within a solution vector. This may include the full solution vector or only a part of it. In Figure 2-2 you switch the whole solution vector.

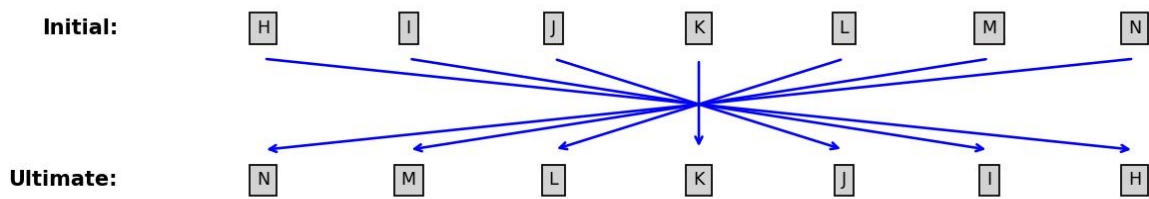


Figure 2-2 Instance Reverse 1

The reverse operator works differently. The vectors are randomly selected at two locations in Figure 2-3. The values between the points flip.

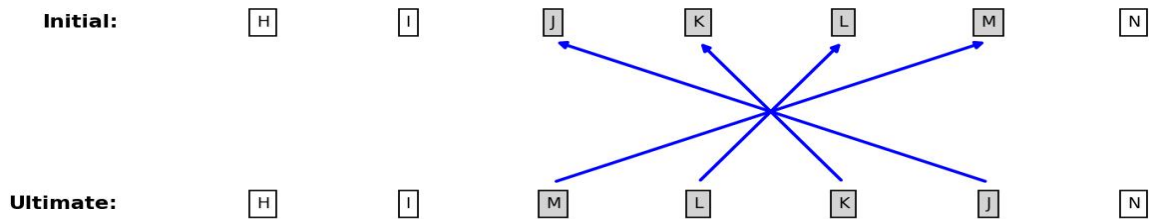


Figure 2-3 Instance Reverse 2

The Insert operator involves randomly selecting two positions in the solution vector. first position provides the value to be inserted into the second position. Given that some problems may impose constraints on duplicate values within the same solution vector, this operator also employs two modes of operation, which are frequently utilized. For the problem addressed in this work, the first mode is shown in Figure 5 and basically executes the process above.

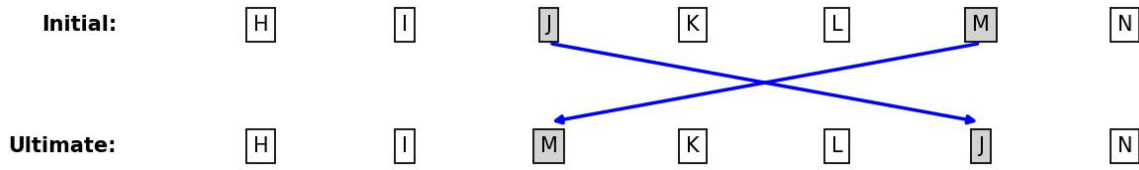


Figure 2-4 Instance insert

For problems whose constraints forbid duplicate values in the solution vector, like the Salesman Traveling Problem (STP), the Insert operator acts like this: Two random positions of the solution vector are chosen. The first value goes in the second position. However, to avoid losing the value initially at the second position, it is shifted to the next position, and subsequent values are displaced sequentially.

The value in the last position wraps around to the first position, continuing this process until the first randomly chosen position is filled, as illustrated in Figure 2-5.

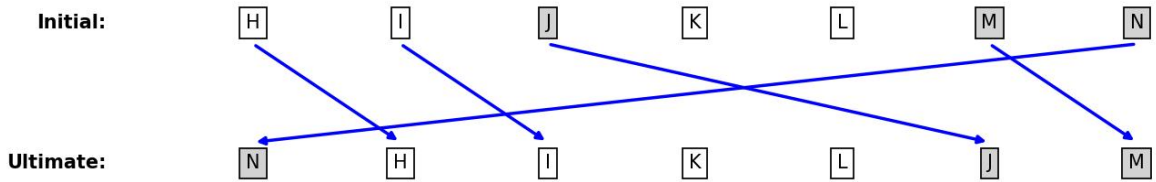


Figure 2-5 Instance in verse 2

The Genetic Crossover Operator is based on species evolutionary processes and involves combining information from two parent vectors. This strategy is considered part of natural and bio-inspired computing^[10]. It operates so that two solutions, First and Second, are drawn from the population. A random number of two positions is selected, and the values between these positions from First and the complementary segment from Second are added together to create an individual.

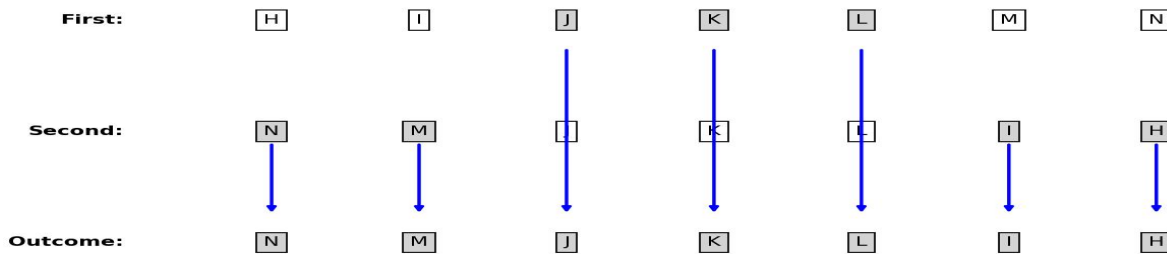


Figure 2-6 Instance of crossing genetics

Genetic mutations change genetic material and increase population diversity stochastically. It mimics natural mutation mechanisms, Goldberg, Goldberg, and Luna (2016) wrote. This is analogous to the Swap operator above. But it also allows for the configuration of a mutation rate - the percentage of positions changed in an individual. Genetic mutation takes three forms^[9].

point mutation: Changing one position in the solution vector.

Block Mutation: Change of a contiguous sequence of positions.

Inversion Mutation: Reverse of a segment of the solution vector.

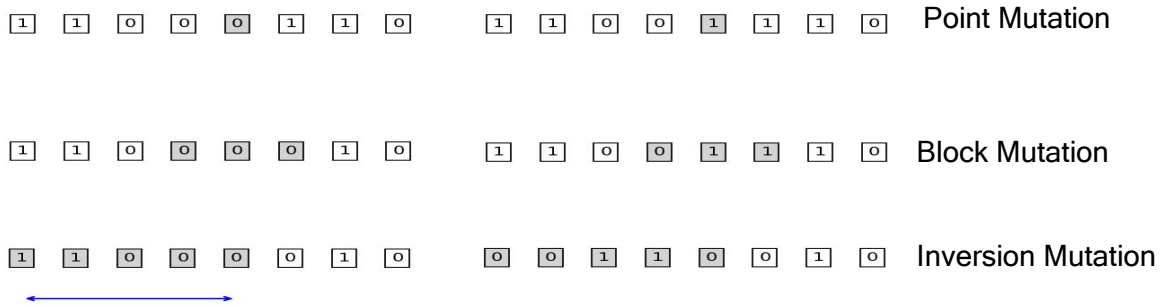


Figure 2-7 Instance of Mutation

In some applications, such operations become complex because they must obey the limitations of the problem. To which they are applied. Regarding this work, if the problem is partially flexible, at least one machine cannot perform all the operations. Not all modifications made by the described operators would be possible.

CHAPTER 3 LITERATURE REVIEW

Herein, various studies from the literature pertaining to the problem or the method adopted to address it are analyzed to provide the theoretical basis for this study. Most of these works were published recently to highlight some of the recent developments in the field.

The publications analyzed in this bibliographic research use a range of ways to attain their objectives, including metaheuristics and local and global search tactics. Several authors limited the make-up time while others reduced waiting times between operations, including preventive maintenance tasks during machine downtime. Some studies have also addressed multi-objective optimization by reducing makespan, overall maximum workload, and the workload of the most heavily loaded machine. Such strategies helped authors refine solutions.

Works adopting a single-objective approach were referenced, considering that the multi-objective approach would be divided into two parts: routing and production scheduling, which would be addressed individually. Another noteworthy point is that applying a multi-objective equation to consolidate the three objectives into a single value (fitness) can partially address the problem in a less complex manner.

The following sections analyze works by renowned authors in the research area relevant to this proposal, contributing to its development. The studies are organized chronologically. Given that multi-objective strategies may decompose the problem into multiple single-objective subproblems and considering this as a viable alternative, the reviewed works are Divided into two categories.: single-objective and multi-objective approaches.

3.1 Solving FJSP with a Hybrid Artificial Bee Colony Approach

Thammano and Phu-Ang (2013) conducted an insightful study to minimize the total execution time of all operations across all jobs. In this work, FJSP was decomposed into

two subproblems: Identifying the order of operations and choosing the best machine for each task.

Five search methods were utilized. The Harmony Search algorithm (Lee & Geem, 2005) was employed for population initialization. The Iterated Local Search technique (Thammano & Phu-Ang, 2013), the Scatter Search technique (Thammano & Phu-Ang, 2013), and the Filter and Fan algorithm (Thammano & Phu-Ang, 2013) were applied to perform local neighborhood searches. Finally, the Simulated Annealing algorithm addressed solutions trapped in local optima. The integration and synchronization of these five methods ensured that the authors demonstrated better algorithmic performance than three other benchmark algorithms.

The results demonstrated that the most well-known values were achieved for two instances across the ten instances tested. In cases where the algorithm did not attain the optimal known values, it closely approximated them and, in most cases, outperformed the other algorithms in terms of efficiency.

Although Thammano and Phu-Ang (2013) adopted a single-objective approach in this study, their strategy of decomposing the problem into two subproblems and addressing them separately significantly improved the development of their proposed methodology.

3.2 Solving FJSP Comparing Various Heuristics

Zheng, Lian, and Mesghouni (2014) explore three distinct heuristics for resolving the FJSP: AG Integrated, ACO, and ABC. Unlike other approaches, the authors integrate these heuristics with an Insertion Algorithm (IA) to address dynamic maintenance scheduling. They utilize IA to schedule preventive maintenance tasks during idle machine periods generated by the FJSP-solving process. This strategy, termed condition-based maintenance, is a preventive maintenance approach to reducing machine downtime.

Combining the three heuristics and the IA was suggested to improve FJSP solutions. Applying the IA reduces uncertainty due to machine breakdowns since preventive maintenance scheduling reduces machine unavailability due to failures. Not only do machine breakdowns create uncertainty, but also employee absence, operational errors, and

equipment problems that cause production delays.

While several variables contribute to machine unavailability, the authors recommend condition-based maintenance to reduce one important source of interruption.

Given the innovative nature of the problem and the absence of established benchmarks to validate the algorithm's effectiveness, the authors compared their IA with the simulated scheduling algorithm from Neale and Cameron (1979), as cited in Zheng, Lian, and Mesghouni (2014), to establish a comparative criterion for the proposed method.

The solution procedure had two stages:

- Pre-scheduling: Solving the FJSP using the three heuristics.
- Maintenance Insertion: Integrating preventive maintenance tasks into the pre-scheduled solution using the proposed insertion algorithm.

The insertion algorithm basically involves finding inevitable idle time intervals in the schedule and scheduling preventive maintenance tasks within those intervals. This requires locating idle periods in each machine's scheduling sequence and inserting the preventive maintenance tasks for each machine. Even if the idle interval is shorter than the required maintenance duration, the task is started at the start of the interval and all subsequent operations are delayed accordingly.

The performance of the three heuristics was evaluated against the best-known results from Xia and Wu (2005) and Neale and Cameron (1979), as cited in Zheng, Lian, and Mesghouni (2014).

The test cases used are from Kacem, Hammadi, and Borne (2002), where problem sizes are denoted by names such as J8M8, representing a scenario with 8 jobs and 8 machines.

Zheng, Lian, and Mesghouni (2014) concluded that the quality of the final scheduling outcome significantly relies on the pre-scheduled Strategy determined during the standard FJSP resolution stage.

Among the five implemented algorithms, the integrated GA achieved results equivalent to those of the Hybrid Algorithm Genetic (HAG), with both outperforming the others. The HGA performed best for the J15M10 instance, while the integrated GA yielded

superior results for the J8M8 instance.

This work is referenced in the present study as it demonstrates the feasibility of employing an algorithm to integrate scheduled maintenance into a production schedule, achieving results that closely approximate the well-known benchmarks Solution of FJSP with BCO and strategies of searches

Table 3-1 Comparison of the results

Algorithm	J8M8	J10M1	J15M1
IGA	14	7	12
ACO	15	7	16
ABC	16	8	15
SOA+SPO	15	7	12
HAG	15	7	11

3.3 Solving FJSP with BCO and Search Strategies

For FJSP, Choo, Wong, and Khader (2016) proposed that the BCO algorithm approach uses a modified two-enhancement scheme using neighborhood N5 (mTPIAIN5). This algorithm uses a brute-force strategy to evaluate possible solutions to optimize scheduling and resource allocation in the job shop environment, which requires high computational time (TP). Local search has two phases: swap and insertion operations, and a minor perturbation phase on the basis of N5 neighborhood structure. Here, the N5 neighborhood is executed by swapping all possible operations on machines along the critical path. But such brute-force is time consuming especially for large problems. For the computational cost, the authors used the bottleneck-shifting heuristic (BSH) to select a machine on the critical path (a "bottleneck" machine) and restrain perturbations to operations on this machine. This method removes the overhead of TESN5 local search.

The local search of the N5 neighborhood exchanges consecutive operations along the critical path. This exchange swaps operations without precedence constraints so that operations on the critical path can be interchanged. Switching operations means

consecutive operations may be reversed.

The N5 neighborhood structure seeks to identify local optima by exhaustively evaluating all possible swaps along the critical path. However, as the number of candidate swaps grows exponentially, this search approach becomes computationally expensive. The authors focused on identifying local optima and choosing the critical path using a bottleneck detection heuristic to address this. This heuristic adjusts particular operations, decreasing the number of changes while keeping manageable complexity even in big cases.

The modified two-enhancement scheme with N5 neighborhood perturbation (mTPIAIN5) is a local search that targets only the bottleneck machine.

The BSH functions by decomposing the FJSP into multiple single-machine scheduling problems. According to the authors, this reduces problem complexity, as solving simpler "one-machine" subproblems is more tractable. To minimize maximum tardiness, the authors employed the due date earliest (DDE) dispatching rule, which shortens the critical path.

Two selection strategies were tested: a greedy strategy and a linear ranking strategy. These yielded two distinct BCO codes integrated with mTPIAIN5:

- BCO with GS-TESN5 (greedy strategy),
- BCO + RLS-TESN5 (ranking linear selection).

Choo, Wong, and Khader (2016) evaluated performance using three codes: BCO with GS-TESN5, BCO + RLS-TESN5, and the "pure" BCO + TESN5 (without modifications).

The proposed algorithms were evaluated through testing. on 82 benchmark instances from the OR-Library (Beasley, 2017), which includes flexible, fully flexible, and partially flexible problems ranging from 6 jobs $\times$ 6 machines to 50 jobs $\times$ 10 machines.

The results demonstrated that the suggested strategies worked:

- BCO-based on GS-TESN5 solved 56% of the 82 problems where the best-known values differed by $\leq 1\%$.
- 54% of problems were solved within the same deviation threshold using BCO + RLS-TESN5.

Concerning computational time (TP):

- BCO GS-TESN5 reduced TP by 10% compared to BCO + TESN5.
- BCO + RLS-TESN5 had 25% less TP than BCO + TESN5.

The authors concluded that both strategies result in much lower average TP for benchmark problems.

While using a single-objective approach, this work shows that bee colony-based algorithms outperform others in computational speed but maintain solution quality.

3.4 Solving FJSP with Artificial Bee Colony and SPO

Muthiah, Rajkumar, and Rajkumar (2016) propose a hybrid approach combining ABC and SPO. This methodology aims to reduce makespan while overcoming the inherent limitations of both algorithms.

The authors classify Artificial Bee Colony as effective in identifying global optima but note its drawbacks, which are familiar to many metaheuristics. In standard Artificial Bee Colony (ABC), solutions that fail to improve after a predefined number of iterations are replaced by scout bees, mimicking biological behavior. Each solution x is associated with a limit- X variable, incremented after n iterations without improvement. Once the limit- X crosses a threshold, the onlooker bee investigating solution X becomes a scout bee that replaces X with a new random solution.

The hybrid technique uses SPO's velocity and position update equations to change from onlooker to scout bee instead of limiting it. The relevant equations for SPO's particle movement mechanics are imported into ABC's framework for a more efficient hybrid process. The velocity and position updates enable dynamic adjustment of solution exploration, enhancing convergence speed and solution quality.

The workflow of the suggested hybrid algorithm is illustrated below.

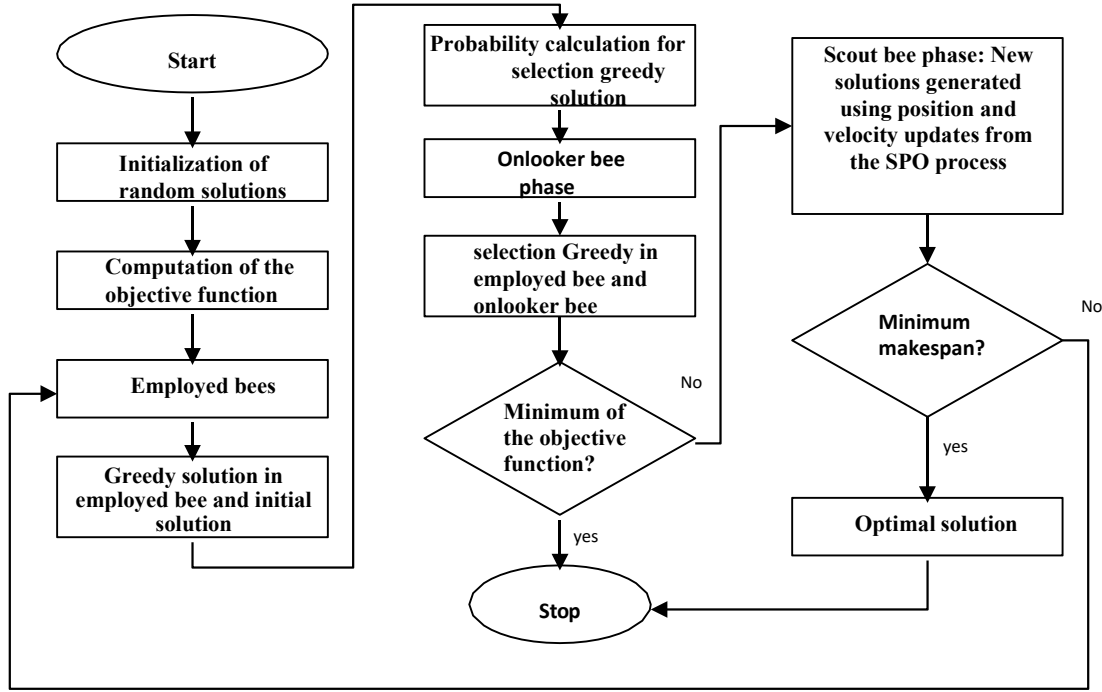


Figure 3-1 Representative Diagram of the Algorithm

The authors' results indicate that combining the Artificial Bee Colony with SPO as a hybrid proposal is a robust strategy compared to other methods.

Though using a single-objective minimization, this work is of particular interest because it hybridizes ABC with another metaheuristic, allowing the algorithm to obtain the best-known results among the benchmark instances used for validation. This work shaped a hybrid approach proposed in the following chapter.

3.5 Solving Machine Scheduling Problems with Sequence Dependent Using Hybrid Artificial Bee Colony

Yue et al. 2016 adopted the Pareto Method for the Hybridizing Artificial Bee Colony (ABC) algorithm for a machine scheduling problem with sequence-dependent setup times. The authors characterized sequence-independent setup time as a scenario where setup duration is dependent only on the current job and sequence-dependent setup time as a scenario where setup time is dependent on both the current job and the previous job. It's this distinction that means an operation is sequence dependent if it depends on the completion of prior operations. Otherwise, it is sequence-independent. This sequence-dependent machine scheduling problem is called the JSP.

The aim was to simultaneously minimize makespan and total weighted tardiness. Developed, tested, and compared against three other algorithms implemented by the authors, the proposed Algorithm was called the Combined Pareto-Based Artificial Bee Colony algorithm (CPBABC).

- Advanced Strength Pareto Evolutionary Algorithm (ASPEA)
- Nondominated Sorting Algorithm Genetic II (NSAG-II) (Deb, Pratap, et al., 2002),
- Swarm Particle Optimization (SPO) (Kennedy & Eberhart, 1995).

Yue et al. (2016) maintained equal numbers of employed and onlooker bees, treating In practice, scout bees randomly select new food sources when the "limit cycle" counter reaches some threshold. The food sources that the algorithm processes have two layers.

- Encoding and Permutation Layer (Layer 1): Decides which machines are assigned to tasks.
- The Job Sequence Layer (Layer 2): The order of operations for each job group is determined.

Figure 3-2 illustrates this approach. In the example, the upper nodes represent machines assigned to tasks (e.g., Machine 1 executes Job 2). In contrast, the lower nodes depict the task execution sequence

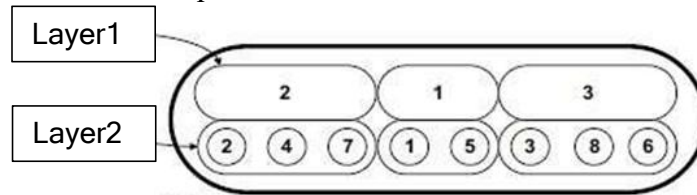


Figure 3-2 Layer Representation.

The starting population of food sources., also referred to as initial food sources, is generated randomly. The sole constraint during this generation is that each food source must be feasible; if not, it is discarded and regenerated randomly. The number of generated food sources equals the number of employed bees, since each bee is assigned one source to explore.

Predefined parameter values influence the Artificial Bee Colony algorithm, similar to

most search algorithms. For parameter tuning Yue et al. (2016) used Taguchi method. This method designs a sequence of experiments in an orthogonal array (OA), where each experiment corresponds to different parameter levels. Every parameter has several values tested so that the optimal parameter set for algorithmic performance can be identified. In practice, the Taguchi method fine-tunes the algorithm to utilize the best parameter values for the target dataset.

The CPBABC Algorithm was tested using the optimal parameter set derived from the Taguchi method. It was executed 10 times alongside the other benchmark algorithms (ASPEA, NSGAI-BEG, and SPO) for comparative analysis.

Figure 3-3, presented by the authors, shows the number of Pareto-optimal solutions obtained by each algorithm across datasets categorized into five groups based on problem size

More Pareto solutions indicate greater algorithmic efficiency in balancing conflicting objectives.

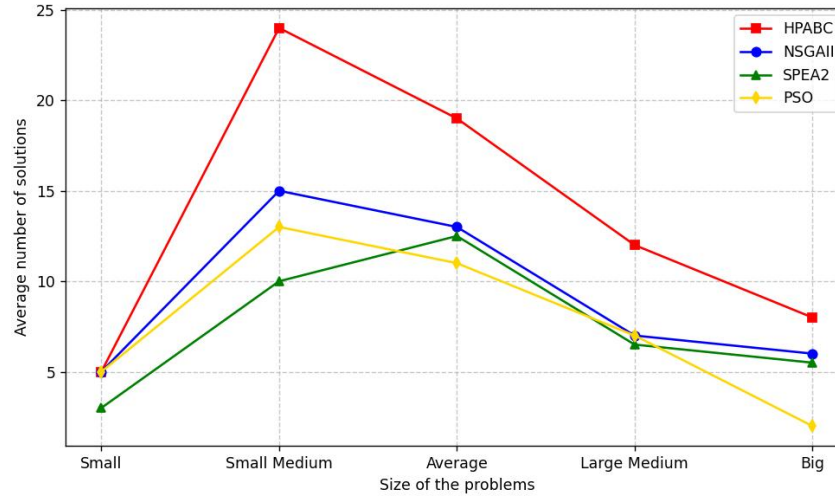


Figure 3-3 Number of Pareto-Optimal Solutions.

The graph in Figure 3-3 shows how many distinct solutions each algorithm found, plotted against problem instance sizes on the x-axis. The proposed algorithm (the red line) gives a larger diversity of solutions than the other benchmark algorithms.

The resultant research shows that the Hybrid Artificial Bee Colony Pareto (HABCP)

algorithm proposed by Yue et al. While the ASPEA, SPO, and NSGAI algorithms occasionally yield marginally better results in specific executions, this merely underscores their lack of repeatability (i.e., inability to consistently achieve identical outcomes across multiple runs). According to the authors, these algorithms exhibit instability in identifying the best-known solutions for large-scale problems. In summary, HPABC consistently outperforms ASPEA, SPO, and NSGAI across all tested scenarios and can generate high-quality solutions for diverse problem categories.

Although this study analyzes two objectives simultaneously, it is categorized under single-objective approaches in this review. The multi-objective section will exclusively feature works addressing more than two objectives.

This study validated the diversity of solutions identified (i.e., the number of distinct solutions achieving equivalent objective values). Further, it supported the decision to hybridize ABC with Pareto principles due to the superior quality of solutions demonstrated by the authors. Works with Multi-Objective Approaches

The following studies adopt a multi-objective framework, addressing optimization challenges with two or more conflicting criteria.

3.6 Hybrid Algorithm Between Artificial Bee Colony, Tabu Search (TS), and Pareto Applied to Multi-Objective FJSP

Li, Pan, and Tasgetiren (2014) addressed the multi-objective FJSP through a two-stage hybrid algorithm (NABCM), integrating ABC, Investigation Anathema (IA), and Pareto optimization. Each heuristic operates in distinct phases: The first stage involves IA doing local search, while the second involves compiling non-dominated solutions following Pareto principles.

Initially, IA optimizes for makespan (C_m), workload total (W_t), and maximum workload machine (W_m) for three objectives. Together, these objectives form a weighted fitness function.

$$\min f = C_m \times \Omega_1 + W_t \times \Omega_2 + W_m \times \Omega_3 \quad (3-1)$$

where Ω_1 , Ω_2 , Ω_3 describe user-defined weights. The second stage employs Pareto

optimization to maintain a set of non-dominated solutions dynamically. This set is a reference for bees to initiate local searches around each solution. If the size of the non-dominated set diverges from the number of bees, each one haphazardly selects a solution to explore using IA-based strategies.

The authors validated the algorithm on benchmark instances from Kacem and Brandimarte, achieving the best-known results for the defined objectives. In their results (Table 3-2), columns f_1 , f_2 , and f_3 correspond to C_m , W_t and W_m , respectively. The hybrid approach demonstrates robustness in balancing exploration (via ABC and TS) and exploitation (via Pareto optimization), ensuring scalability across problem sizes.

Table 3-2 Results of Kacem's Instances

Problem	DABCA		
n x m	C_m	W_t	W_m
4 x 5	11	32	10
4 x 5	11	34	9
4 x 5	12	32	8
4 x 5	13	33	7
8 x 8	14	77	12
8 x 8	15	75	12
8 x 8	16	73	13
8 x 8	16	77	11
10 x 10	7	42	6
10 x 10	8	42	5
10 x 10	7	43	5
10 x 10	8	41	7
15 x 10	11	91	11
15 x 10	11	93	10

The authors also presented detailed results for each of Brandimarte's instances, which can be found in the cited article.

In addition to analyzing results for the aforementioned benchmark instances, the authors applied the algorithm to problems involving preventive maintenance task scheduling, aiming to utilize a portion of machine idle time for performing preventive maintenance on each machine. The results were satisfactory compared to those obtained by other researchers, as shown below.

Table 3-3 Results for Kacem' s instances.

Problem	DABCA			FBS-BOA			HAG		
n x m	Cm	Wt	Wm	Cm	Wt	Wm	Cm	Wt	Wm
8 x 8	17	105	15	18	103	16	17	105	15
8 x 8	18	103	16	-	-	-	-	-	-
10 x 10	8	61	7	8	61	7	9	60	8
10 x 10	9	60	8	-	-	-	-	-	-
15 x 10	12	107	12	-	n/a	-	12	107	12

Table 3-3 reports values achieved by three different algorithms for modified problem instances. They represent cases where Kacem originally benchmarked problems but included machine-assigned preventive maintenance tasks. These maintenance tasks are performed during machine idle periods, which improves the overall result over those typically seen in standard Kacem instances.

3.7 Hybrid Artificial Bee Colony Algorithm Aiming to Present Solution Diversity

The hybrid algorithm proposed by Chao, Lin and Lu (2017) addressed the FJSP, named the Artificial Bee Colony Algorithm with Diversity Index Search (DABCA). This algorithm incorporates a technique called Stepwise Machine Operation Assignment (SMOA). The central aim of this study was to generate a wide diversity of solutions, even if these solutions exhibited identical objective values.

The authors demonstrated a significant diversity of schedules for each result, as shown in Column 5 of Table 3-4. This indicates the algorithm's capability to identify multiple solutions for the same problem. The researchers validated their approach using Kacem's 10x7 and 10x10 benchmark instances.

Table 3-4 Results for Two Kacem Instances.

Problem n x m	DABCA			
	Cm	Wt	Wm	Qty Solutions
10 x 7	15	61	11	22
10 x 7	16	60	12	36
10 x 7	15	62	10	16
10 x 10	8	41	7	66
10 x 10	8	42	5	42
10 x 10	7	43	5	58
10 x 10	7	42	6	34

Table 3-4 indicates a high diversity of solutions obtained through the DABCA. Such diversity reflects identical objective values across different schedules and production plans, indicating that the algorithm can find multiple paths to the same optimal or near-optimal results.

3.8 Genetic Algorithm with Bee Algorithm Applied to Multi-Objective FJSP

Deng, Gong, and al. (2017) studied the Flexible Job Shop Scheduling Problem (FJSP) using a multi-objective optimization approach aimed at minimizing makespan, the workload of the machine that is most loaded, and the workload total among all machines. The proposed strategy employs a two-stage optimization mechanism.

The Nondominated Sorting Genetic Algorithm II with Bee Evolutionary Guiding (NSGAI-BEG) generates the initial population in the first stage. Bees are placed in initial solutions to explore the solution space exhaustively. This stage consists of nine steps, including random population generation and individual fitness calculations for each objective.

The second stage comprises seven steps, refining solutions through genetic operators such as crossover and various mutation types. Bees apply these operators while considering all three objectives. Transitions between Stages 1 and 2 depend on the quality of solutions found. The top n are selected in Stage 1 and divided in Stage 2, creating a binary tournament. Some crossover happens to winners and some happens to losers. The resulting

population is subjected to two types of mutations - combined and evaluated.

The iterations for each Stage are different, 100 for Stage 1 and 200 for Stage 2, which is not usually done in other studies.

The authors achieved nearly all best-known results for Kacem, Hammadi, and Borne's (2002) benchmark instances, except for results 11, 93, and 10, which are considered among the best for the 15x10 instance. Despite this, the algorithm demonstrates significant promise for large-scale, multi-objective FJSP instances.

Table 3-5 reports the researchers' results using the proposed algorithm on Kacem's instances.

Table 3-5 Results achieved

Problem n x m	NSGAI-BEG		
	Cm	Wt	Wm
4 x 5	11	10	32
4 x 5	11	9	34
4 x 5	12	8	32
4 x 5	13	7	33
8 x 8	14	12	77
8 x 8	15	12	75
8 x 8	16	11	77
8 x 8	16	13	73
10 x 10	7	5	43
10 x 10	7	6	42
10 x 10	8	5	42
10 x 10	8	7	41
15 x 10	11	11	91
15 x 10	12	10	93
15 x 10	11	10	95

That study was relevant because the authors applied proposed algorithms to a set of benchmark instances commonly used in the literature and obtained some of the best results. These outcomes will serve as a reference for validating and evaluating the algorithm's efficiency developed in this work.

The authors did not use the Artificial Bee Colony algorithm but bee-inspired evolutionary principles to guide it through solution space exploration. These strategies

allow the algorithm to escape local optima and are implemented in the proposal detailed in the next chapter.

3.9 Hybridization with Other Metaheuristics

In some studies, cited herein, the Artificial Bee Colony algorithm was combined with at least one other metaheuristic. These hybridizations have generally been satisfactory and show that auxiliary metaheuristics may improve algorithmic performance. Thus, this work will also examine such strategies.

Muthiah Rajkumar and Rajkumar (2016) proposed a hybrid Artificial Bee Colony and SPO approach to solve the FJSP in a single-objective context by minimizing the makespan. They outperformed standalone ABC or SPO implementations.

Li and Peng et al. (2017) introduced a hybrid ABC and Investigation Anathema (IA) method, matching the most well-known results for 12 out of 13 benchmark instances. Their comparative analysis of seven algorithms highlighted the hybrid approach's robustness, with only one instance failing to reach the optimal benchmark.

Li, Pan, and Gao (2011) developed a multi-objective hybrid algorithm integrating ABC with Pareto optimization. Applied to five instances, their method achieved the best-known values at the time, underscoring the promise of hybrid strategies for FJSP.

Li, Pan, and Tasgetiren (2014) extended this field by combining ABC, IA, and Pareto principles. Tested on five Kacem instances, their hybrid algorithm met or exceeded benchmarks, demonstrating the power of multi-metaheuristic integration.

Still many researchers use hybrid approaches to address diverse FJSP instances often with state-of-the art results. From these successes, the current work will adopt a hybrid methodology to optimize scheduling outcomes.

3.10 Final Considerations of the Chapter

Taking advantage of this targeted review of FJSP solution methods and metaheuristics, Chapter 4 will propose a hybrid ABC-Pareto algorithm based on promising results in previous studies.

The exploration targets include Pareto optimization and local search which have

shown promise to improve solution quality and diversity. The proposed approach is to combine ABC as the primary metaheuristic and Pareto methods for balanced exploitation/exploration in order to deal with multi-objective FJSP.

The hybrid framework will extend existing benchmarks by leveraging synergies between metaheuristics for better computation and solution resilience.

CHAPTER 4 WORK PROPOSAL

This work proposes a method for solving the multi-objective FJSP by using the Artificial Bee Colony metaheuristic modified for discrete algorithms and adding auxiliary metaheuristics to achieve the goals of multi-objective FJSP resolution.

The algorithm's development is evaluated against the objectives outlined in the proposal and validated by applying various well-known benchmark instances, such as those by Kacem, Hammadi, and Borne (2002) and Brandimarte (1993). Applying such instances permits comparison with results achieved by other researchers using the same benchmarks.

The literature identifies some critical issues in FJSP manufacturing environments involving scheduling and routing operations:

- Computational costs of production scheduling should be reduced.
- Multiple reliable schedules must be generated to avoid resource waste.
- The need to handle complex subproblems related to scheduling and routing

activities within acceptable time limits.

Considering these challenges, a proposal has been developed to address the current needs effectively.

The ABC metaheuristic is proposed to address the first challenge, as it operates with heuristics and incurs lower computational costs than exact methods. This proposal is based on the works of Muthiah, Rajkumar, and Rajjumar (2016); Zheng, Lian, and Mesghouni (2014); Choo, Wong, and Khader (2016); Yue et al. (2016); Li, Pan, and Tasgetiren (2014); Chao, Lin, and Lu (2017); Deng, Gong et al. (2017); Li, Pan, and Gao (2011); and Gao et al. (2015). These authors demonstrated in their studies that using metaheuristics for solving complex problems is highly efficient from a computational perspective, and the results are usually satisfactory.

A results presentation strategy similar to that used by Yue, Guan, et al. (2016) addresses the second challenge. The authors demonstrate the number of solutions with identical objective values the algorithm can generate. This approach allows production

managers to have a wide variety of schedules with the same desired objectives, enabling them to allocate workloads differently across machines, such as having one machine work more or less than another without impacting overall costs.

A proposed modeling approach separates the FJSP into routing and scheduling subproblems to address the third challenge. This modeling technique is similar to those used by several authors, such as Thammano and Phu-Ang (2013), Zhou et al. (2011), Li, Pan, and Gao (2011), Li, Pan, and Xie (2011), Gao et al. (2015), and Li, Pan, and Tasgetiren (2014). These authors present the two subproblems using two distinct vectors: one for routing and another for scheduling. Both vectors are required to obtain a complete solution.

The proposed modeling introduces a pre-scheduled routing vector that is initially given enough schedule to represent a feasible solution. Aside from this routing vector, the model includes an operation exchange matrix for dynamic rescheduling of operations if required. This is a strategy different from that reported in the literature and will be described further in this chapter.

For the routing problem, a mutation operator is suggested. This operator is built for the employed bee search phase, the onlooker bee search phase, and the scout bee search phase of the algorithm. By integrating the mutation operator during these stages the algorithm ensures continuous refinement and optimization of routing solutions.

For the scout bee phase, the Pareto Dominance metaheuristic proposed by Li, Pan, and Tasgetiren (2014) is used. A batch of dominant solutions forms a Pareto archive from which scout bees can select the best solutions. To identify these dominant solutions, a function adds the values of the multiple objectives to a single value, transforming the problem into a single-objective context. This lets the algorithm focus on optimizing a single unified metric while still allowing multi-objective exploration.

While routing vectors are initially built with pre-scheduled solutions for feasibility reasons, the best-known solutions are obtained during the routing vector's final scheduling adjustments. This final scheduling is carried out by an operation exchange matrix that modifies operations scheduling dynamically to optimize the solution. Using this approach,

algorithms can improve solutions beyond the initial feasible state.

Yue, Guan, and others (2017) used the Pareto Frontier technique. Results are presented in 2017. Graphs representing this technique are displayed graphically to give an intuitive representation of the quality of results in comparison with those reported in the literature. Seeing how the Pareto Frontier performs illustrates how the algorithm can achieve multiple goals simultaneously.

Through this research, considering the proposed techniques for the ABC approach, it is expected that the results will be of equal or superior quality to those found in the literature for the instances of Kacem, Hammadi, and Borne (2002) and Brandimarte (1993). Integrating advanced strategies and optimizations aims to achieve state-of-the-art performance, ensuring the proposed method stands out in addressing the Flexible Job Shop Scheduling Problem.

Figure 4-1 shows the main routine of the code in a non-detailed way to help understand the basic steps of the code described next. The flowchart only serves to indicate which part of the algorithm each step belongs to. This visual representation helps understand the overall algorithm structure and flow.

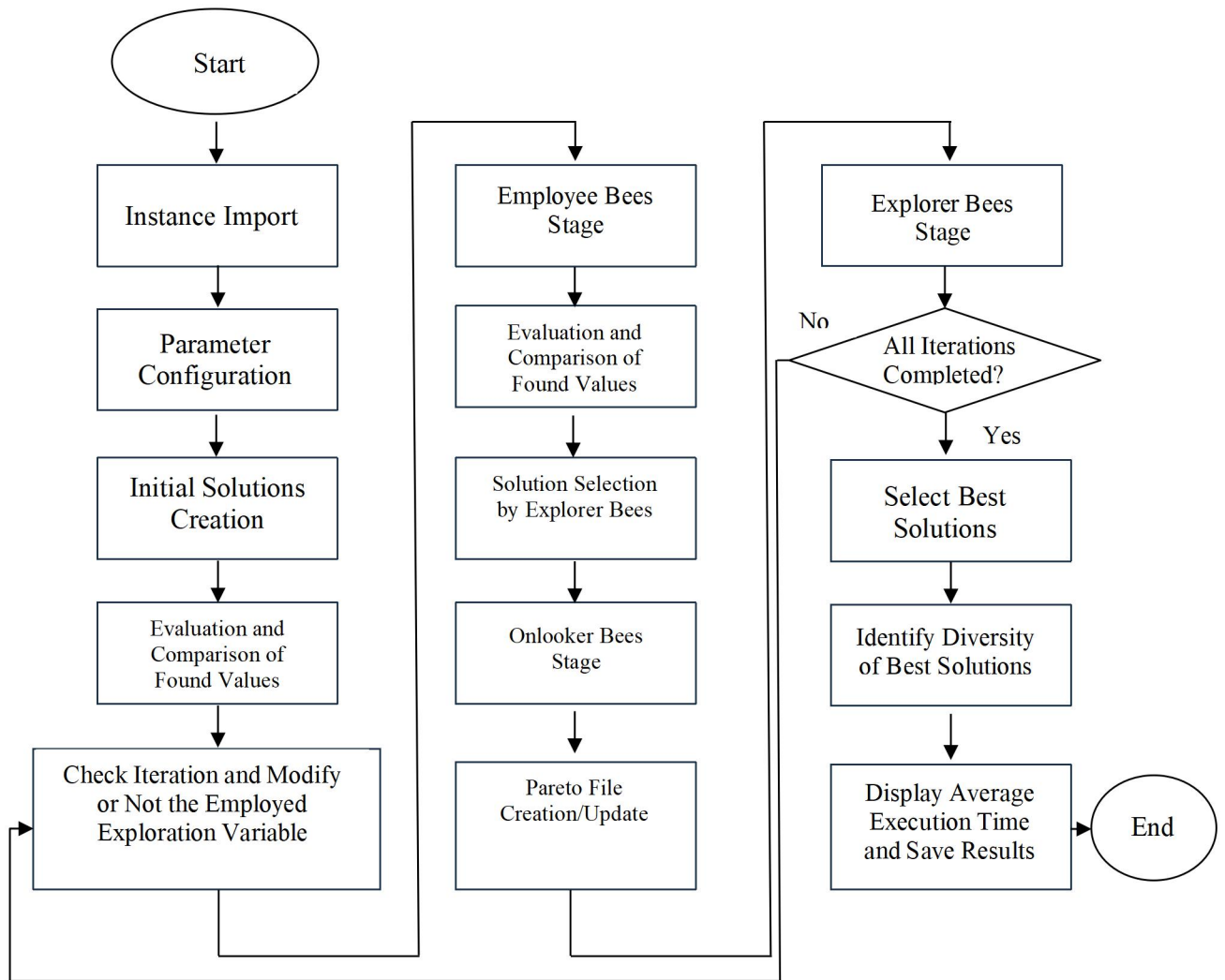


Figure 4-1 General Flowchart of the Code, Representing the Main Function

4.1 Description of methods and strategies used

This section details the strategies used to develop the algorithm. The benchmark data used for optimization is first imported. These instances are loaded as matrices into the program memory through some function. Table 4-1 illustrates how the imported 4x5 instance is structured: 4 jobs, 5 machines

Table 4-1 Import of Processing Times for the 4x5 Instance.

Job	Operations	M1	M2	M3	M4	M5
Job1	Op1	2	5	4	1	2
Job1	Op2	5	4	5	7	5
Job1	Op3	4	5	5	2	3
Job2	Op1	2	5	4	7	8
Job2	Op2	5	6	9	7	6
Job2	Op3	4	5	4	54	5
Job3	Op1	6	1	2	7	4
Job3	Op2	5	2	5	2	4
Job3	Op3	6	5	1	2	7
Job4	Op1	1	5	2	4	12
Job4	Op2	5	1	2	5	9

The row and column headers were not imported but were added to the figure for guidance.

4.1.1 Configuration of Code Adjustment Parameters

Here all necessary parameters of the algorithm stage are defined. Such configurations let the code handle different instances that have different characteristics.

This section details the essential configuration parameters required for the algorithm's functionality. Parameters related solely to visual aspects, such as data output modes and object presentation, are briefly explained in Table 4-1.

The code lets you specify how many times the algorithm will be executed. This allows collecting results from multiple runs and creating statistical graphs. While this parameter has no predefined maximum limit, the code design ties it to the free RAM of the computer. Some modifications might remove this limitation and the code would run without execution constraints.

Variable t is the number of iterations that the algorithm will go through.

parameter N_Bee defines the number of bees used in the algorithm. This number is generally proportional to the size of the problem being addressed, with more significant issues requiring more bees to achieve good results.

The parameter $Limit$ configures the "no improvement" limit, determining how many iterations a solution is explored without improvement before being abandoned. Based on

tests across various instances, this value typically ranges between 15 and 20.

Employed Exploration is a variable whose value decreases during iterations. The search behavior of employed bees is adjusted between exploration and exploitation. Increased employment value enhances the exploitable capability of employed bees. This parameter is initially set at around 20 but drops to 2 after 90% of iterations are completed. At this point, those who are employed bees shift from exploitation to exploration, while scout bees handle exploitation by perturbing solutions. The value of Employed Exploration determines how many changes are made per solution at a time, a strategy detailed in the employed bee phase.

Whether the exploration variable changes from exploitation to exploration during iterations or remains fixed is determined by the parameter dynamic employed exploration. Assume set to "yes," employ exploration changes over iterations. If true, employment remains constant throughout the execution of the algorithm.

The parameter LFP (Pareto File Size) specifies the size of the Pareto archive. to be created. This variable is typically configured with values around 0.2, corresponding to 20% of the total bees. This value is based on literature and empirical testing, which concluded that extensive archives introduce unnecessary computational overhead. The Pareto archive is always populated with the best solutions found (top 20%). However, if the archive size is increased, lower-quality solutions may also be included, which can degrade performance.

To address potential stagnation issues in certain instances, the parameter PerturbScout was introduced. This parameter is set to a value greater than 0 when scout bees are desired to perturb the selected Pareto solutions. The solutions used by scout bees are drawn from the Pareto archive, which may or may not be perturbed upon selection. The value of PerturbScout represents the percentage of operations that will be perturbed by scout bees, as illustrated in Figure 4-2.

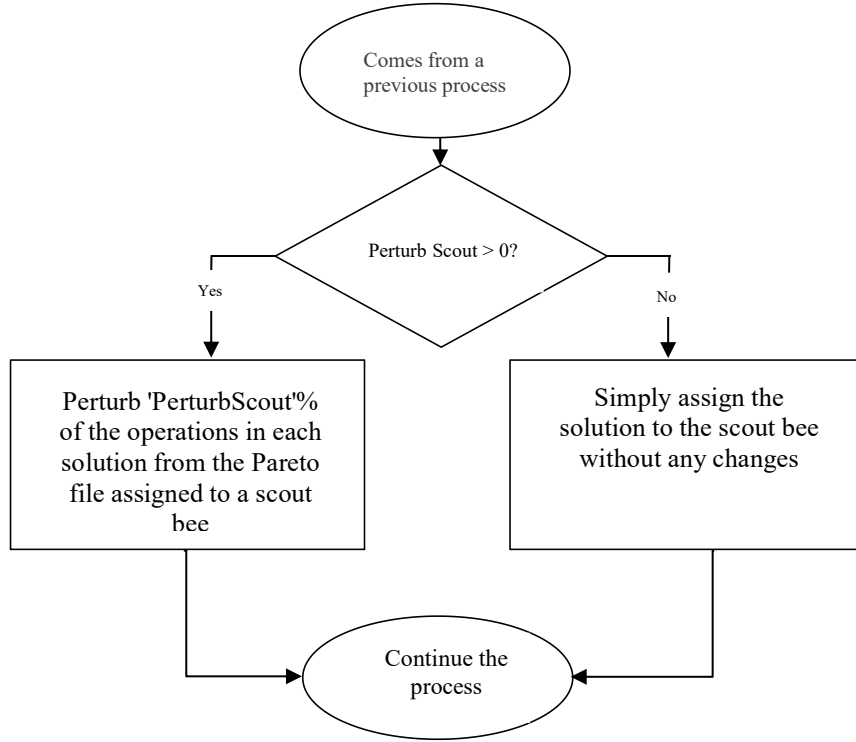


Figure 4-2 Flowchart of Solution Perturbation Process

The parameters below configure a search system for onlooker bees. This search method has three values: M1, M2, and M3. The sum of these values must equal 100 (representing 100% of operations). These values define the percentage chance of onlooker bees performing their local search in the three search modes. The functionality of these modes is described in detail in the onlooker bee phase.

WCm, WWt, and WWm are the fitness function weights. Each weight corresponds to an optimization objective. These are set differently for each problem and thus influence how the algorithm prioritizes goals depending on the particulars of each instance.

The algorithm also has settings for result presentation, such as auditory signals to indicate when specific results are achieved and configurations for displaying the progression of iterations. However, these settings do not affect solution quality, and the associated variables have been omitted from this discussion.

After completing the initial configurations, the algorithm loads the data for the instance to be processed and calls the function that generates the initial solutions.

4.1.2 Creation of Initial Solutions

The starting solutions have been successfully developed, paving the way for further progress and innovation through a function that creates a probabilistic roulette wheel to distribute machine times for each operation. Initially, this roulette identified and excluded the slowest machine (the worst machine for executing the operation), ensuring it never participated in the initial solutions. However, after guidance and testing, it became clear that this was not the best approach. The proposal was revised, and the current method penalizes the worst machine by adding 10% to its processing time.

Example: If a machine has a processing time of 10 units and is the slowest, its time is considered 11 units (10% more) for the roulette distribution. This strategy reduces the likelihood of selecting this machine but does not eliminate it, as in the previous version.

The pseudocode for this step is as follows:

- Select operation i , identify the machine with the worst time and penalize it by adding 10% to its value.
- Identify possible machines and their respective times for executing the operation.
- Distribute the times on a probabilistic roulette wheel, assigning a more significant share to the shortest time, and randomly select the machine to execute the operation.
- Repeat step a for all solutions and all bees.
- End.

This is adjustable, but tests have shown good results with a 10% penalty. In the initial solutions, tests were conducted for machines that were the worst, with penalties ranging from 10% to 60%.

4.1.3 Function fitness

Functional fitness makes purpose management easier. This procedure is executed whenever a solution is changed, with the first execution occurring after creating the initial solutions. The function calculates the weighted sum of normalized objectives, as shown in

Equation 6:

$$E_z = C_{m_z} * w_{cm} + W_{t_z} * w_{wt} + W_{m_z} * w_{wm} \mid \{C_m, W_t \text{ and } W_m\} \text{ are normalized (4-1)}$$

Where:

- E_z : Effectiveness of the solution z .
- C_{m_z} : Makespan objective of solution z .
- W_{t_z} : workload Total objective of solution z .
- W_{m_z} : Maximum workload machine objective of solution z .
- w_{cm} : Predefined weight for makespan values.
- w_{wt} : Predefined weight for workload total values.
- w_{wm} : Predefined weight for maximum workload machine values.

The normalization of $\{C_m, W_t, W_m\}$ ensures that larger values do not dominate smaller ones, balancing the influence of each objective. This strategy simplifies problem handling but makes tuning weights more difficult.

4.1.4 Aggressiveness of Changes by Employed Bees (Exploitation vs. Exploration)

This stage of the code tracks the number of iterations. Employed bees gradually decrease exploitation and increase exploration as the algorithm progresses. This strategy means solutions are changed less frequently over time, focusing instead on neighborhood exploration. This helps the algorithm get good results faster.

4.1.5 Solution Representation

One vector may represent the complete solution. This vector contains the machines assigned to perform operations and their order of execution. Operations are represented by the indices of the columns where the machines are stored. Table 4-2 provides an illustrative example of a solution vector.

Table 4-2 Complete Vector of a Solution

Machines	3	4	1	1	2	2	2	5	4	1	3	4
Positions from the execution	7	1	4	11	2	8	12	5	3	9	6	10

A solution is modified in two ways. The first involves changing a machine in the first

part of the vector, making sure that the replacement machine can work in situations where it is not completely flexible. The second method uses the swap operation in the second part of the vector to change the scheduled operations order. But this strategy should take into account problem constraints like the order of operations within each job.

4.1.6 Employed Bee Phase

For this phase, a roulette wheel is proposed to distribute the machines that can modify the initial solutions. The count of worker bees matches the set of initial solutions, meaning a single bee is assigned a solution to alter. The operation of this roulette wheel is analogous to that used for generating initial solutions, with the key difference being that it does not penalize the worst machine, allowing all machines to have equal proportional chances of selection. Figure 4-3 presents a flowchart of the operations performed in this stage.

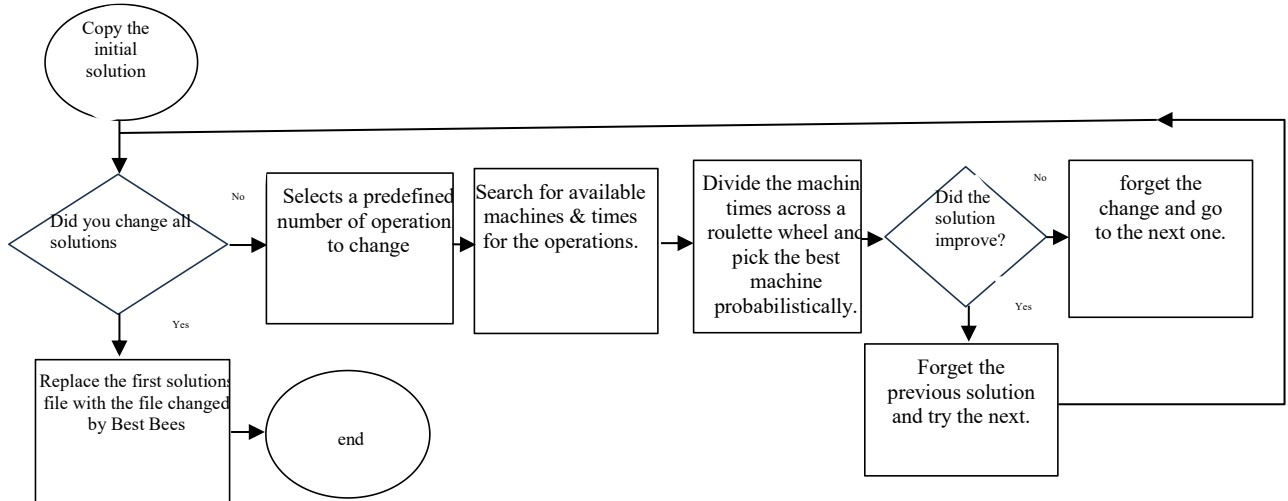


Figure 4-3 Flowchart of representation from the stage of the Bees employees

Although this stage is easily explainable, its operation is slightly more complex. This part of the code considers partially flexible instances and assigns operations only to machines capable of executing each process (feasible machines). This feature eliminates the need for additional treatment of solutions generated by employed bees, slightly increasing execution speed for instances with these characteristics.

The group of employed bees is also responsible for verifying whether the solutions found are better than the previous ones. When the new solutions perform better, they are

saved, while the older ones are eliminated. Conversely, if the new solutions are inferior, they are not recorded, and the "Limit" counter increases individually for each solution. The assessment of whether a solution has shown improvement is carried out using the fitness function, as previously detailed.

4.1.7 Onlooker Bee Phase

The onlooker bee phase occurs after the employed bees assign solutions they have found. This assignment uses a roulette wheel that divides the fitness values of the solutions inversely - the lower the fitness, the bigger the share of the roulette wheel. In this case, fitness is the solution, so the problem is easier to handle than if we consider the three objectives separately.

The fitness is generated similarly across all stages of the algorithm, so the equation will not be reiterated each time it is mentioned.

For visualization of this proposal, Figure 4-4 shows the roulette wheel with percentages for 20 solutions to be used for solution assignment. Values generated are for illustration purposes, since a typical problem has more than 20 solutions, and a detailed illustrative presentation is impossible.

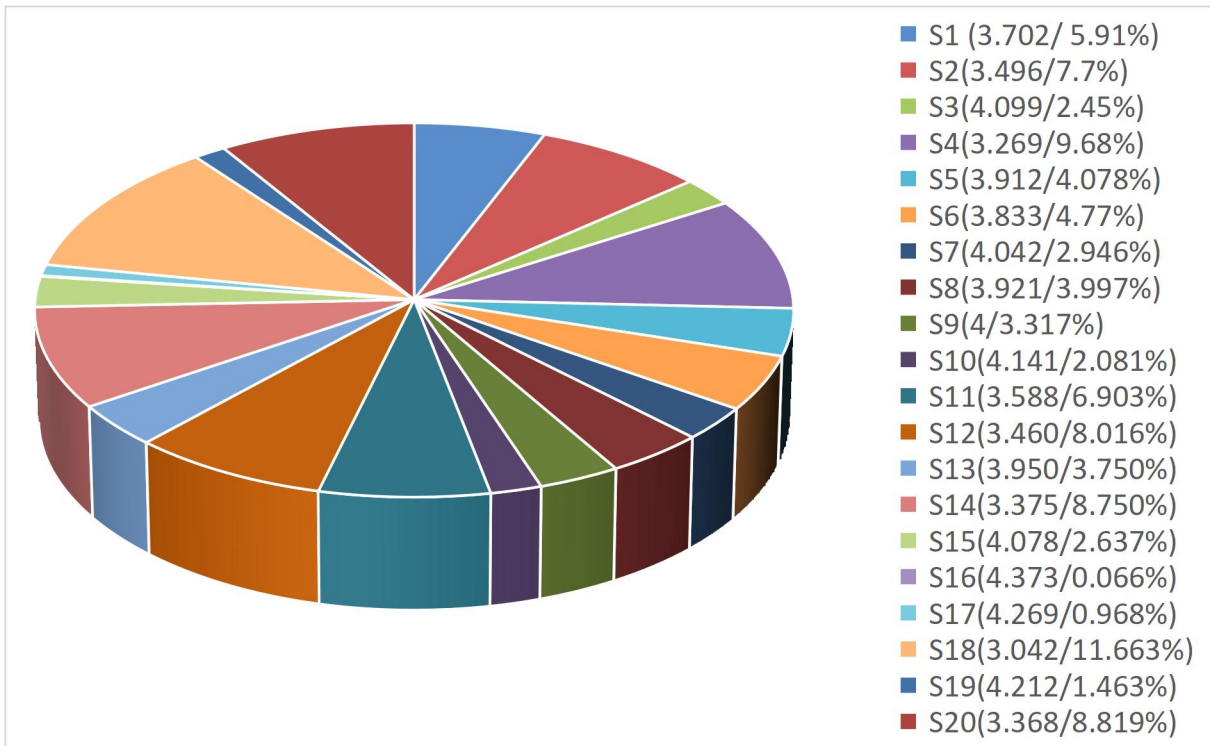


Figure 4-4 Illustrative Probabilistic Roulette Wheel

In this probabilistic roulette wheel, all solutions are distributed with slices proportional to their quality. The more effective the solution, the better the odds that an onlooker bee will select it.

Figure 4-4 illustrates that the worst solution among the 20 represented is S16, and the best is S18. Solution S16 has the highest fitness value of 4.373017 and a selection probability of 0.0664%, while solution S18 has the lowest fitness value of 3.042553 and a selection probability of 11.6636%.

After distributing all fitness values on the roulette wheel, each onlooker bee is assigned a solution through a random draw, allowing the exact solution to be assigned to multiple bees. Consequently, the best solutions are assigned to multiple bees and are thus explored more extensively than the poorer solutions.

The search system for onlooker bees differs from that of employed bees. It consists of three local search methods, each with configurable probabilities of being selected. These methods are randomly chosen, with the probability of each being defined at the start of the code. All three methods are configured differently depending on the needs and particulars of the instance.

The constraint for this configuration is that the sum of the three values must equal 1, representing 100% of the onlooker bees' searches.

These are the priorities for the three search methods:

Method 1: Probabilistic Roulette with Three Best Machines.

With this method, you find the three best machines and then divide their processing times in proportion on a roulette wheel to select one. The chosen machine is always one of the three best among the feasible machines. The roulette wheel for this stage is illustrated in Figure 4-5.

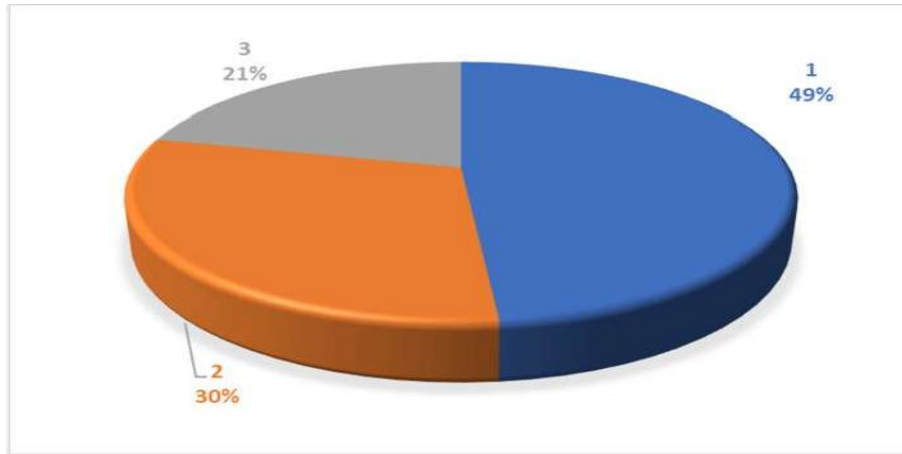


Figure 4-5 Illustrative Roulette Wheel for Onlooker Bee Search

The times distributed to generate this roulette wheel are 2, 4, and 5, corresponding to indices 1, 2, and 3, respectively.

Method 2: Best Machine by Execution Time, with Tiebreaker by Least Loaded Machine

As the name suggests, this method selects the feasible machine with the shortest execution time, regardless of its current workload. However, suppose two or more machines share the best execution time (e.g., two machines with a processing time of 1 unit). A tiebreaker criterion is applied: selecting the least loaded machine among the best. This strategy significantly increases convergence speed but may introduce bias when one machine is significantly faster.

Method 3: Least Loaded Machine: This method selects the least loaded machine to execute the operation. It does not consider any other criteria and counterbalances the bias of Method 2. While Method 2 favors the fastest machine, Method 3 prioritizes the least loaded one, ensuring the algorithm does not always select the same machine for all operations.

The choice of method is probabilistic, determined by the onlooker bees. Three variables

(M1, M2, and M3) define the probability of each method being selected, and these are configured at the start of the code. The configuration varies by instance, as each instance has unique characteristics that influence the optimal settings.

The onlooker bees also verify whether the solutions found are better than the previous ones. If a solution improves, it is stored, and the previous one is discarded. If the solution

does not improve, it is not memorized, and the "Limit" counter is incremented individually for that solution. The "Limit" counter is tied to the solutions, meaning if a solution does not improve after modifications by employed bees, the counter is incremented by 1. The counter is raised to 2 if the solution does not improve after the onlooker bee phase. If it reaches the predefined limit, a scout bee changes the solution using procedures described later on.

4.1.8 Pareto Archive

This Pareto archive is generated when onlookers finish searching. It represents a fraction of the ultimate solutions achieved so far. Its size depends on the LFP variable predefined at the start of the algorithm. That configuration produces a fixed size set that is constant throughout execution.

The archive is composed after ranking the solutions found by the onlooker bees. The scout bees save and use these solutions when searching for new ones.

Since the proposal deals with multi-objective optimization, the ranking is built on the fitness of the solutions. During preliminary tests, the archive size typically ranged between 10% and 30% of the total amount of bees. The archive containing 20% of the bees achieved the best preliminary results.

One strategy to speed up the algorithm in the early iterations is to initialize the Pareto archive after a specific number of iterations. The scout bees only use the archive after n iterations, where n is the predefined number of Limit changes. In other words, if a solution does not improve within n iterations, it is changed by a solution found by a scout bee. This approach ensures that several unnecessary procedures (e.g., solution ranking, archive construction) are avoided if the Pareto archive is generated too early.

To clarify the explanation above, Figure 4-6 illustrates the flowchart of the described strategy.

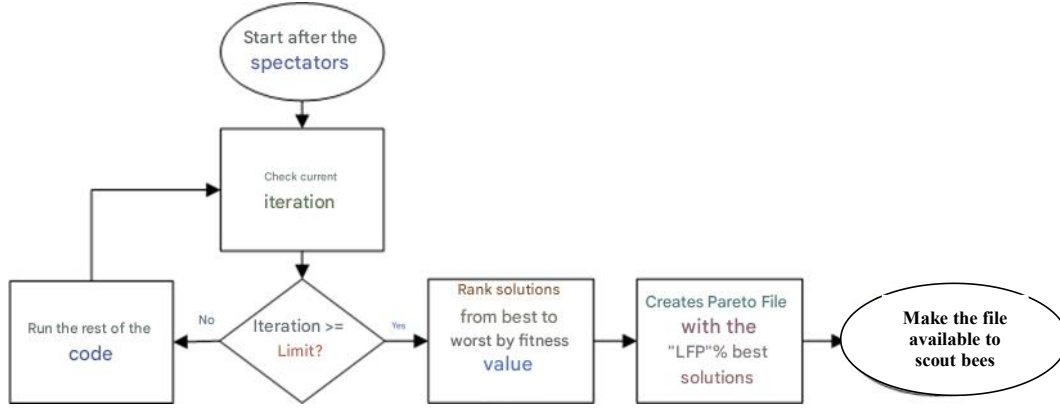


Figure 4-6 Flowchart from the stage of creation of file of solutions Pareto.

To clarify the flowchart above, this formula describes the strategy

$$S_i = \{s_{(1)}, s_{(2)}, \dots, s_{(n)}\}$$

$$\text{so that } F(s_{(1)}) \leq F(s_{(2)}) \leq \dots \leq F(s_{(n)})$$

Then define the “Pareto file” archive $P(i)$ by

$$P(i) = \begin{cases} \emptyset, & i < L \\ \{s_{(1)}, s_{(2)}, \dots, s_{(\alpha n)}\}, & i \geq L \end{cases}$$

Were

- i = current iteration,
- L = the “Limit” iteration at which you switch modes,
- S_i = current population of solutions at iteration i ,
- $F: S_i \rightarrow \mathbb{R}$ = their fitness values,
- $\alpha \in (0, 1]$ = the fraction of top solutions you keep (e.g., “LFP%”).

Before iteration L , the Pareto archive $P(i)$ remains empty. Once iteration L is reached and thereafter, the current population of n solutions is sorted in ascending order of fitness so that $s_{(1)}$ is the best, $s_{(2)}$ the second best, and so on and the top αn solutions (where α is the chosen fraction) are selected and stored in $P(i)$.

4.1.9 Scout Bee Phase

This stage primarily concerns replacing solutions that have not improved over n iterations (where n denotes an arbitrary number of iterations defined by the limit variable).

This stage proposes two configurable operational modes.

The first mode is for scout bees to select a solution directly from the Pareto Archive without altering it. This approach uses the Pareto Archive as a reference set for solution replacement.

The second mode employs the first approach coupled with a predefined perturbation mechanism. In this case, scout bees select a solution at random from the Pareto set and apply a perturbation to specific components of the chosen solution. The perturbation magnitude is governed by the *PerturbScout* variable, which specifies the percentage of operations (or variables) to be altered within each solution. A perturbation rate of approximately 1% during preliminary testing yielded satisfactory results, as detailed in the subsequent chapter.

4.1.10 Strategy for Identifying Multiple Solutions for the Same Objective Values.

Following the solutions processing by the three bee types (scout, employed, and onlooker bees), The algorithm evaluates and verifies the results thus far. It is proposed that the strategy should identify predefined target values for the objectives rather than pursuing the globally optimal solution (though the system can be reconfigured for the former). Checks are periodically conducted in iterative phases to ensure that target values for each objective have been reached. These checks involve comparing the current solutions against the predefined objective thresholds.

Imagine solutions satisfying or exceeding predefined objective values are found at any iteration or during intermediate checks between algorithmic stages. The algorithm then interrupts its normal workflow to log the result. This logging saves the current state of all variables, produces a Gantt chart for the identified solution and records metadata such as:

- The number of iterations until the target value was reached.
- The type of bee that found the solution is the honey bee.
- The computational time required is, in seconds.
- Values obtained for the three objectives.

While the algorithm reports and saves these findings for detailed post-hoc analysis of variable states, it does not terminate. Instead, it dynamically reduces one of the objective

thresholds. This starts the search again for improved solutions (with lower objective values) or alternative schedules that satisfy the same objective criteria.

When not configured with predefined target values, the algorithm minimizes the objectives to the greatest extent possible and reports the best solutions found during execution. Hence, target values are not mandatory for the algorithm's operation.

Preconfigured minimization criterion takes as input the Makespan or Workload priority objective for minimization. Upon meeting target thresholds, these values determine which goal gets prioritized for reduction. The pseudo-code following illustrates the logic that causes this behavior:

- a) If solutions meeting or exceeding all three objectives' target values are found:
- b) Retrieve the minimization priority from the Minimization Criterion variable.
- c) If Minimization Criterion = Makespan, do:

Target Makespan = Achieved Makespan 1.

- d) If Minimization Criterion = WorkLoad, do:

Target WorkLoad = Achieved WorkLoad 1.

- e) End.

This logic prevents the algorithm from stopping prematurely when the first valid solutions are found. Without this incremental target reduction, the algorithm would repeatedly pause when it found a valid solution and would report the same values redundantly. Since the target for the prioritized objective is dynamically lowered, the algorithm finds better solutions while continuing to search. Importantly, although only superior results are displayed, all solutions - even suboptimal - are stored for retrospective analysis. This comprehensive log is important to quantify the number of viable solutions found during the search

4.1.11 Results Evaluation

The proposed evaluation methodology compares the algorithm's achieved objective values with predefined targets. This verification, which is done thrice per iteration, checks that the results meet or exceed the intended objectives. With this strategy, if an employed

bee spots a solution matching the target criteria, that solution is flagged and presented immediately without going through subsequent algorithmic stages.

4.1.12 Result Visualization Methods

This framework proposes two primary visualization formats for solutions presentation. The first type of chart used is Gantt charts, commonly used to represent the scheduling of jobs and machine assignments over time. The second is the Pareto frontier plot, a scatter-plot format showing non-dominated solutions. These plots allow visual comparison of the algorithm results with known state-of-the-art benchmarks from the literature, indicating the quality and competitiveness of the proposed solutions versus known alternatives.

4.1.13 Validation of Found Results

The algorithm outputs were validated using two different methodologies. The first involves manually applying FJSP principles to generated Gantt charts for small-scale instances with limited operations. Metrics such as makespan (visually identified as the total schedule length), workload total, and maximum workload machine were manually computed, minus machine idle times. This laborious process has confirmed the consistency of scheduling logic in more minor cases.

Second, the algorithm's Gantt charts for small instances are contrasted with those from earlier research. Although less time-consuming than manual validation, this approach was similarly limited to small-scale instances to ensure tractability. Both methods confirmed the algorithm's ability to generate valid and competitive solutions consistent with theoretical expectations and empirical benchmarks.

CHAPTER 5 Results and Experimental Analysis

The section offers an empirical evaluation of the suggested strategies described in the previous chapter using benchmark instances based on the work of Kacem, Hammadi, and Borne (2002). It is a somewhat dated dataset but an established reference in FJSP research. The algorithm was executed 30 times to compute the average execution time.

Examples of benchmark instances used for comparison are those presented by Kacem et al. They include problems of various sizes - from small configurations (5 machines, 4 jobs, and 12 operations in Kacem's cases) to large ones (14 machines, 20 jobs, and 240 operations in Brandimarte's cases).

The experimental parameters were defined based on the methodologies reviewed in Chapter 3 and insights gained through iterative testing and analysis of established techniques described in the literature.

4 key parameters characterized algorithmic performance.:

- Comparison with Prior Studies: Results were contrasted against those reported by researchers who applied at least one of the techniques integrated into the proposed algorithm.
- Outcomes were analyzed in relation to other methodologies in cross-technique benchmarking.
- Assessment of solutions: The variety of solutions produced by the algorithm was compared with the diversity reported by known algorithms in the literature.
- Optic frontier positioning: quality of solutions was visualized by Pareto frontier plots for direct comparison with known optimal or near-optimal solutions.

Computational execution times were excluded from cross-study comparisons due to disparities in hardware configurations across research teams. However, aggregate execution times for 30 runs are provided in subsequent tables for transparency.

Following sections describe the experimental results of Kacem et al.

5.1 Kacem, Hammadi, and Borne (2002) Instances

These Kacem instances that we have evaluated here are 4x5, 8x8, 10x10, and 15x10. Kacem defines the 4x5 instance as small-scale and flexible - all machines can process every operation. The 8x8 instance is partially flexible since some machines cannot perform certain operations. Kacem regards this as a very difficult instance to compute. Large-scale flexible 10x10 and 15x10 instances are available.

All Kacem instances are flexible except for 8x8, where processing times are fixed per machine-operation pair. The 8x8 example has partial flexibility due to machine eligibility constraints: a subset of operations prohibits using one or two machines. Despite these constraints, the instance's structural similarity to other Kacem benchmarks permits results comparable to those reported in the literature.

For the 4x5 instance (comprising 12 operations in total), the algorithm was configured with fewer iterations and a reduced population of bees due to its lower complexity. Across all executions, the algorithm consistently identified solutions meeting the target objectives. Detailed parameter settings for this instance are provided in Table 5-1.

Table 5-1 Presentation of configuration values for Kacem instances

Instance	Total Iterations	N-Bees	Limit	Initial Employed Bee Exploration	Pareto Archive	M1	M2	M3	WCm	WWt	WWm	Average Time (s)
4 x 5	400	200	10	8	40	10	50	40	2.2	1.7	1.5	1548
8 x 8	600	200	15	20	120	20	70	10	1.8	1.5	1.2	1987.2
10x 10	600	200	15	20	120	50	40	10	5	2	1	1925.7
15 x 10	1000	300	20	25	60	10	50	40	8	1	1	5999.4

The values for iterations and the number of bee variables were intentionally set much higher than the target results required. This design choice gives the algorithm enough time and computational resources to explore different solutions even if the desired outcomes are usually known in the first iterations. The algorithm typically finds state-of-the-art or novel solutions in the first 10 iterations, and subsequent iterations are devoted to increasing solution variety, as shown in Table 5-1.

Due to its inherent complexity, the partially flexible 8x8 instance (comprising 27

n x m	4 x 4				8 x 8				10 x 10				15 x 10			
	Cm	Wt	Wm	Sol	Cm	Wt	Wm	Sol	Cm	Wt	Wm	Sol	Cm	Wt	Wm	Sol

operations, 8 jobs, and 8 machines) required higher iterations and bee populations. Despite machine eligibility constraints (where one or two machines are excluded per operation), the instance retains structural similarities to Kacem's fully flexible cases, such as variable processing times across machines. Multiple configurations were tested to optimize efficiency, with Table 5-1 highlighting the setup that yielded the most incredible solution diversity.

The weight parameters wcm, wwt, and wwm were calibrated iteratively. Prioritizing WCm frequently led to concurrent reductions in WWt while overemphasizing WWM degraded performance due to excessive focus on minimizing the most loaded machine.

The 10x10 and 15x10 instances, classified as large-scale, contain 30 and 56 operations, respectively. Both feature machines with highly variable processing times (ranging from 1–23 ut for 10x10 and 1–85 ut for 15x10), simplifying machine selection by naturally excluding slower machines from optimal schedules. Table 5-1 provides configuration details and average execution times for these instances.

The problem is multi-objective so multiple interpretations of the best solutions are possible. In response to this, the results are presented first by the most minor to the most significant makespan and second by the most minor workload total. The maximum workload machine is the last sorting criterion.

In Table 5-2, the Sol column indicates the number of distinct solutions for each combination of objective values. Literature studies with only one solution per objective are marked with Sol = 1. This comparison highlights the proposed algorithm's ability to match or exceed the best-known results while generating a broader range of valid solutions.

For the 4x5 instance, the algorithm successfully identified all known best results (see the last rows of Table 5-2). The highest diversity was achieved for solutions 11, 32, and 10, with six distinct schedules generated. These schedules are visualized in the Gantt charts.

Table 5-2 Comparison of Results from Other Authors with the Proposed Algorithm.

DABCA ^[15]	11	32	10	1	14	77	12	1	7	42	6	1	11	91	11	1
	11	34	9	1	15	75	12	1	7	42	5	1	11	93	10	1
	12	32	8	1	16	73	13	1	8	41	7	1	-	-	-	-
	13	33	7	1	16	77	11	1	8	42	5	1	-	-	-	-
NSGAI-BEG ^[16]	11	32	10	1	14	77	12	1	7	42	6	1	11	91	11	1
	11	34	8	1	15	75	12	1	7	43	5	1	11	95	10	1
	12	32	8	1	16	73	13	1	8	41	7	1	12	93	10	1
	13	33	7	1	17	77	11	1	8	42	5	1	-	-	-	-
SPO ^[17]	-	-	-	-	-	-	-	-	7	43	5	25	-	-	-	-
	-	-	-	-	-	-	-	-	8	41	7	35	-	-	-	-
	-	-	-	-	-	-	-	-	8	42	5	16	-	-	-	-
ABC-DIC ^[18]	-	-	-	-	-	-	-	-	7	42	6	34	-	-	-	-
	-	-	-	-	-	-	-	-	7	43	5	58	-	-	-	-
	-	-	-	-	-	-	-	-	8	41	7	66	-	-	-	-
	-	-	-	-	-	-	-	-	8	42	5	42	-	-	-	-
EPABC ^[19]	11	32	10	1	14	77	12	1	7	42	6	1	11	91	11	1
	11	32	8	1	15	75	12	1	7	43	5	1	11	93	10	1
	13	33	7	1	16	73	13	1	8	41	7	1	-	-	-	-
	11	34	9	1	16	77	11	1	8	42	5	1	-	-	-	-
P-DABC ^[20]	11	32	10	1	14	77	12	1	7	43	5	1	11	93	11	1
	12	32	8	1	15	75	12	1	8	41	7	1	12	91	10	1
	13	33	7	1	16	73	13	1	8	42	5	1	-	-	-	-
SPO+RRHC ^[55]	-	-	-	-	12	73	13	1	7	43	5	1	11	91	11	1
	-	-	-	-	14	77	12	1	7	42	6	1	14	93	10	1
	-	-	-	-	15	75	12	1	8	42	5	1	-	-	-	-
	-	-	-	-	16	77	11	1	8	41	7	1	-	-	-	-
PABC	11	32	10	6	14	77	12	4	7	42	6	51	12	91	12	1
	11	34	9	1	15	75	12	24	8	41	7	3360	12	93	11	28
	12	32	8	4	16	73	13	53	8	42	5	144	13	92	11	140
	13	33	7	3	16	77	11	1	-	-	-	-	-	-	-	-

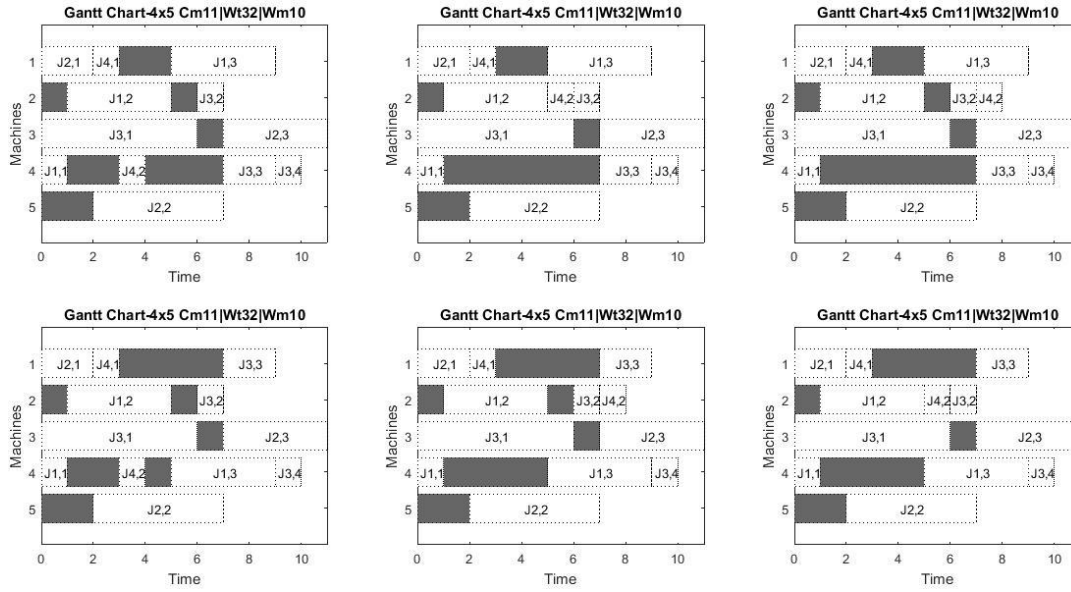


Figure 5-1 All Results for the Objective Set 11, 32, 10 in the 4x5 Instance.

Figure 5-1 illustrates that the Gantt charts differ in structure but return the same values for the three objectives: Cm, Wt, and Wm. This illustrates how the algorithm may generate various scheduling configurations while satisfying the same optimization targets.

Similarly, Figure 5-2 shows four Gantt charts for the solution set 12, 32, and 8. Different charts have identical objective values despite differences in job sequencing and machine assignments, demonstrating that the algorithm can produce different yet equally optimal solutions.

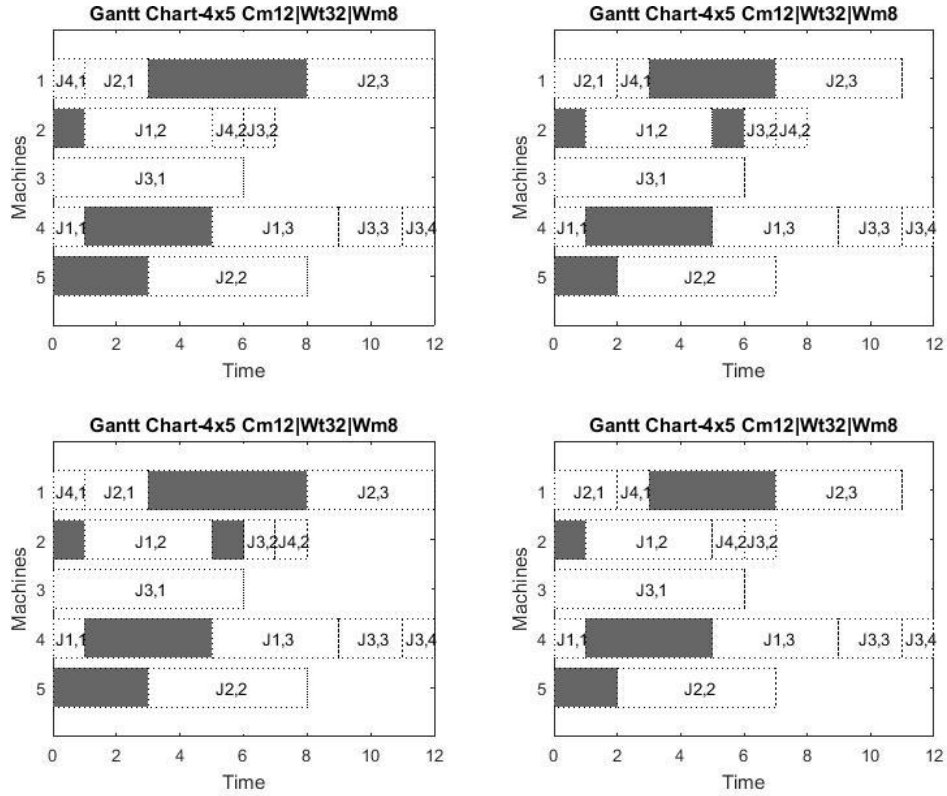


Figure 5-2 All Results for the Objective Set 12, 32, 8 in the 4x5 Instance

Figure 5-3 presents the distinct Gantt charts generated for results 13,33,7, which feature the lowest Wm among known best results. The objective values are identical; however, the generated schedules exhibit structural differences in their Gantt charts.

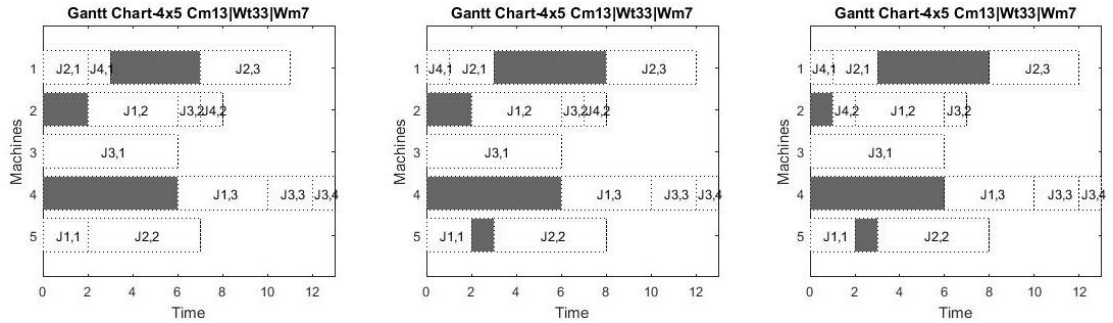


Figure 5-3 All Results for the Objective Set 13, 33, 7 in the 4x5 Instance.

Finally, the instance 11,34,9, which yielded only one solution, is illustrated in Figure 5-4.

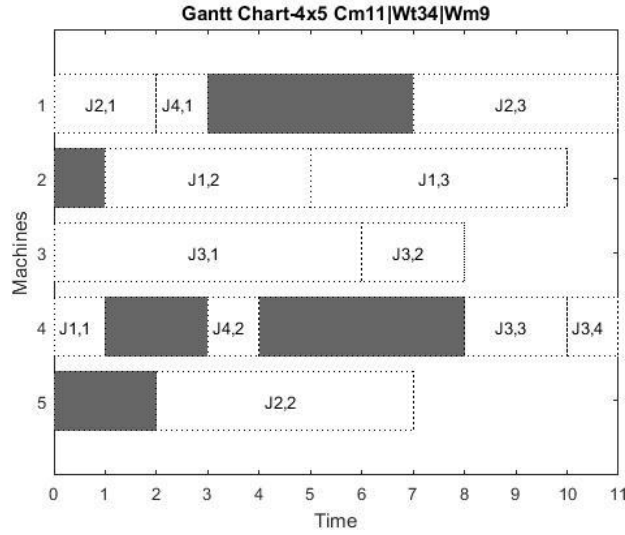


Figure 5-4 Results for the Objective Set 11, 34, 9 in the 4x5 Instance

For the 4x5 instance, 14 different results were obtained for all solutions identified for the four best-known objective combinations from the literature.

Figure 5-5 compares the 4x5 instance Pareto frontier with respect to the most familiar benchmarks from the literature. Green markers are the best-known results from the literature, and red are the non-dominated solutions identified by the given algorithm. The three plots in Figure 5-5 are labeled with the respective objective pairs compared.

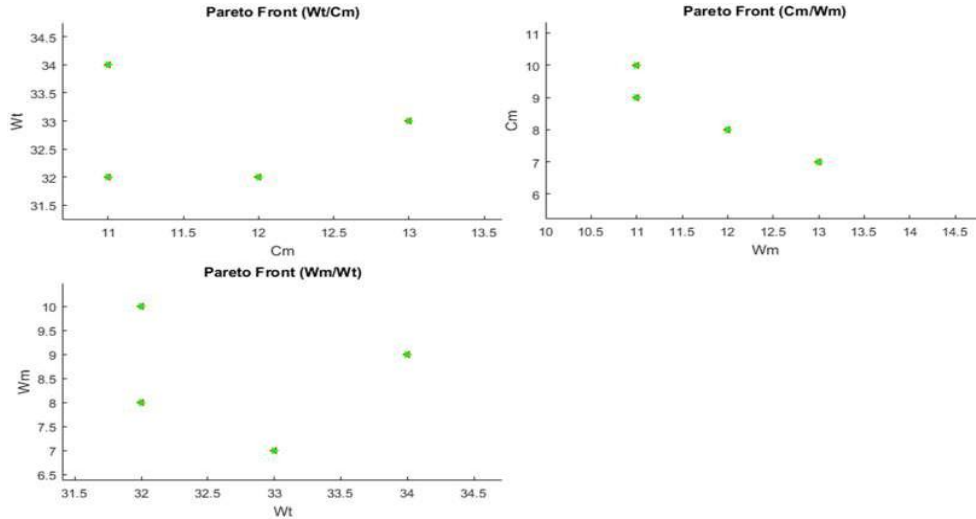


Figure 5-5 Pareto Frontier for the 4x5 Instance

The green and red points overlap, indicating that the best-known values were identified. For the 8x8 instance, the result with the highest diversity was 16,73,13, which

has the lowest Wt known for the values 16,13 (Cm and Wm, respectively), with 53 distinct solutions. The result with the lowest diversity was 16,77,11, with the lowest Wm known for 16,77 yielding only 1 solution.

Figure 5-6 displays the number of distinct solutions for the 8x8 instance. The x-axis represents the multi-objective results, and the y-axis shows the number of unique solutions.

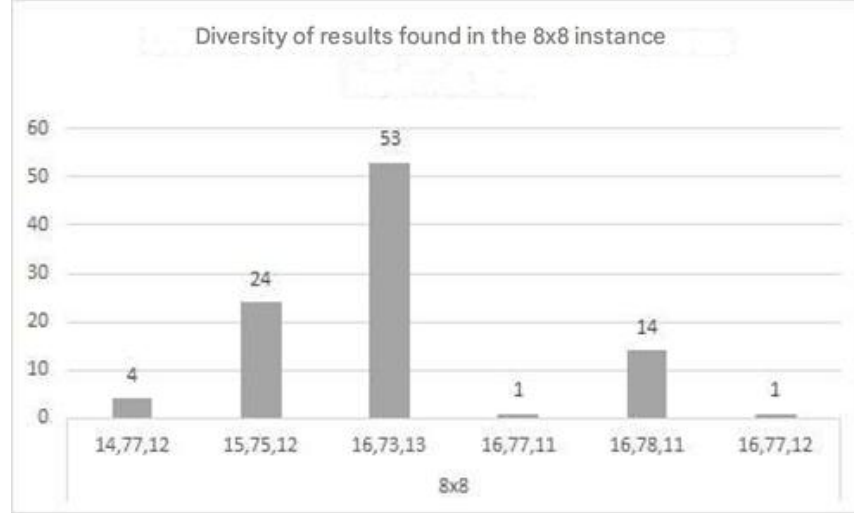


Figure 5-6 Number of Solutions Identified per Result.

Given this instance's high diversity of solutions, it is impractical to present all corresponding graphs. Therefore, the complete diversity will not be displayed.

Figure 5-7 illustrates 2 of the 24 solutions found for the result 15,75,12.

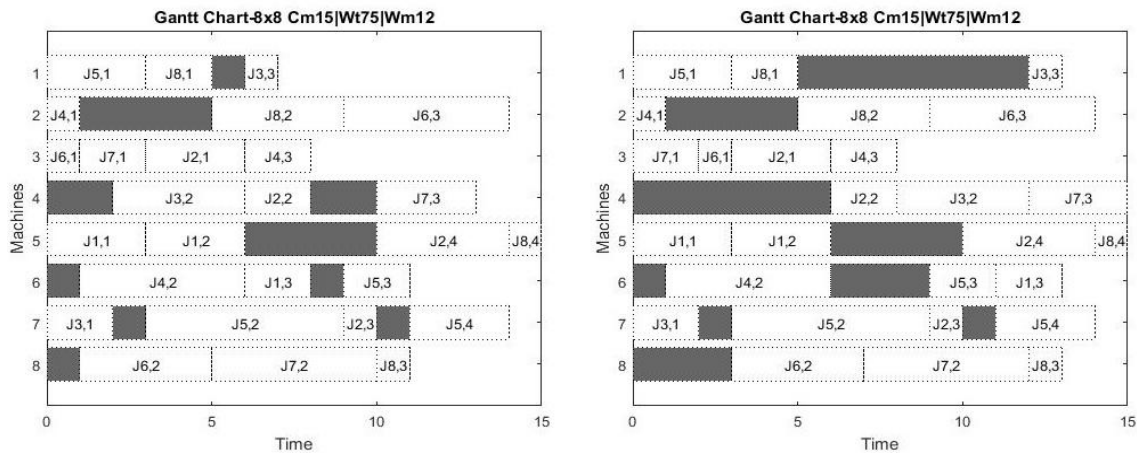


Figure 5-7 Two Solutions Identified for the 8x8 Instance.

The Gantt charts in Figure 5-7 present two distinct schedules for the results 15,75,12

(Cm, Wt, Wm). While both achieve identical objective values, their structural differences highlight variations in scheduling strategies:

Left Chart: Features a schedule with lower idle time and only one machine completing its tasks at the maximum makespan

Exemplary Chart: Shows machines 4 and 5 completing operations at the maximum makespan

These differences illustrate that no schedule is inherently superior; their suitability depends on operational priorities.

Two of the 53 solutions for result 16 are shown in Figure 5-8. Structured differences among these charts emphasize how well the algorithm can produce various schedules with equivalent objective values.

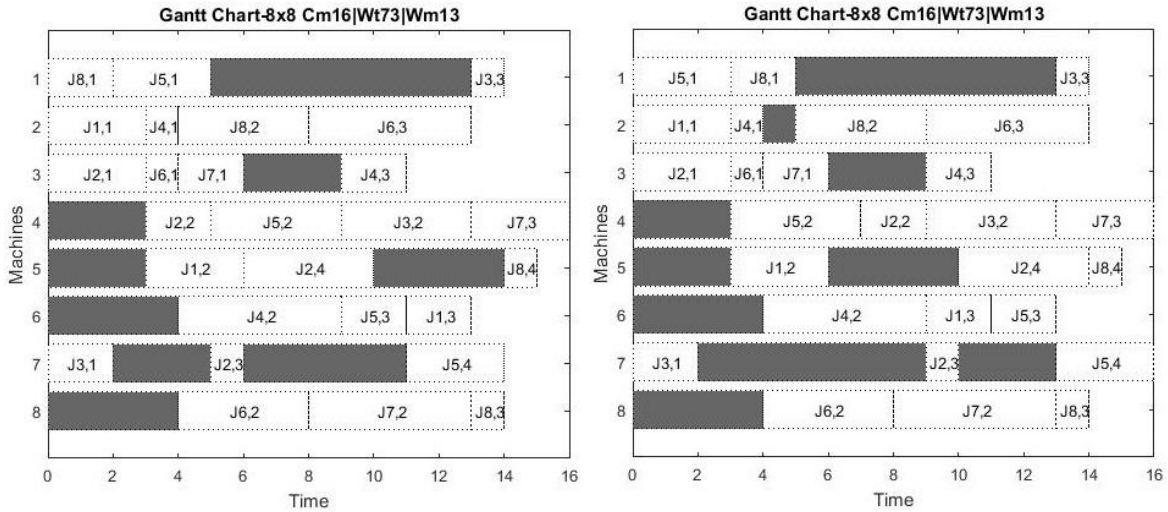


Figure 5-8 Solutions Identified for the 8x8 Instance

The structural differences between the Gantt charts in Figure 5-8 are due to different objectives being set for each task. With 53 solutions, there are few discrepancies in key metrics:

Idle Time Variation: Very small differences in machine waiting periods between operations.

Only one or two machines consistently complete operations within the maximum makepan.

This uniformity hints that the algorithm finds itself converging to efficient scheduling

patterns even when producing different solutions.

Figure 5-9 shows three solutions found for result 14,77,12. Such charts further demonstrate that the algorithm can handle structural diversity while respecting objective constraints.

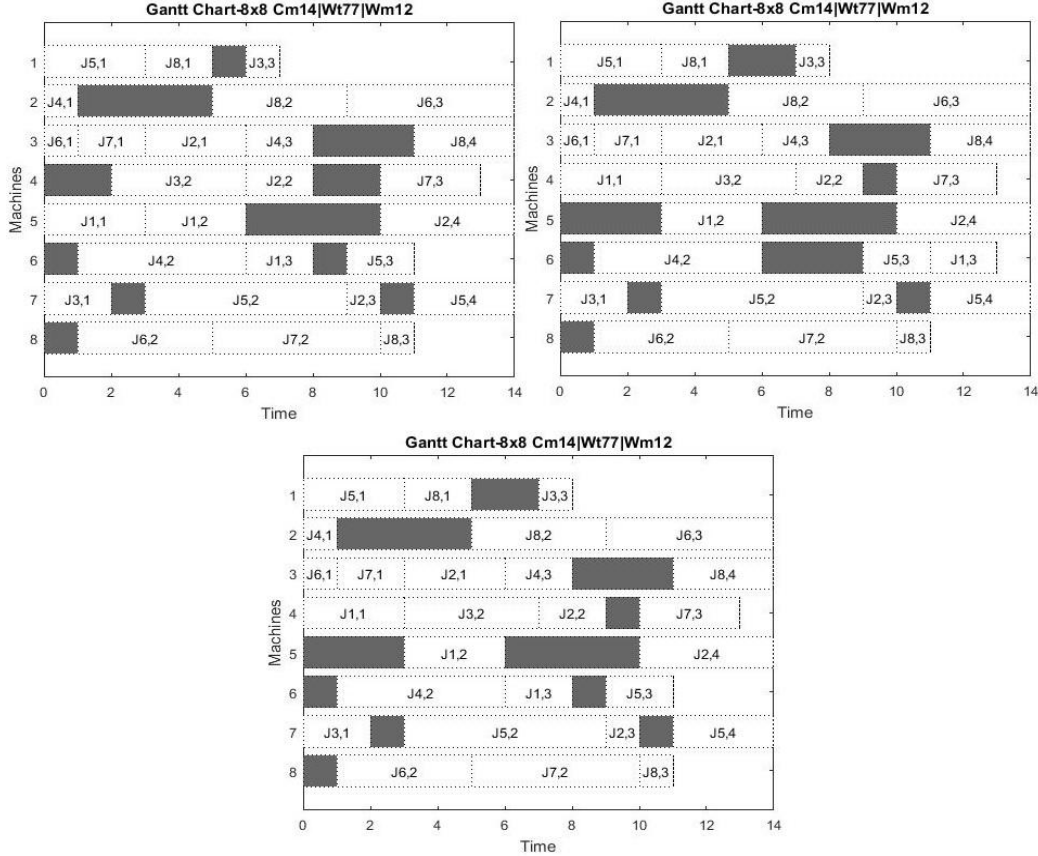


Figure 5-9 Solutions Identified for the 8x8 Instance

The three Gantt charts in Figure 5-9 exhibit minimal structural variations. Charts 2 and 3 differ only in the scheduling of machine 6, while Charts 1 and 3 vary in the assignments for machines 1 and 4. Such minimal diversity constrains the exploration of scheduling alternatives for the objective of set 14,77,12.

In Figure 5-10, we see the solution to the equation 16,77,11 only being identified once across executions. This resulted in the hardest outcome. Another similar schedule (19,77,11) was found repeatedly. Only the assignment of operation J6,3 (executed by machine 2 in the left chart) differs between the two schedules below. The algorithm's inability to generate diverse schedules for 16,77,11 despite minor operational variations highlights a limitation

in its capacity to explore alternative configurations under stringent objective constraints.

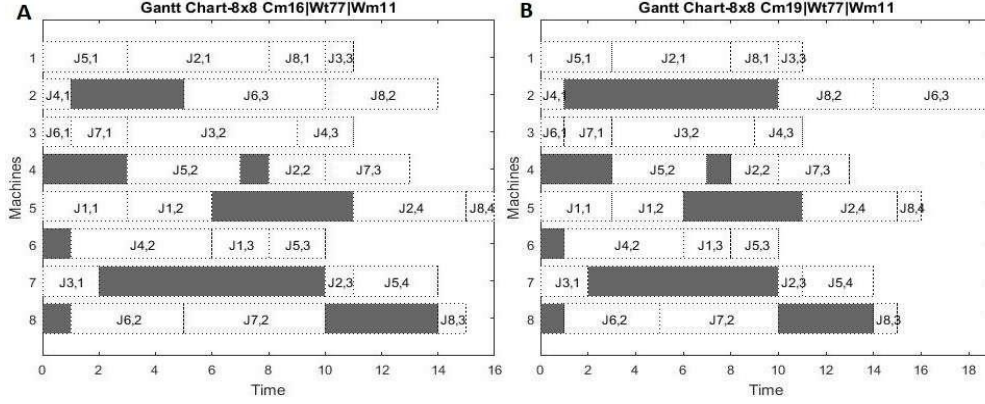


Figure 5-10 Results similar to those found for the 8x8 instance

Considering only the best-known results from the literature, this instance's total number of distinct schedules amounts to 81 unique Gantt charts.

Figure 5-11 presents the Pareto frontier for the 8x8 instance across all objectives for improved visualization of the results compared to the best-known literature benchmarks. Green markers represent the most notable results from the literature, while red markers denote the non-dominated solutions identified via the algorithm. Every single one of the three plots in Figure 5-11 is labeled at the top, and the respective objective pairs are compared.

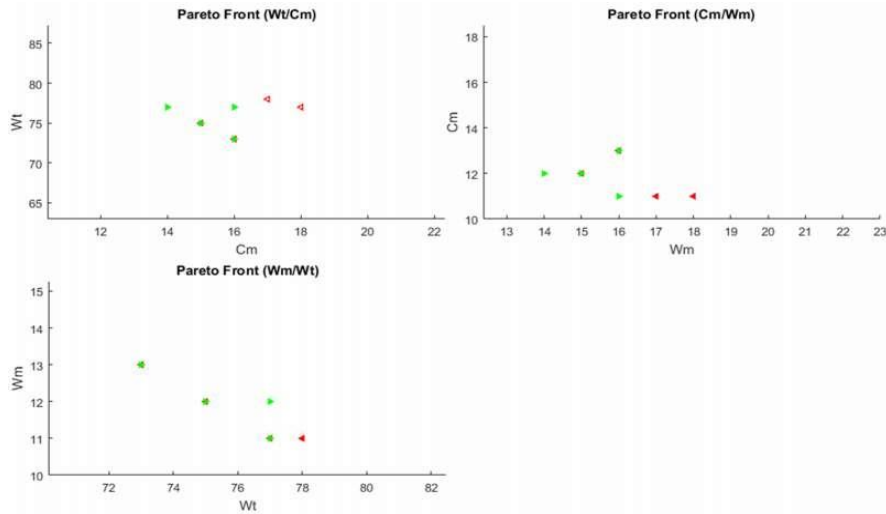


Figure 5-11 Pareto frontier for the 8x8 instance

For the 10x10 instance, the result with the highest diversity was 8,41,7, which has the

lowest Wt known for the values 8,7, with 3,360 distinct solutions. The result with the lowest diversity was 8,43,5, with the lowest Wm known for the values 8,43, totaling 7 distinct solutions.

Among the best-known results, the only solution the algorithm did not identify was 7,43,5 (highlighted in bold in Table 5-2). However, it closely approached this result with 7,43,6, generating 431 distinct solutions. The 10x10 instance demands high algorithmic complexity to achieve 7,43,5, as this combination includes the two lowest individual values, for instance, $C_m = 7$ and $W_m = 5$.

For this instance, only two Gantt charts will be presented per best-known result (where notable differences exist); otherwise, a single chart is provided. This simplification is necessary due to the large number of operations and solutions. Figure 5-12 illustrates two of the 70 solutions identified for the results 7,42,6.

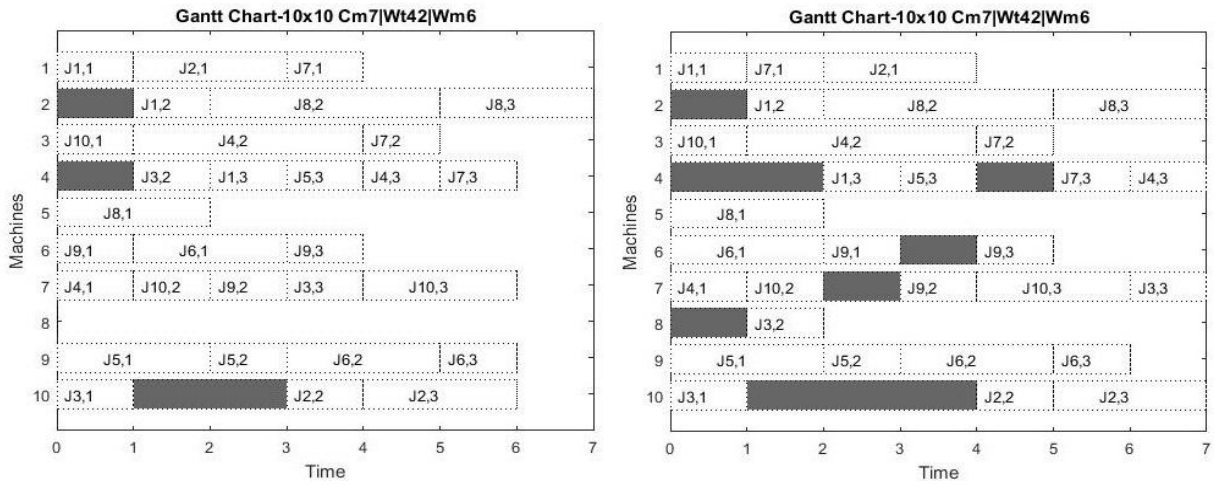


Figure 5-12 Results found for the 10x10 instance

As shown in Figure 5-12, one of the schedules does not utilize machine 8 for any operations, and machine 5 remains idle most of the time. A positive feature of this configuration is the possibility of performing maintenance on machines 5 and 8 while the process runs without compromising any of the evaluated objectives. Another notable characteristic is the idle time difference between the two schedules: the first has only 4 ut of idle time, while the second has 10. A final observation is that in the first Gantt chart, only one machine completes operations at the makespan ($C_m = 7$). In contrast, four of the ten

machines finish simultaneously at $C_m = 7$ in the second.

Figure 5-13 illustrates one of the 3,360 solutions identified for the result 8,41,7.

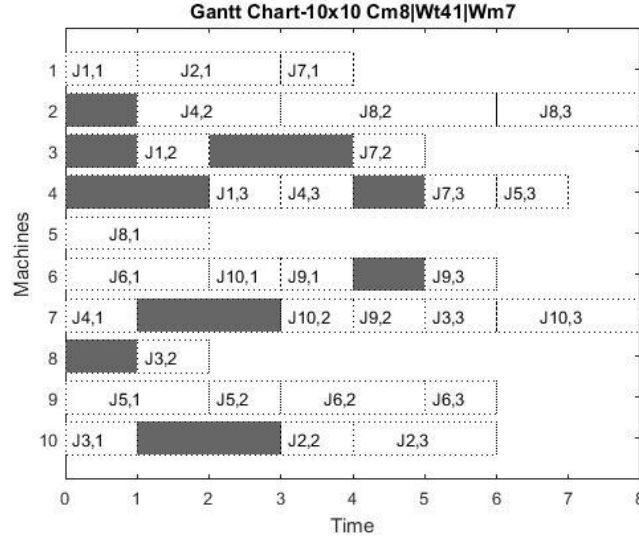


Figure 5-13 Result found for the 10x10 instance

In the configuration shown in Figure 5-13, machine 8 is utilized but becomes available from time unit 2 onward, as does machine 5. Due to the large number of schedules, analyzing and comparing them was not considered feasible. However, it was observed that idle times across schedules are not significantly divergent. No schedules were found that entirely omit the use of a machine, and the number of machines completing operations at the maximum time (C_m) ranges from 1 to 4.

In Figure 5-14, a diagram of the diversities of solutions found for each objective value in the 10x10 instance is shown.

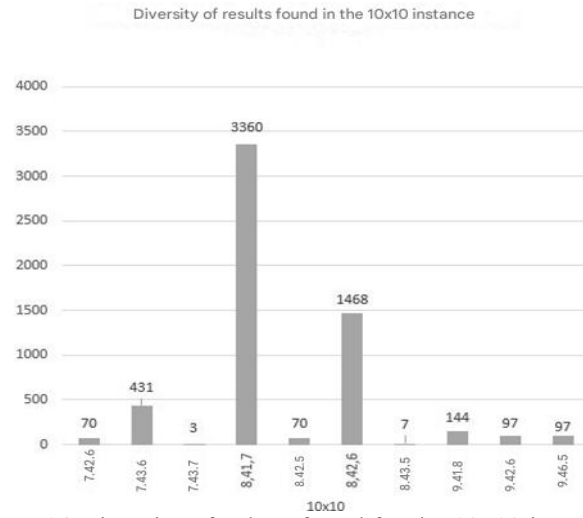


Figure 5-14 Diversity of values found for the 10x10 instance

Figure 5-14 shows that the highest scheduling variation was 8,41,7, and the lowest was 7,43,7.

Figure 5-15 presents the Pareto frontier for the 10x10 instance across all objectives to improve visualization of the results matched to the well-known benchmarks from the literature. Green markers represent the most notable results from the literature, and red markers denote the non-dominated solutions identified by the presented algorithm.

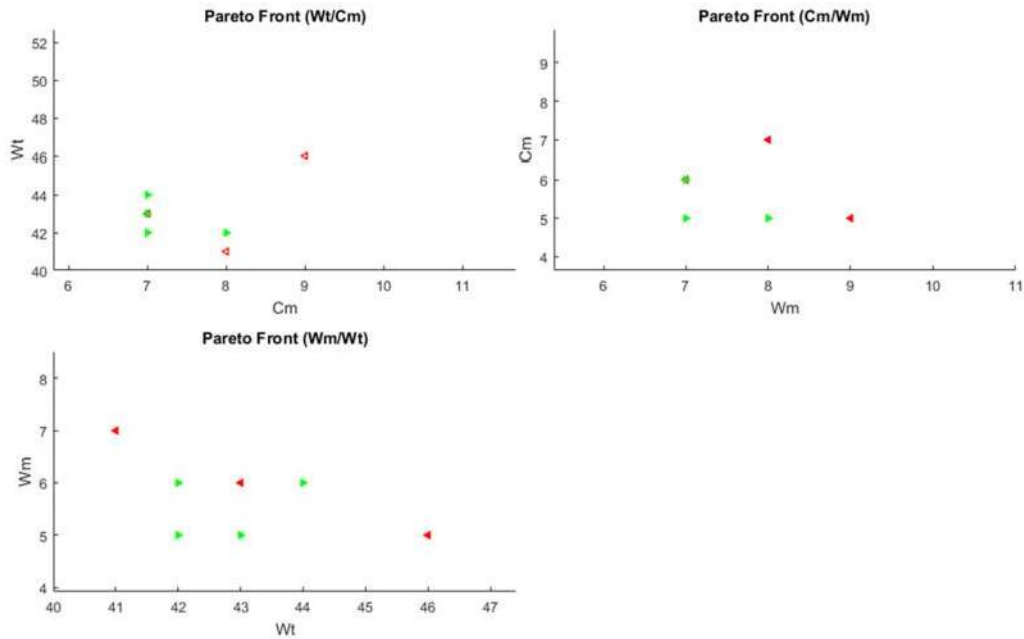


Figure 5-15 Pareto frontier for the 10x10 instance

As shown in Figure 5-15, the Pareto frontier plots indicate that only one of the well-

known results was not identified.

For the 15x10 instance (the most significant Kacem benchmark), the result with the highest diversity was 13,91,12, which has the lowest Wt known for the values 11,11(Cm, Wm). While the algorithm did not identify the optimal multi-objective value for this instance (11,91,11), it discovered several high-diversity solutions, as illustrated in Figure 5-16.

The result with the lowest diversity for this instance was 12,91,12, yielding only one schedule. Although not among the best-known results, this solution remains viable, albeit lacking diversity.

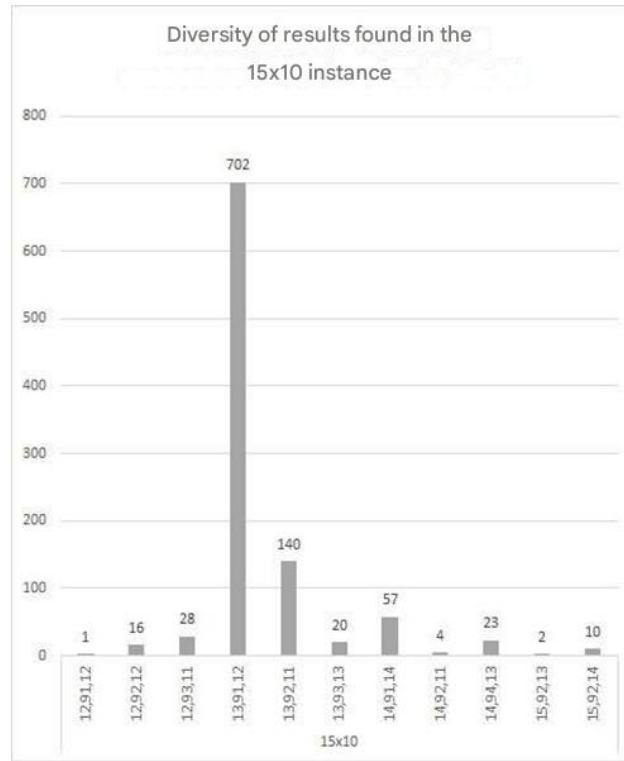


Figure 5-16 Diversity found for the 15x10 instance

The highest diversity was 702 solutions for result 13,91,12 for the 15x10 instance, and the lowest was 12,91,12 with only 1 solution.

It is a big undertaking with many operations and is therefore unlikely to yield the best results. Across all executions, the algorithm failed to attain the desired multi-objective values. However, it achieved the minimum Wt (91) when considered individually. The known minima for Cm (11) and Wm (10) were not identified.

Because no best-known multi-objective results were identified by the algorithm, only the best-found solutions are presented via Gantt charts. Figure 5-17 illustrates one of these solutions: machines M2 and M10 are kept occupied during the schedule. An alternative strategy to get the optimal result 11,91,11 could be to optimize the critical path, but this principle was not explicitly incorporated in the algorithm design.

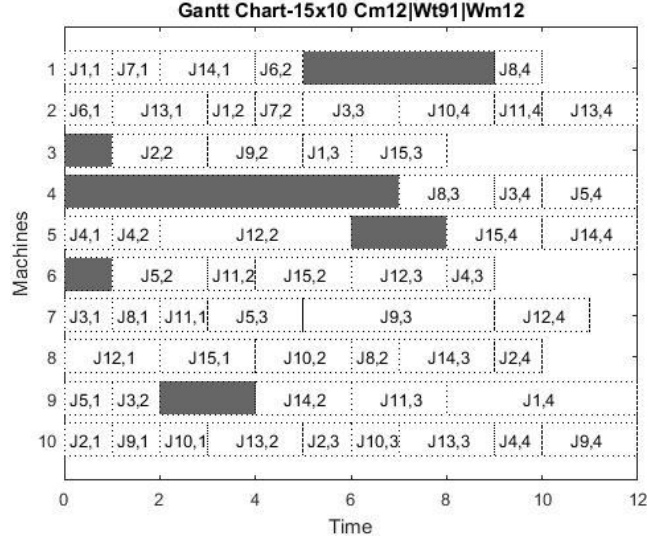


Figure 5-17 Result found for the 15x10 instance

The presented results demonstrate that the algorithm successfully identified all well-known solutions for the first three instances, except one consequence in the 10x10 instance. For the 15x10 instance, while the algorithm did not achieve the best-known benchmarks, it generated a high diversity of solutions. To attain optimal results for this more prominent instance, modifications to the search mechanisms or parameter tuning aligned with its specific characteristics are likely required.

For improved visualization of the results in contrast with the well-known benchmarks from the literature, Figure 5-18 presents the Pareto frontier for the 15x10 instance across all objectives. Green markers represent the most notable results from the literature, while red markers denote the non-dominated solutions identified by the suggested algorithm. The three graphs presented in Figure 5-18 are identified with the name at the top.

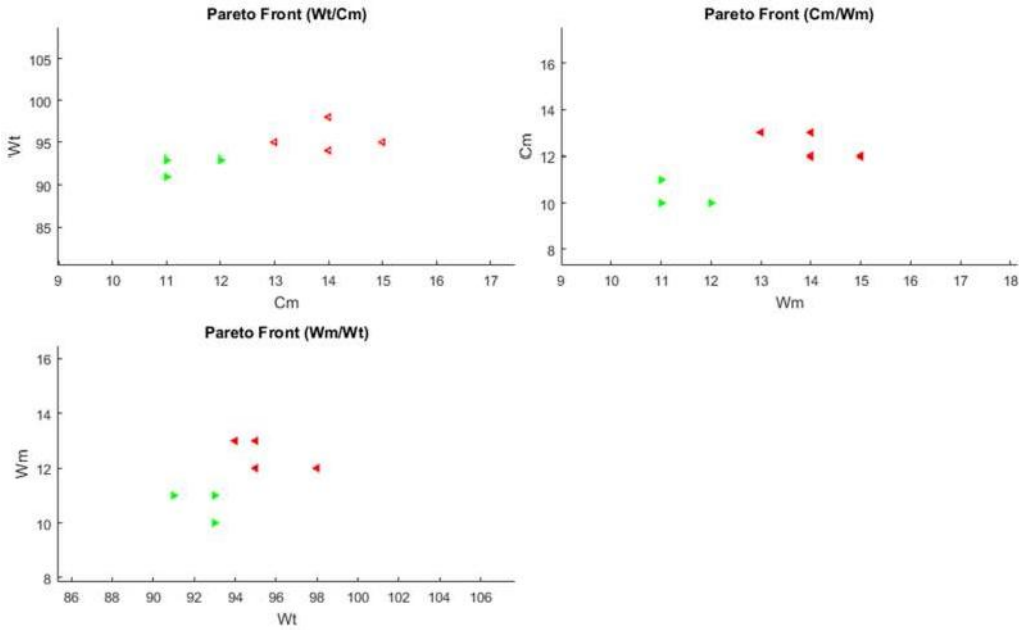


Figure 5-18 Pareto frontier for the 15x10 instance

As observed in Figure 5-18, the identified results approach but do not fully reach the best-known benchmarks from the literature.

This part presents the experimental results obtained in our study with the Brandimarte cases.

5.2 Instances of Brandimarte (1993)

The Brandimarte (1993) instances used in this work were mk01, mk02, mk03, mk04, mk05, mk07, mk08, and mk09. These instances differ significantly from Kacem's, as they are all partially flexible, and in general, machines execute operations with identical processing times for the same operation. When analyzing the results for instances with limited machine availability per operation, a lower diversity of solutions was observed. This conclusion was drawn due to the reduced number of possible machine-operation permutations the algorithm could explore. Since the algorithm was developed based on instances where most or all machines are available for each operation (with widely varying processing times), the opposing characteristics of Brandimarte's instances partially suppress the algorithm's efficiency.

The developed search mechanisms are oriented towards machine selection rather than operation scheduling. The ordering of operations in the scheduling phase was the

bottleneck of the algorithm for Brandimarte's instances.

These are large instances - from 10x6 to 20x10, and operations from 240 to 55. Compared to Kacem's instances, which range between 56 to 12 operations, even the most minor Brandimarte instance (mk01: 10x6, 55 operations) is nearly the size of Kacem's most significant (15x10, 56 operations).

Although the algorithm was not fully optimized for Brandimarte's benchmark, the results were satisfactory in some instances.

Tables and figures with comparisons will be presented to facilitate visualization of the results. Table 5-3 shows the configurations used to achieve the results discussed below.

Table 5-3 Configurations of the Brandimarte instances.

Instance	Total Iterations	Number of Bees	Limit	Initial Exploration of Employee	Pareto File	M1	M2	M3	WCm	WTw	WMw	Average Time (s)
MK01	600	400	15	15	80	7	80	13	1.8	1	1.2	1857.6
MK02	500	600	15	15	120	7	80	13	1.8	1	1.2	4265.7
MK03	900	400	15	15	80	7	80	13	1.8	1	1.2	5841
MK04	700	400	15	15	80	7	80	13	2	2.5	1	5099.4
MK05	700	400	15	15	80	7	80	13	1.8	1	1.2	5274
MK06	1000	200	15	15	80	2	30	68	3	1	1	7751.7
MK07	700	400	15	15	80	7	80	13	1.8	1	1.2	4380.6
MK08	1000	200	15	15	60	10	20	70	3	1	1	7669.2
MK09	900	200	15	15	60	25	60	15	15	3	1	8204.1

As mentioned, the algorithm was developed generically to find solutions across any instance. However, although it functions as intended, it is not as efficient for Brandimarte's instances as it is for Kacem's. Given this reduced efficiency for such instances, Table 5-3 shows that the number of executions and the population of bees were significantly increased to enable the algorithm to identify high-quality solutions.

The following sections present the best results obtained and compare them against the best-known benchmarks from the literature.

Considering all cases, the best outcomes were achieved for the mk02 and mk05 instances. The mk05 instance exhibited the highest solution diversity; however, the best-known values were not attained for any instances. Pareto frontier plots will be presented for each instance to demonstrate the quality of the best solutions identified visually.

Table 5-4 Comparison of the results obtained from the literature for the mk01, mk02, mk03, and mk04 instances of Brandimarte.

n x m	MK01(10 x 6)				MK02(10 x 6)				MK03(15 x 8)				MK04(15 x 8)			
	Cm	Wt	Wm	sol	Cm	Wt	Wm	sol	Cm	Wt	Wm	sol	Cm	Wt	Wm	sol
DABCA	40	167	36	1	28	155	28	1	204	850	204	1	60	374	60	1
	40	162	38	1	26	151	26	1	210	848	210	1	61	368	60	1
	40	164	37	1	-	-	-	-	813	844	213	1	61	365	61	1
NSGAIL-BEG	40	169	36	1	26	151	26	1	204	855	199	1	60	390	59	1
	42	158	39	1	27	146	27	1	204	871	144	1	63	371	61	1
	43	155	40	1	29	145	27	1	204	882	135	1	65	362	63	1
HDSPO	40	167	36	1	26	150	26	1	204	850	204	1	60	372	60	1
	41	163	37	1	67	145	27	1	210	848	210	1	61	366	60	1
	42	165	36	1	28	144	28	1	213	844	213	1	62	363	60	1
PABC	42	167	42	255	27	151	27	35	204	865	204	1	75	356	75	1
	43	163	43	1	28	146	27	1	204	874	204	20	75	357	75	6
	43	165	42	1	28	146	28	4	204	882	204	21	76	349	75	1
	43	166	42	98	28	147	28	160	209	859	204	1	76	350	75	6
	44	162	43	6	28	148	27	71	213	897	213	18	76	353	75	26
	44	164	42	5	29	144	29	3	231	858	213	1	78	349	75	13
	45	159	44	4	29	146	29	23	236	891	204	54	79	346	78	8
	45	160	43	23	29	146	29	14	-	-	-	-	81	345	73	2
	45	160	43	16	29	147	29	75	-	-	-	-	81	347	67	4
	45	162	42	8	29	148	29	221	-	-	-	-	89	348	67	10
	45	164	43	34	29	150	28	456	-	-	-	-	90	353	62	26
	46	158	44	12	29	152	26	92	-	-	-	-	96	353	63	11
	46	160	43	3	30	143	31	1	-	-	-	-	130	326	130	1
	46	168	36	20	30	146	27	742	-	-	-	-	137	326	137	3
	47	156	45	4	30	147	30	23	-	-	-	-	145	326	145	6
	47	159	43	4	30	148	27	70	-	-	-	-	150	324	146	1
	47	167	36	3649	30	152	27	2	-	-	-	-	-	-	-	-
	49	164	37	28	31	145	27	54	-	-	-	-	-	-	-	-
	50	153	46	24	31	147	27	11	-	-	-	-	-	-	-	-
	50	165	36	1	31	147	27	124	-	-	-	-	-	-	-	-
	53	167	36	1	31	151	27	447	-	-	-	-	-	-	-	-
	54	162	39	8	32	143	30	1	-	-	-	-	-	-	-	-
	56	153	42	1	33	142	33	25	-	-	-	-	-	-	-	-
	57	153	42	9	33	145	27	1	-	-	-	-	-	-	-	-
	57	159	39	1	34	142	31	2	-	-	-	-	-	-	-	-
	57	161	38	1	35	141	33	1	-	-	-	-	-	-	-	-
	57	166	36	1	36	143	29	2	-	-	-	-	-	-	-	-
	58	153	42	6	54	142	25	1	-	-	-	-	-	-	-	-
	59	153	42	39	55	142	26	10	-	-	-	-	-	-	-	-

Table 5-5 Comparison of the results obtained with those from the literature for the mk05, mk07, mk08, and mk09 instances of Brandimarte

n x m	MK05 (15x4)				MK07 (20x5)				MK08 (20x10)				MK09 (20x10)			
	C _m	Wt	Wm	Sol	C _m	Wt	Wm	Sol	C _m	Wt	Wm	Sol	C _m	Wt	Wm	Sol
DABCA ^[18]	172	687	172	1	139	693	139	1	523	2524	523	1	-	-	-	-
	173	683	173	1	140	685	140	1	524	2519	524	1	-	-	-	-
	175	682	175	1	143	683	143	1	533	2514	533	1	-	-	-	-
NSGAIL-BEG ^[16]	173	683	173	1	139	693	139	1	523	2524	515	1	-	-	-	-
	175	682	175	1	140	686	138	1	523	2534	497	1	-	-	-	-
	183	677	183	1	144	673	144	1	524	2519	524	1	-	-	-	-
HDSPO	172	687	172	1	139	693	139	1	523	2524	523	1	307	2281	299	1
	173	683	173	1	141	692	141	1	524	2519	524	1	309	2273	299	1
	175	682	175	1	144	673	144	1	533	2514	533	1	311	2260	310	1
PABC	173	685	173	5334	143	695	143	3	562	2514	551	1	357	2365	339	1
	173	686	173	7168	144	676	144	5	563	2503	560	1	358	2347	323	1
	174	687	174	2531	144	677	144	12	567	2522	533	1	359	2340	335	1
	175	683	175	465	144	692	144	170	568	2512	551	1	361	2328	388	1
	175	684	175	8855	145	696	141	1	568	2524	533	1	363	2321	347	1
	175	686	175	840	146	688	146	16	570	2522	533	1	369	2308	349	1
	175	687	175	12854	148	693	148	86	570	2534	524	1	369	2313	341	2
	176	682	176	652	156	692	141	1	571	2519	524	1	416	2306	373	1
	176	683	176	18	161	671	161	1	572	2522	533	1	427	2297	410	1
	176	685	173	1	165	669	156	1	572	2527	524	1	435	2292	405	1
	177	686	173	3958	165	690	141	1	573	2498	569	1	440	2288	388	1
	178	685	173	3159	170	665	167	1	573	2517	542	1	-	-	-	-
	209	672	209	325	175	664	168	1	573	2522	533	4	-	-	-	-
	-	-	-	-	178	662	166	1	591	2484	587	1	-	-	-	-
	-	-	-	-	184	660	184	1	591	2532	523	1	-	-	-	-
	-	-	-	-	195	667	163	1	592	2538	523	1	-	-	-	-
	-	-	-	-	196	661	181	1	593	2528	523	1	-	-	-	-
	-	-	-	-	211	658	211	1	595	2484	587	1	-	-	-	-
	-	-	-	-	-	-	-	-	595	2532	523	2	-	-	-	-

As the problems are enormous, only Pareto frontier plots will be shown for each instance in this case.

Figure 5-19 shows the most familiar results from literature for the MK01 instance (green markers) together with the best solutions identified by the proposed algorithm (red markers).

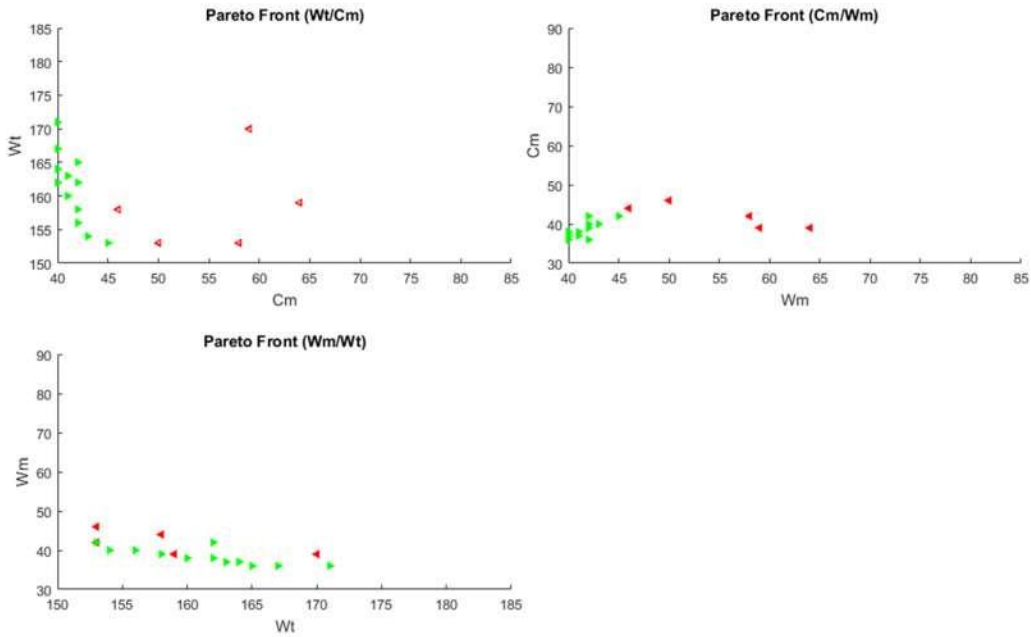


Figure 5-19 Pareto Frontier for Brandimarte mk01 Instance

As seen, the best-known results cluster together to form a barrier, and the identified solutions cross the barrier.

Figure 5-20 illustrates the Pareto frontier for the mk02 instance, where the proposed algorithm's solutions intermingle with the best-known benchmarks without distinct separation.

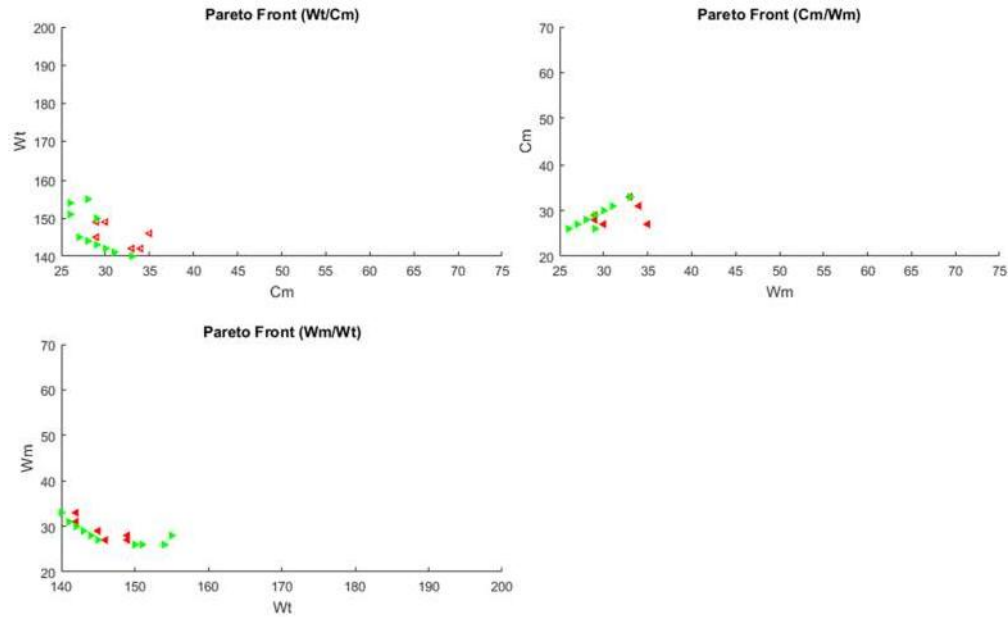


Figure 5-20 Pareto Frontier for Brandimarte mk02 Instance

Figure 5-20 shows the Pareto frontier for the variable k . The results show that uncovered solutions stand apart from the frontier drawn by the best-known benchmarks in the Wt/Cm plot. Yet, when analyzed individually, the values of the identified solutions are close to the best-known results. Conversely, the Cm/Wm plot demonstrates that the algorithm found the lowest Wm values if the objectives are treated independently.

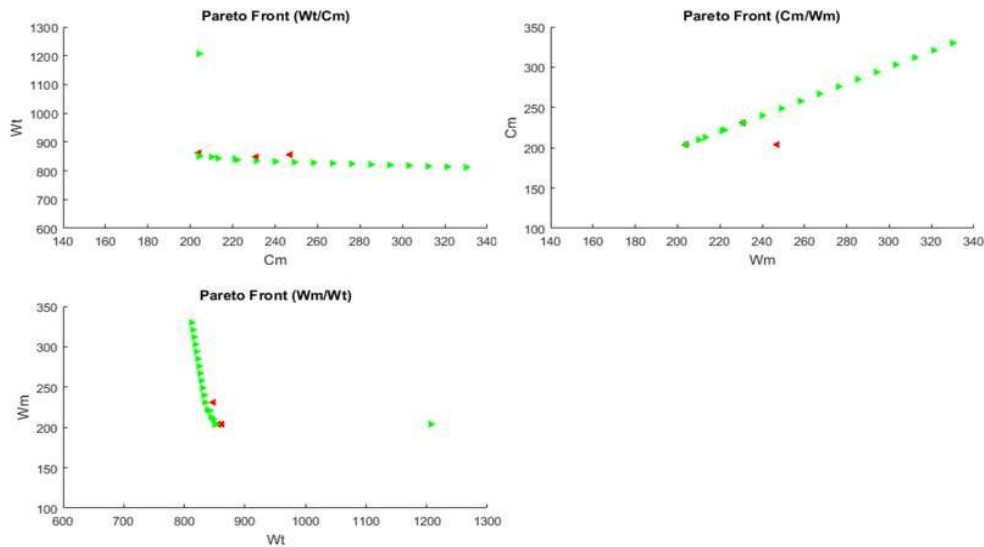


Figure 5-21 Pareto Frontier for Brandimarte mk03 Instance

Figure 5-21 presents the Pareto frontier for the mk04 instance. In the Wt/Cm

comparison, the results tend to form a secondary frontier, though it remains dominated by the Pareto frontier of the best-known solutions. This secondary frontier corresponds to the solutions identified by the algorithm, and enhancing its quality would require modifications to the scheduling methods within the proposed code.

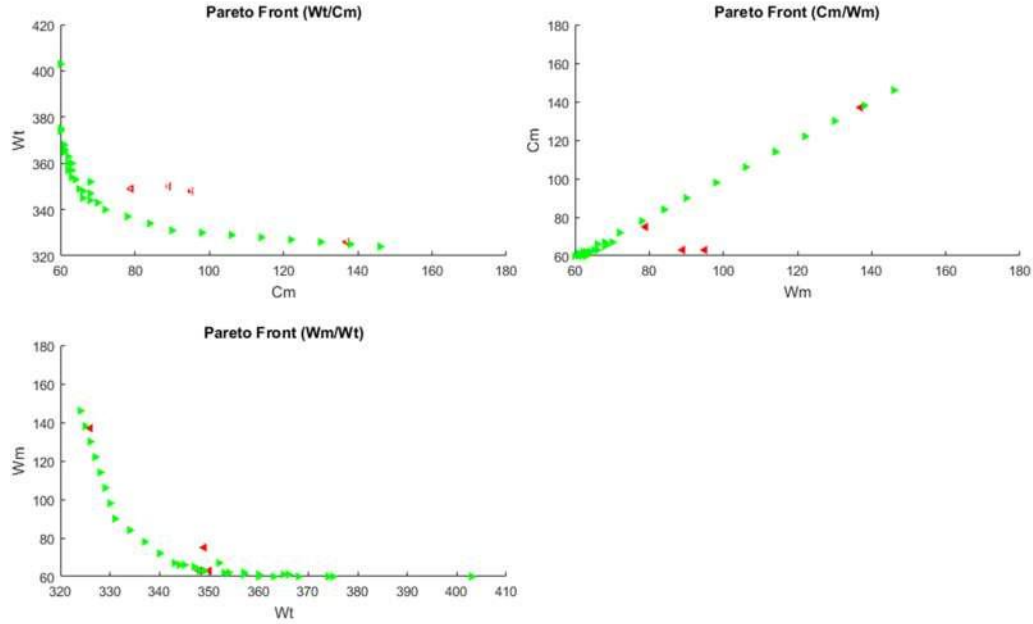


Figure 5-22 Pareto Frontier for Brandimarte mk04 Instance

Figure 5-22 illustrates the Pareto frontier for the mk05 instance, where the identified values align closely with the best-known results. This instance yielded the highest diversity of solutions, with over 46,000 distinct schedules generated for 13 key results, as depicted in Figure 5-23.

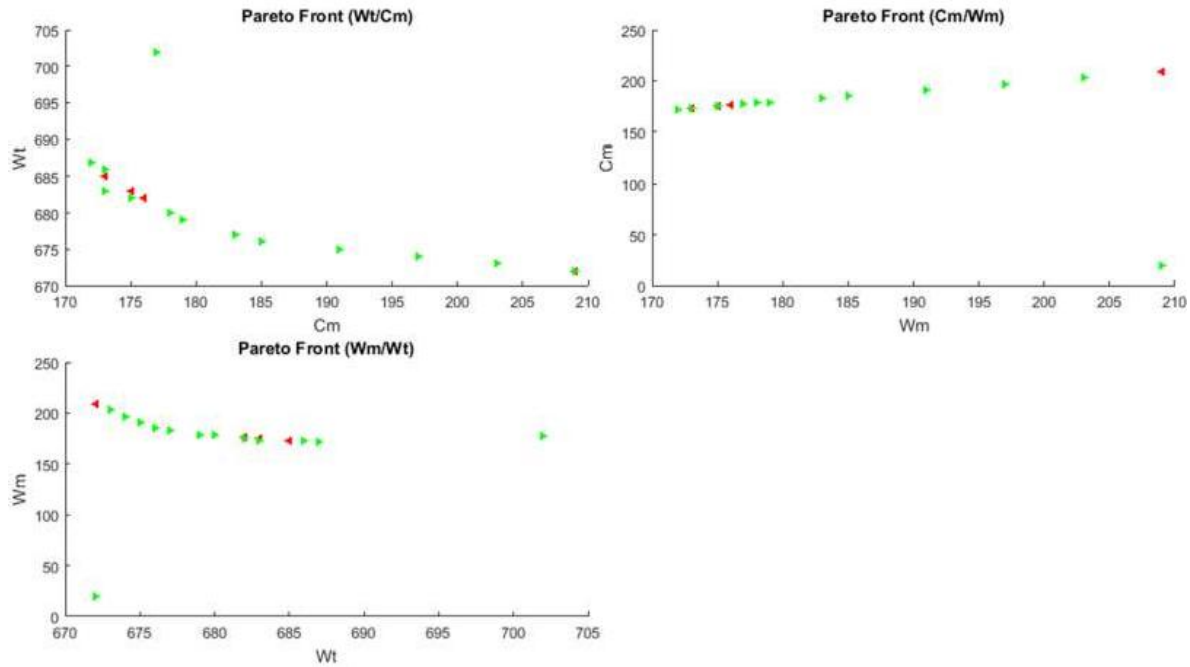


Figure 5-23 Pareto Frontier for Brandimarte mk05 Instance

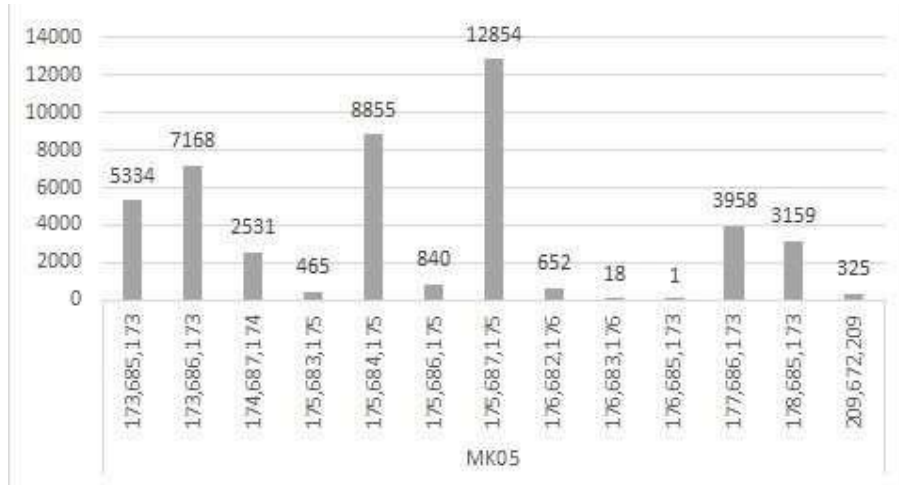


Figure 5-24 Diversity for the Brandimarte mk05 instance

Figure 5-24 shows the Pareto frontier for the 2008 model. Here the algorithm's solutions approach the most familiar benchmarks in literature. In Wt /Cm and Wm/Wt plots, the red markers (algorithm's results) closely mirror the shape of the green markers (literature benchmarks), with both sets clustered tightly. Specific solutions for this instance exhibited substantial diversity. As shown in Figure 5-25, the highest number of distinct solutions was identified for the objective combination, 144,692,144, totaling 170 unique

schedules.

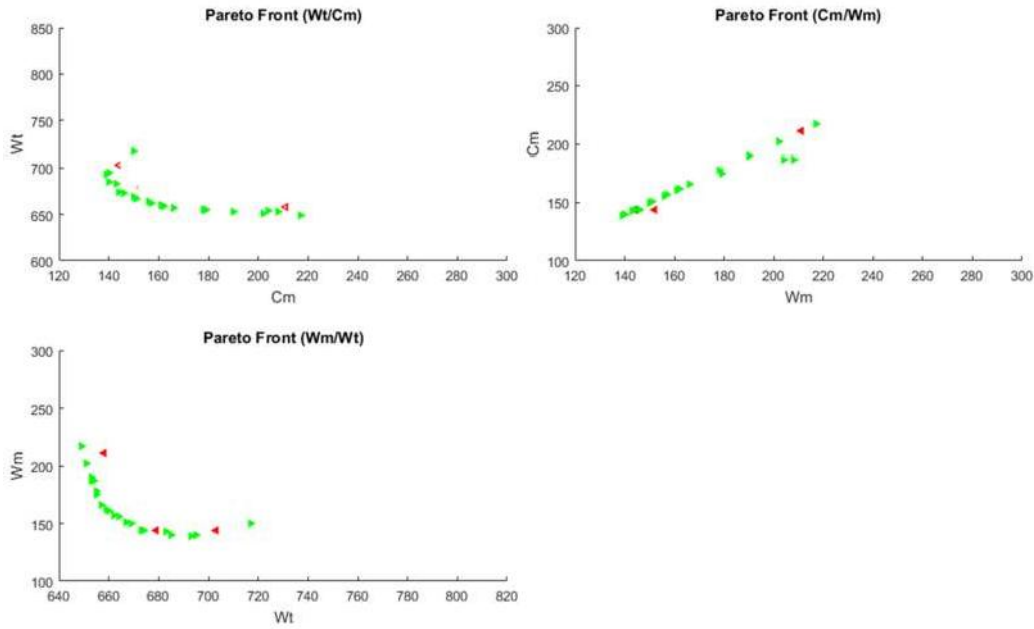


Figure 5-25 Pareto frontier for the Brandimarte mk07 instance

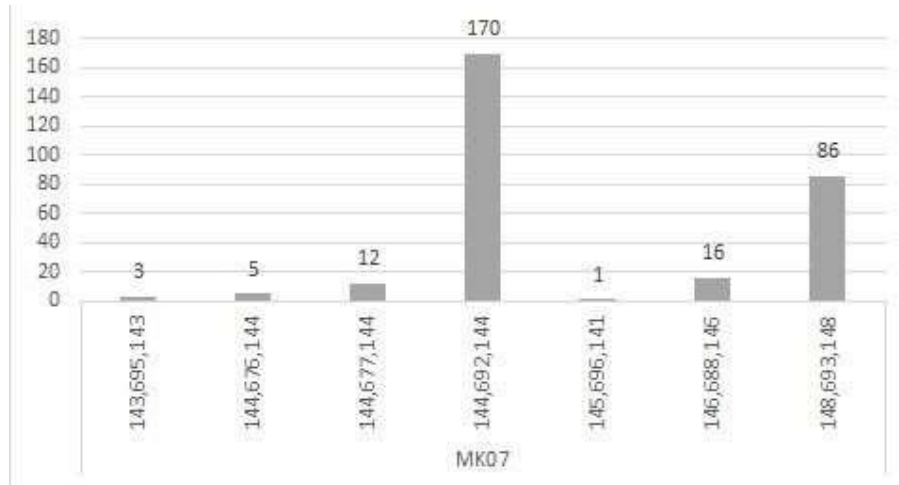


Figure 5-26 Diversity for the Brandimarte mk07 instance

Figure 5-26 illustrates the Pareto frontier for the mk08 instance. For this instance, the algorithm did not achieve the most well-known multi-objective values. However, when objectives are analyzed individually, nearly all the most notable results for Wt (workload total) and Wm (workload machine) were attained. Solution diversity remained low for this instance: only one value yielded 4 distinct configurations, and another produced 2 configurations, as detailed in Table 5-5.

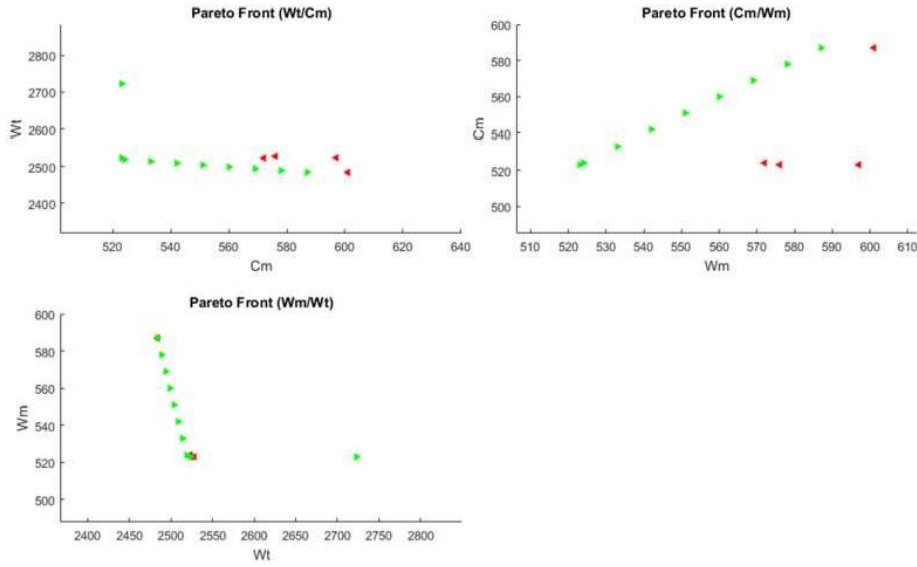


Figure 5-27 Pareto frontier for the mk08 instance

Figure 5-27 illustrates the Pareto frontier for the mk09 instance. For this instance, the algorithm failed to achieve the best-known results when analyzed using multi-objective or individual criteria. Solution diversity was notably low, with most objective values yielding only one distinct configuration. The sole exception was 369,2313,341, producing Wt solutions.

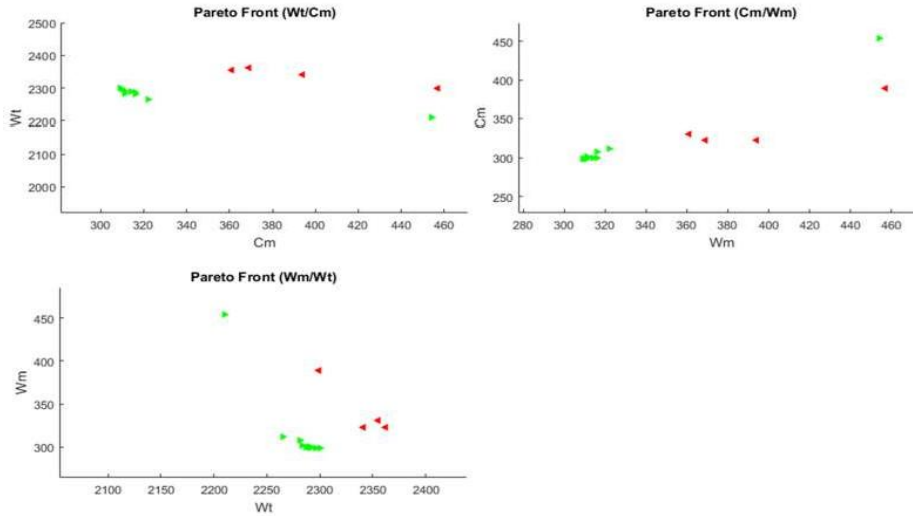


Figure 5-28 Pareto frontier for the mk09 instance

Although the results for the Brandimarte instances essentially did not achieve the best-known benchmarks as expected, a high diversity of viable solutions was observed across most instances.

CHAPTER 6 CONCLUSION AND FINAL CONSIDERATIONS

This work proposed an algorithm to deal with the FJSP using a bio-inspired collective intelligence method from computational intelligence called the Artificial Bee Colony. To enhance this method, Pareto dominance was proposed to support multi-objective optimization. A representation of results in the form of a Pareto frontier was adopted to improve the presentation of outcomes. Three distinct local inquiry mechanisms were created and proposed to ensure the algorithm's ability to handle and deliver satisfactory results across diverse instances.

The findings presented in the previous chapter compellingly demonstrate the effectiveness of the proposed algorithm. Exhibits strong capabilities in solving multi-objective FJSP problems and generating high solution diversity. In some instances, the algorithm did not achieve the best-known results due to limitations in the operation scheduling phase. However, for most instances where it fell short, it approached these benchmarks closely. For instances where the best-known multi-objective results were not achieved, the individual objective analysis revealed that most objectives (e.g., W_t and W_m) were attained except for C_m (makespan).

Although the algorithm did not identify all optimal solutions for the tested instances, it produced high solution diversity. This diversity is critical in real-world production scheduling, as it allows the same cost metrics to be maintained across objectives while implementing different schedules. As described earlier, specific schedules with the best-known results enable minimal machine utilization. For example, Figure 5-12 illustrates a schedule achieving one of the most notable results where Machine 8 remains unused. With such underutilized machines, maintenance can be performed during operation without disruptions, or the machine can be repurposed for other tasks or even removed entirely, given that optimal schedules exist without its use. Another advantage of diverse schedules is the ability to select configurations with higher or lower machine idle time, depending on

managerial needs. High diversity ensures schedules exhibit varied machine idle times.

During the result analysis, it became evident that the algorithm struggles with operation scheduling, which modifies the sequence in which actions are carried out. While the algorithm was designed for scheduling, this component is non-configurable and limited for large-scale solutions. This constraint makes the algorithm inefficient for instances with many operations per job. Based on these findings, the algorithm is inefficient for large-scale problems in its current form but could be adapted for such scenarios.

In conclusion, the results for the first three Kacem instances were excellent regarding multi-objective performance, as high diversity was achieved for the best-known values. None of the reference studies reviewed present similarly high diversity across all values for most benchmark instances. Consider only the most prominent results for Kacem's first three instances: 3,651 different solutions - 14 for the 4x5 instance, 82 for the 8x8, and 3,555 for the 10x10 instance.

From the results and analyses gathered, the recommended use cases for this algorithm are, without any modifications, fully flexible or partially flexible scenarios where several machines with heterogeneous processing times are available for the same operations. Instances resembling Brandimarte's benchmarks with limited machine availability and uniform processing times are not recommended due to the algorithm's scheduling bottleneck.

6.1 Future Work

Based on the above results and summarized results for Brandimarte instances described in the preceding section, a few improvements are suggested for the proposed algorithm. Adding a dedicated function to explore alternative scheduling sequences would be a major improvement, targeting results that currently challenge the algorithm greatly. One more recommendation is to parallelize certain code components. Given the large scale of some instances the algorithm takes a long computational time.

As a complement to rescheduling strategies, critical path analysis or the BSH

method proposed by Choo, Wong, and Khader (2016) is recommended to facilitate makespan (C_m) minimization in highly complex cases. A parameter tuning methodology is also suggested for configuration efficiency improvement. The Taguchi method (Taguchi, Elsayed, and Hsiang, 1988) was initially considered for this task but due to complexity, it was deferred as a secondary priority and is still unimplemented.

References

- [1] Wang S, Yu J, etc. An effective heuristic for flexible job-shop scheduling problems with maintenance activities[J]. *Computers & Industrial Engineering*, 2010, 59(3): 436-447.
- [2] Asadzadeh L, Zamanifar K, etc. A parallel artificial bee colony algorithm for the job shop scheduling problem with a dynamic migration strategy[J]. *Computers & Industrial Engineering*, 2016, 102: 359-367.
- [3] Garey M R, Johnson D S, Sethi R. The complexity of flow shop and job shop scheduling[J]. *Mathematics of Operations Research*, 1976, 1(2): 117-129.
- [4] Barnes J W, Chambers J B. Flexible job shop scheduling by tabu search [Technical report][J]. Graduate Program in Operations Research and Industrial Engineering, The University of Texas at Austin, 1996.
- [5] Yazdani M, Gholami M, Zandieh M, etc. A simulated annealing algorithm for flexible job-shop scheduling problem[J]. *Journal of Applied Sciences*, 2009, 9(4): 662-670.
- [6] Xing L N, Chen Y W, Wang P, etc. The knowledge-based ant colony optimization for flexible job shop scheduling problems[J]. *Applied Soft Computing*, 2010, 10(3): 888-896.
- [7] Li J Q, Pan Q K, Xie S X, etc. The hybrid artificial bee colony algorithm for flexible job shop scheduling problems[J]. *International Journal of Computers Communications & Control*, 2011, 6(2): 286-296.
- [8] Gao K Z, Suganthan P N, Pan Q K, etc. An improved artificial bee colony algorithm for flexible job-shop scheduling problems with fuzzy processing time[J]. *Expert Systems with Applications*, 2015, 42: 52-67.
- [9] Goldberg M C, Goldberg E F, Menezes M D S, etc. Quota traveling car renter problem: Model and evolutionary algorithm[J]. *Information Sciences*, 2016, 367: 232-245.
- [10] Kacem I, Hammadi S, Borne P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems[J]. *IEEE Transactions on Systems, Man, and Cybernetics: Part C*, 2002, 32(1): 1-13.
- [11] Zheng Y, Lian L, Mesghouni K. Comparative study of heuristics algorithms in solving flexible job shop scheduling problem with condition-based maintenance[J]. *Journal of Industrial Engineering and Management*, 2014, 7(2): 518-531.
- [12] Xia W, Wu Z. A practical hybrid optimization approach for multi-objective flexible job-shop scheduling problems[J]. *Computers & Industrial Engineering*, 2005, 48(2): 409-425.
- [13] Neale M, Cameron J R. Committee for Terotechnology: A guide to the condition monitoring of machinery[M]. HMSO Publications, 1979.
- [14] Zitzler E. SPEA2: Improving the performance of the strength pareto evolutionary algorithm[R]. Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 2001.
- [15] Li J Q, Pan Q K, Gao K Z. Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems[J]. *The International Journal of Advanced Manufacturing Technology*, 2011, 55(9-12): 1159-1169.

- [16] Deng Q, Gong G, Gong X, etc. A bee evolutionary guiding nondominated sorting algorithm genetic II for multiobjective flexible job-shop scheduling[J]. Computational Intelligence and Neuroscience, 2017, 2017: 1-20.
- [17] Kennedy J, Eberhart R. Particle swarm optimization[C]. In: Proceedings of the IEEE International Conference on Neural Networks, 1995: 1942-1948.
- [18] Li J Q, Pan Q K, Tasgetiren M F. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities[J]. Applied Mathematical Modelling, 2014, 38(3): 1111-1132.
- [19] Wang L, Zhou G, Xu Y, etc. An enhanced Pareto-based artificial bee colony algorithm for the multi-objective flexible job shop scheduling[J]. The International Journal of Advanced Manufacturing Technology, 2012, 60(9-12): 1111-1123.
- [20] Kato E R R, Aranha G D D A, Tsunaki R H. A new approach to solve the flexible job shop problem based on a hybrid particle swarm optimization and random-restart hill climbing[J]. Computers & Industrial Engineering, 2018, 125: 178-189.
- [21] Taguchi G, Hsiang T C. Quality Engineering in Production System[M], 1989.
- [22] Choo W M, Wong L P, Khader A T, etc. A modified bee colony optimization with local search approach for job shop scheduling problems relevant to bottleneck machines[J]. International Journal of Advanced Soft Computing Applications, 2016, 8(2): 52-78
- [23] Muthiah A, Rajkumar A, Rajkumar R. Hybridization of artificial bee colony algorithm with particle swarm optimization algorithm for flexible job shop scheduling[C]. In: 2016 International Conference on Energy Efficient Technologies for Sustainability, 2016: 896-903.
- [24] Karaboga D, Akay B. A comparative study of Artificial Bee Colony algorithm[J]. Applied Mathematics and Computation, 2009, 214(1): 108-132.
- [25] Dorigo M, Stützle T. Ant Colony Optimization[M]. MIT Press, 2004.
- [26] Kennedy J, Eberhart R. Particle Swarm Optimization[C]. In: Proceedings of the IEEE International Conference on Neural Networks, 1995: 1942-1948.
- [27] Blum C. Ant colony optimization: Introduction and recent trends[J]. Physics of Life Reviews, 2005, 2(4): 353-373.
- [28] Glover F, Laguna M. Tabu Search[M]. Kluwer Academic Publishers, 1997.
- [29] Engin M, Dogan B. A hybrid metaheuristic algorithm for the flexible job shop scheduling problem[J]. Computers & Industrial Engineering, 2011, 61(3): 465-475.
- [30] Cheng T C E. Analysis of local search procedures for scheduling problems[J]. International Journal of Production Research, 1997, 35(14): 4009-4021.
- [31] Pinedo M. Scheduling: Theory, Algorithms, and Systems[M]. Springer, 2016.
- [32] Li K, Tang G. A genetic algorithm for flexible job shop scheduling problem[J]. International Journal of Production Research, 2007, 45(2): 411-432.
- [33] Bektas T, Laporte G. The pollution-routing problem[J]. Transportation Science, 2006, 40(1): 141-150.
- [34] Zhang X, Gen M. Evolving divide-and-conquer strategies for constrained optimization[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2006, 36(6): 1386-1398.
- [35] Goel V, Chaudhary A. A hybrid differential evolution algorithm for solving flexible job shop scheduling problems[J]. Expert Systems with Applications, 2017, 89: 12-22.

- [36] Rouse D, Davison E. A new hybrid approach for solving the flexible job shop scheduling problem[J]. *International Journal of Advanced Manufacturing Technology*, 2006, 28(10): 1087-1095.
- [37] Blum C, Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison[J]. *ACM Computing Surveys*, 2003, 35(3): 268-308.
- [38] Talbi E G. Metaheuristics: From Design to Implementation[M]. Wiley, 2009.
- [39] Bousbia Salah A, Mhirech H, Bensaoula A. A novel hybrid metaheuristic approach for solving flexible job shop scheduling problem[J]. *Journal of Manufacturing Systems*, 2018, 47: 153-167.
- [40] Zhong R Y, Xu C, Klotz E. Smart manufacturing and the internet of things[J]. *International Journal of Advanced Manufacturing Technology*, 2015, 83(1-4): 509-514.
- [41] Chen L, Xu X. A review of scheduling techniques for complex manufacturing systems[J]. *Journal of Intelligent Manufacturing*, 2014, 25(4): 865-878.
- [42] Song H, Lee D. A survey of swarm intelligence algorithms in multi-objective optimization[J]. *IEEE Transactions on Evolutionary Computation*, 2012, 16(6): 757-771.
- [43] Zhang Y, Liu J. A review of artificial bee colony algorithm and its applications in scheduling problems[J]. *Journal of Heuristics*, 2010, 16(3): 369-390.
- [44] Karaboga D. An idea based on a honey bee swarm for numerical optimization[R]. Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
- [45] Shao X, Liu W, Liu Q, etc. Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem[J]. *International Journal of Advanced Manufacturing Technology*, 2013, 68(9-12): 2886-2902.
- [46] Liu Y, Ouyang D, Gu W, etc. A discrete artificial bee colony algorithm for permutation flow shop scheduling[C]. In: 2016 9th International Symposium on Computational Intelligence and Design, 2016: 161-164.
- [47] Thammano A, Phu-ang A. A hybrid artificial bee colony algorithm with local search for flexible job-shop scheduling problem[J]. *Procedia Computer Science*, 2013, 20: 96-101.
- [48] Deb K, Pratap A, Agarwal S, etc. A fast and elitist multi-objective genetic algorithm: NSGA-II[J]. *IEEE Transactions on Evolutionary Computation*, 2002, 6(2): 182-197.
- [49] Allahverdi A, Ng C T, Cheng T E, etc. A survey of scheduling problems with setup times or costs[J]. *European Journal of Operational Research*, 2008, 187(3): 985-1032.
- [50] Lee K S, Geem Z W. A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice[J]. *Computer Methods in Applied Mechanics and Engineering*, 2005, 194: 3902-3933.
- [51] Deb K, Agrawal S, Pratap A, etc. A fast elitist non-dominated sorting algorithm genetic for multi-objective optimization: NSGA-II[C]. In: *International Conference on Parallel Problem Solving from Nature*, 2002: 1-11.
- [52] Lu C L, Chiu S Y, Yen S J, etc. Solving the flexible job-shop scheduling problem based on multi-objective PSO with Pareto diversity search[J]. *International Journal of Intelligent Information Processing*, 2013, 4(3): 70.
- [53] Yue L, Guan Z, Saif U, etc. Hybrid Pareto artificial bee colony algorithm for multi-objective single machine group scheduling problem with sequence-dependent setup times and learning effects[J]. *SpringerPlus*, 2016, 5(1): 1-31.

- [54] Beasley J E. OR-Library[R]. People. Retrieved October 21, 2017.
- [55] Li X, Peng Z, Du B, etc. Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems[J]. Computers & Industrial Engineering, 2017, 113: 10-26.
- [56] Slack N, Chambers S, Johnston R. Administração da produção[M], 2009.
- [57] Chao L P, Lin T C, Lu C L. Flexible job-shop scheduling based on hybrid artificial bee colony algorithm with diversity index search for multiple decision-making search for multiple decisions making in manufacturing systems[J]. International Journal of Advanced Engineering and Management Research, 2017, 2(5): 1403-1417.
- [58] Gao K Z, Suganthan P N, Chua T J, etc. Pareto-based discrete harmony search algorithm for flexible job shop scheduling[C]. In: International Conference on Intelligent Systems Design and Applications, 2012: 953-956.

Appendix

List of Abbreviations

FJSP	Flexible job shop problem
JSP	Job shop problem
IA	Investigation Anathema
SOA	Stochastic Optimization Algorithm
AG	Algorithm Genetic
ACO	Ant Colony Optimization
ABC	Algorithm bee colony
FOLI	First out Last in
FOFI	First out First in
Cm	Makespan
Wt	Workload total
Wm	Workload machine
WWm	Weighted machine Workload
WWt	Weighted total Workload
WCm	Weighted makespan
SPO	Swarm Particle optimization
AI	Artificial Intelligence
BCO	Bee Colony optimization
PT	Processing time
TU	Time Unit
BSH	Bottleneck Shifting Heuristic
DDE	Due Date Earliest
MTPIAIN5	Modified two-phase improvement approach incorporating N5 neighborhood perturbation

SG	Selection Greedy
RLS	Ranking Linear selection
CPBABC	Combined Pareto-Based Artificial Bee Colony Algorithm
ASPEA	Advanced Strength Pareto Evolutionary Algorithm
NABCM	Noncontinuous Artificial Bee Colony Method
PABC	Artificial bee colony Pareto
DABCA	Discrete Artificial Bee Colony Algorithm
SMOA	Stepwise Machine Operation Assignment
NSGAI-BEG	Nondominated Sorting Genetic Algorithm II with Bee Evolutionary Guidance
FBS-BOA	Fibonacci Branch Search Based Optimization Algorithm

Catalogue of academic papers and achievements published during the degree study period

- [1] Dong S, You J, **MOHAMED EL-ATTAOUY**, Li M, etc. Investigation of Lamb wave modes recognition and acoustic emission source localization for steel plate based on golden jackal optimization VMD parameters and CWT[J]. Measurement, 2024, 242 : 116103. DOI: <https://doi.org/10.1016/j.measurement.2024.116103>. (SCI 2 区 TOP, JCR: Q1, IF=5.2, WOS:001352895400001)

Acknowledgments

I want to express my deepest gratitude to my advisor, Professor Li Ming. Since my enrollment, he has guided me through the vast ocean of academia with profound knowledge, a rigorous scholarly attitude, and selfless dedication. From the selection of my thesis topic to the organization of research ideas to every detail of writing and revising the thesis, he has provided meticulous guidance and patient assistance.

While writing my thesis, I have also received help from many classmates. I am grateful to my senior and junior fellow students for their care and encouragement in both study and life, sharing their research experiences and insights with me, making me feel the warmth of a big family during my graduate life.

I am thankful to my family. My parents have always been my biggest support. You have provided the conditions for me to learn and inspired me to follow my dreams. You encouraged me, comforted me through difficulties and setbacks, and helped me stand up again.

Lastly, I would like to thank everyone who has helped me during my master's studies.

尚德敏学 唯实惟新

