SCHEDULING OPTIMIZATION OF CELLULAR FLOWSHOP WITH SEQUENCE DEPENDENT SETUP TIMES

Al-Mehdi Mohamed M. Ibrahem

A thesis Submitted to the Faculty of Graduate Studies of the University of Manitoba in Partial Fulfillment of the Requirements of the Degree of

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering, University of Manitoba

Winnipeg, Manitoba, Canada

Copyright © 2015 by Al-Mehdi Ibrahem

Abstract

In cellular manufacturing systems, minimization of the completion time has a great impact on production time, material flow, and productivity. An effective scheduling is crucial to attaining the advantages of cellular manufacturing systems.

This dissertation focuses on solving the Flowshop Manufacturing Cell (cellular flowshop) Scheduling Problem with Sequence Dependent Setup Times (FMCSP with SDSTs) considering two performance measures: the total flow time as a mono objective, and the makespan and total flow time combined as a bi-criteria scheduling problem. The proposed problem is known to be the NP-hard problem because of its complexity.

Several metaheuristic algorithms based on Genetic Algorithm (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO) are developed for scheduling part families as well as jobs within each part family for FMCSP with SDSTs to minimize the total flow time. A local search method based on SA combined with PSO (named as PSO-SA) is proposed to enhance the intensification and improve the quality of the solution obtained by pure PSO. The effectiveness and efficiency of the proposed metaheuristics are evaluated based on the Relative Percentage Deviation (RPD) from its lower bound, and the robustness of metaheuristics. Results indicate that solutions found by PSO-SA algorithm performed similarly to the best available algorithms for small and medium size problems. Although there is a very small deviation from the best algorithm for the large problems.

A multi-objective Particle Swarm Optimization (MPSO) and a Multi-objective Simulated Annealing (MOSA) Algorithm are further proposed to solve the bi-criteria optimization problem to minimize the total flow time and makespan simultaneously. An improved PSO is combined with Threshold Acceptance (TA) algorithm to improve effectiveness of the proposed MPSO (named as IMPSO-TA) for the convergence of the obtained Pareto Front. The proposed algorithms are evaluated using several Quality Indicators (QI) measures for multi-objective optimization problems. The proposed algorithms can generate approximated Pareto Fronts in a reasonable CPU time. The proposed IMPSO-SA outperforms MOSA algorithm in terms of CPU time and minimize the objective functions.

Acknowledgements

I wish to offer my sincerest thanks to my advisors, Dr. Tarek Elmekkawy, and Dr. Qingjin Peng for their patience, guidance, friendship, support, and encouragement throughout this work. Besides my advisors, I would also like to thank the rest of my thesis committee: Dr. Srimantoorao Appadoo and Dr. Norman Richards for their valuable and constructive suggestions throughout the work. My deepest gratitude to Dr. Iqbal Shahid for his great help and valuable advice in writing the C++ codes used in this work. Also, I would like to express my great appreciation to Dr. Hussein Elmehdi for his endless support.

My sincere thanks also goes to University of Manitoba, Natural Sciences and Engineering Research Council of Canada (NSERC), Ministry of Higher Education and Scientific Research (MOHESR), and CancerCare Manitoba (CCMB), for providing me with financial support opportunities and leading me to working on exciting projects.

Finally, it is clear that I would not have been able to travel down this path without the help, support, and encouragement of my lovely family (brothers, sisters, and wife). Their encouragement has always been my greatest motivational force. There is no way to thank them enough for their efforts in helping me to earn my PhD degree.

To my respectful Father, my caring Mother, brothers and sisters, my compassionate Wife, and my lovely Daughters and Sons.

Table of Contents

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Objectives . . . 5
- 1.3 Work done and Research Assumptions . . . 6
- 1.4 Optimization Approaches . . . 7
- 1.5 List of Publications . . . 8
- 1.6 Thesis Outline . . . 9

2 Literature review 10
- 2.1 Background . . . 10
- 2.2 Complexity of the problem . . . 12
- 2.3 Classification of work on Flowshop Manufacturing Cell Scheduling Problems 12
  - 2.3.1 Makespan Minimization . . . 13
  - 2.3.2 Total Flow Time Minimization . . . 15
  - 2.3.3 Multiobjective Optimization in Flowshop Manufacturing Cell . . . 16
  - 2.3.4 Heuristics and Metaheuristics Approaches . . . 17
  - 2.3.5 Hybridization of Metaheuristics . . . 20
  - 2.3.6 Intensification and Diversification . . . 22
  - 2.3.7 Local Search . . . 22
  - 2.3.8 Parameter Tuning through Taguchi method . . . 24
  - 2.3.9 Performance Analysis and Robustness of Metaheuristics . . . 26
  - 2.3.10 Multi-objective Performance Measures . . . 27
  - 2.3.11 Summary . . . 28

3 A Mathematical Model to Solve a Flowshop Manufacturing Cell with Family Sequence Dependent Setup 29
- 3.1 Background . . . 29
- 3.2 Problem Formulation . . . 30
- 3.3 Mathematical Model . . . 30
- 3.4 Test problem Specifications . . . 34
  - 3.4.1 Schaller test problems . . . 34
  - 3.4.2 Salmasi's test problems . . . 35
- 3.5 Calculating the Total flow time of jobs . . . 38
  - 3.5.1 Illustrative example . . . 41

4 Metaheuristics for scheduling a flowshop manufacturing cell to minimize the total flow time 44
4.1 Introduction . . . 44
4.2 Problem definition and Solution representation . . . 45
4.3 Metaheuristic Approach . . . 46
4.4 Proposed Genetic Algorithm (GA) . . . 47
4.4.1 Solution representation . . . 47
4.4.2 Selection . . . 48
4.4.3 Crossover Operator . . . 49
4.4.4 Mutation Operator . . . 50
4.4.5 Results of the proposed GA algorithm . . . 50
4.5 Particle Swarm Optimization (PSO) Algorithm . . . 58
4.5.1 Initial Swarm . . . 61
4.5.2 Ranked Order Value (ROV) . . . 63
4.5.3 Results of the proposed PSO algorithm . . . 64
4.6 Hybrid Algorithms . . . 70
4.7 Simulated Annealing (SA) . . . 71
4.7.1 SA procedure . . . 73
4.7.2 Selecting the parameters setting of the SA algorithms using the Taguchi method . . . 74
4.7.3 Results of proposed SA algorithm . . . 78
4.8 Hybrid Particle Swarm Optimization algorithm . . . 84
4.8.1 Results of the PSO-SA algorithm . . . 86
4.9 Performance Analysis and Discussion . . . 90
4.9.1 Relative Percentage Deviation (RPD) . . . 90
4.9.2 Robustness of metaheuristics . . . 92
4.10 Summary and conclusions . . . 99

5 Metaheuristics for Solving the Bi-criteria Scheduling Problem of Flowshop Manufacturing cell with Sequence Dependent Setup Times 101
5.1 Introduction . . . 101
5.2 Principles of Multi-objective optimization problem . . . 102
5.2.1 Pareto Dominance . . . 103
5.3 Classical Methods . . . 104
5.3.1 Weighting method . . . 105
5.3.2 Constraint Method . . . 105
5.4 Evolutionary Algorithms . . . 106
5.5 Proposed Algorithms . . . 108
5.6 Multi-objective Particle swarm optimization Algorithm . . . 109
5.6.1 Initialization . . . 111
5.6.2 Pareto Search . . . 111
5.6.3 Computational results . . . 112
5.7 Multi-objective Simulated Annealing Algorithm . . . 115

5.7.1 Initial Solutions . . . 115
5.7.2 Neighbourhood Generation . . . 116
5.7.3 Computational results . . . 118
5.8 An Improved Multi-objective Particle swarm optimization algorithm . . . 119
5.8.1 An Improved Multi-objective Particle Swarm Optimization (IMPSO) 119
5.8.2 Local Search based on Threshold Acceptance (LS-TA) . . . 121
5.9 The proposed IMPSO-TA algorithm . . . 122
5.9.1 Computational results . . . 122
5.10 Performance Analysis and Discussion . . . 127
5.10.1 Performance measures based on Quality Indicators . . . 130
5.10.2 Discussion of the quality measures results . . . 132
5.11 Summary . . . 139

6 Conclusion 140
6.1 Research Contributions . . . 140
6.1.1 Scheduling a flowshop manufacturing cell with sequence-dependent family setup times . . . 140
6.1.2 Solving the Bi-criteria Scheduling Problem of Flowshop Manufacturing cell with Sequence Dependent Setup Times . . . 142
6.2 Recommendations for Future Research . . . 143

APPENDICES 162

A Quality Indicators code 162

List of Figures

1.1 Thesis Outline 9

2.1 Classification of existing literature on a flowshop manufacturing cell scheduling problem 13
2.2 Hybridization of particle swarm optimization and simulated annealing 23

3.1 Test Problem specifications 37
3.2 Completion time of a job in the case of sequence-dependent setup problem . 39
3.3 Gantt chart for sequence I 42
3.4 Gantt chart for sequence II 43
3.5 Gantt chart for sequence III 43

4.1 The solution structure 46
4.2 The crossover operator 50
4.3 Performance of proposed GA vs. ACO for large test problems 51
4.4 Performance of proposed PSO vs. ACO for large test problems 66
4.5 Hybridization schemes of PSO and SA 70
4.6 The main effect plot of the SA algorithm 76
4.7 The interaction plot of the parameter vs. the signal-to-noise ratio 77
4.8 The interaction between the initial and final temp. 77
4.9 The interaction between the final temp. and cooling schedule 78
4.10 Iteration between cooling schedule and cooling rate 79
4.11 Comparison between the proposed SA and ACO for small size problems . . . 80
4.12 Comparison between the proposed SA and ACO for large size problems . . . 80
4.13 Performance of the proposed PSO-SA vs. ACO for 2-machine problems . . . 86
4.14 RPD for the two machine test problems 91
4.15 RPD for the six-machine test problems 92
4.16 The performance of the proposed metaheuristic algorithms 94
4.17 The performance of the proposed metaheuristic algorithms 95
4.18 Interaction plots of number of familes vs. the proposed algorithms 96
4.19 Interaction plots of maximum jobs in family vs. the proposed algorithms . . 97
4.20 Interval plot of RPD of the all test problems 98
4.21 Interval plot for two-machine problems 98
4.22 Interval plot for six-machine problems 99

5.1 The average CPU time for the proposed algorithms 128
5.2 The percentage deviation from makespan for small test problems 129
5.3 The percentage deviation from makespan for large test problems 129
5.4 Pareto fronts obtained by MOSA and IMPSO-TA (SSU55) 137
5.5 Pareto fronts obtained by MOSA and IMPSO-TA (LSU66) 137
5.6 Pareto fronts obtained by MOSA and IMPSO-TA (LSU88) 138
5.7 Pareto fronts obtained by MOSA and IMPSO-TA (SSU34) 138

List of Tables

3.1 The setup time of each machine in two-machine instances 37
3.2 The setup time of each machine in six-machine instances 37
3.3 Processing time of jobs over machines in a cell ($p_{f,j,i}$) 41
3.4 Setup times of part families over machines ($Se_{l,f,i}$) 43

4.1 Results of small test problems for two machine problems obtained by GA algorithm. 52
4.2 Results of medium test problems for two machine problems obtained by GA algorithm. 53
4.3 Results of large test problems for two machine problems obtained by GA algorithm. 54
4.4 Results of small test problems for six machine problems obtained by GA algorithm. 55
4.5 Results of medium test problems for six machine problems obtained by GA algorithm. 56
4.6 Results of large test problems for six machine problems obtained by GA algorithm. 57
4.7 The maximum and minimum values of PSO parameters 61
4.8 Results of small test problems for two machine problems obtained by PSO algorithm. 66
4.9 Results of medium test problems for two machine problems obtained by PSO algorithm. 67
4.10 Results of large test problems for two machine problems obtained by PSO algorithm. 67
4.11 Results of small test problems for six machine problems obtained by PSO algorithm 68
4.12 Results of medium test problems for six machine problems obtained by PSO algorithm 69
4.13 The results of large test problems for six machine problems obtained by PSO algorithm 69
4.14 List of the SA parameters and the corresponding values 75
4.15 Results of the L27 design 75
4.16 The results of small test problems for two machine problems obtained by SA algorithm. 81

4.17 The results of medium test problems for two machine problems obtained by SA algorithm. 81
4.18 The results of large test problems for two machine problems obtained by SA algorithm. 82
4.19 The results of small test problems for six machine problems obtained by SA algorithm. 82
4.20 The results of medium test problems for six machine problems obtained by SA algorithm. 83
4.21 The results of large test problems for six machine problems obtained by SA algorithm. 83
4.22 Results of small test problems for two machine problems obtained by PSO-SA algorithm. 87
4.23 Results of medium test problems for two machine problems obtained by PSO-SA algorithm. 87
4.24 Results of large test problems for two machine problems obtained by PSO-SA algorithm 88
4.25 Results of small test problems for six machine problems obtained by PSO-SA algorithm. 88
4.26 Results of medium test problems for six machine problems obtained by PSO-SA algorithm. 89
4.27 Results of large test problems for six machine problems obtained by PSO-SA algorithm. 89
4.28 The levels of the problem's factors 93
4.29 The experimental design layout and results 93
4.30 The levels of the problem's factors 99

5.1 Results of the proposed MPSO for small setup instances 113
5.2 Results of the proposed MPSO for medium setup instances 113
5.3 Results of the proposed MPSO for large setup instances 114
5.4 Results of the proposed MPSO for Large setup instances 114
5.5 Results of the proposed MOSA for small setup instances 118
5.6 Results of the proposed MOSA for medium setup instances 119
5.7 Results of the proposed MOSA for large setup instances 120
5.8 Results of the proposed MOSA for Large setup instances 120
5.9 Results of the proposed IMPSO-TA for small setup instances 125
5.10 Results of the proposed IMPSO-TA for medium setup instances 125
5.11 Results of the proposed IMPSO-TA for large setup instances 126
5.12 Results of the proposed IMPSO-TA for Large setup instances 126
5.13 Results of QI measures of IMPSO-TA VS MOSA for small setup time instances 134
5.14 Results of QI measures of IMPSO-TA VS MOSA for medium setup times instances 135
5.15 Results of QI measures of IMPSO-TA VS MOSA for large setup times instances 136

List of Algorithms

4.1 The steps of the proposed Genetic Algorithm 49

4.2 The steps of the proposed PSO algorithm 64

4.3 The steps of the proposed SA algorithm 72

4.4 The steps of the proposed PSO-SA algorithm 85

5.1 The steps of the proposed MPSO algorithm 110

5.2 The steps of the proposed MOSA algorithm 117

5.3 The Pseudo code for Loacl Search Threshold Acceptance algorithm LS-TA . 123

5.4 The steps of proposed IMPSO-TA algorithm 124

Acronyms

ACO Ant Colony Optimization

CDS Campbell-Dudek-Smith

CMS Cellular Manufacturing System

CM Coverage Metric

DOE Design of Experiments

EAs Evolutionary Algorithms

FMCSP Flowshop Manufacturing Cell Scheduling Problem

FSSP Flow Shop Scheduling Problem

GA Genetic Algorithm

GSA Genetic and Simulated Annealing Algorithm

GT Group Technology

HACO Hybrid Ant Colony Optimization

IMPSO Improved Multiple Particle Swarm Optimization

JIT Just-in-Time

LB Lower Bound

LS Local Search

MA Memetic Algorithm

MILP Mixed Integer Linear Programming

MOGA Multi-Objective Genetic Algorithm

MPSO Multi-Objective Particle Swarm Optimization

MOSA Multi-Objective Simulated Annealing

MS Makespan

NEH Nawaz-Enscore-Ham

NP Non-Parametric

OA Orthogonal Array

QIs Quality Indicators

PSO Particle Swarm Optimization

RPD Relative Percentage Deviation

SDSTs Sequence-Dependent Setup Times

SISTs Sequence-Independent Setup Times

SNR Signal-To-Noise Ratio

SA Simulated Annealing

TA Threshold Acceptance

TFT Total Flow Time

TS Tabu Search

TLMSA Two-Level Multi-Star Simulated Annealing

WIP Work- In-Process

Chapter 1

Introduction

1.1 Background

The rapid changes of consumers needs have caused a dynamic and highly volatile business environment. Manufacturing organizations seek productive efficiency or cost-effectiveness solutions by competing via fast time to market and low production costs [1]. Cellular Manufacturing (CM) uses Group Technology (GT) in grouping machines according to parts processed. GT is a philosophy of putting the products with similar design or manufacturing characteristics or both in one group [2]. CM aims to improve the productivity by grouping the parts into part families based on their similarity such as production requirements and setup times. Maximum productive efficiency is defined as the maximum output that can be achieved for given inputs [3].The CM benefits reported by [4] include:

- Reduction in work-in-process inventory and lead time; consequently, leads to reduce floor space requirements

- Reduction in setup time as a result of grouping the jobs into part families based on similarities in production requirements
- Simplifying work flow leads to reduction in material handling

Scheduling is a decision-making process that deals with the allocation of resources to tasks over given periods and its goal is to optimize one or more objectives [5]. An efficient job scheduling is a crucial aspect of any manufacturing environment. Thus, the next step for improving the efficiency of a manufacturing cell is to find the best sequence of processing the assigned parts (jobs). This study aims to find the best sequence of part families as well as the jobs or parts in a part family in order to optimize performance measures. This problem is known as Cellular Flowshop Scheduling Problem or Flowshop Manufacturing Cell Scheduling Problem (FMCSP) [6]. In this dissertation, FMCSP will be used to refer to the problem. Setup activities involve obtaining tools, returning tools, cleaning up, setting the required jigs and fixtures, and adjusting tools. In scheduling problems, there are two types of setup times or setup costs: 1) Sequence-Independent Setup Times (SISTs) in which setup time/cost depends solely on the job to be processed regardless of its preceding task. 2) Sequence-Dependent Setup Times (SDSTs) in which setup time/cost depends on both the job to be processed and its preceding job. In the second case, the setup times not only depend on the current and previous jobs but also the current and preceding part family, thus compounding the optimization problem. The purpose of this research is to study the FMCSP with SDSTs.

There are two common relevant objectives in studying FMCSPs: 1) minimization of the

makespan (the completion time of the last job on last machine), and 2) minimization of the total flow time (the sum of the completion times) [7].

There is limited work on the FMCSP with SDSTs [8], and research has paid little attention to the minimization of the TFT, which is an important measure to improve productivity. Due to the NP-hard nature of the considered problem, using a purely mathematical programming optimization approach to determine an optimal solution may not be efficient in practice especially with large-size problems. On the other hand, simple heuristics such as the Campbell-Dudek-Smith (CDM) algorithm [9], and the NEH algorithm [10]; usually yields poor quality solutions. Metaheuristics, such as Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithm (GA) have been proven efficient for many NP-hard optimization problems, including many classes of scheduling problems. There is a need for more algorithms to solve the considered problem, and the need arises to find efficient algorithms to optimize the completion time performance measures. Moreover, developing an effective algorithm to optimize the TFT and the MS simultaneously as a bi-objective optimization would be a useful achievement. These efficient algorithms will help the production managers and decision-makers improve the throughput rate of the manufacturing system. Therefore, this research addresses the problem of scheduling part families as well as parts (jobs) within each part family in a Flowshop Manufacturing Cell Scheduling with sequence dependent setup times. The objective is to minimize the total flow time (TFT) of jobs in each family. In addition, this study investigates minimizing both TFT and MS simultaneously.

This research is inspired by the practical implementation of group technology and cellular manufacturing in industry. Numerous practical applications of FMCSP with SDSTs are reported in the literature, such as printed circuit board [11], semiconductor industry [12], electronics manufacturing [13], and automotive paint [14, 15]. For instance, in manufacturing of clothes, before dyeing a yarn, the dyeing tank (machine that processes the yarn to be dyed) needs to be cleaned. The setup (cleaning) time necessary to prepare for dyeing a coming job can be different, depending on the colors of the coming yarn and the one just finishing dyeing [16]. If the preceding job is blue and the following one is white, then a full cleaning of the dyeing tank is required. However, if the preceding job is white color and the following one is blue, then just a rough cleaning the dyeing tank is necessary. Therefore, time required when the tank is changed from white to blue is less than time required to change from blue to white.

Pearn et al [17] considered the integrated circuit assembly scheduling problem (ICASP), which has many real-world applications. In the IC assembly factories, jobs are assigned processing priorities and are grouped by their product types, which must be processed on groups of identical parallel machines. Furthermore, the job processing time depends on the product type, and the machine setup time is sequentially dependent on the sequence of jobs to be processed. The major four process stages of the IC manufacturing include wafer fabrication, wafer probe, IC assembly, and final testing. The processing time of a job may vary, depending on the product type.

Besides the wide application of the Cellular Manufacturing Systems (CMS), the high level of competitiveness in the current market has forced manufacturers to improve the produc-

tivity and use small-lot-size production systems. The complexity of production systems has therefore increased because of the variety of product items, variety of production processes, and difficulty in production planning, scheduling and control. A current popular trend is to produce small batches with short lead times to rapidly respond to changes in customer requirements [18]. The intense competition has forced manufacturing organizations to find methods to reduce cost and to deliver product on time with high customer satisfaction [19]. Cellular manufacturing helps to highlight the economic advantage of batch manufacturing.

In a cellular manufacturing environment, machines are grouped into cells. Each cell is dedicated to the production of a specific part family. A cell consists of machines or workstations, arranged in a processing sequence. Optimization of production scheduling in cellular manufacturing is the focus of this thesis.

1.2 Objectives

The main objective of this thesis is to design efficient metaheuristic algorithms to schedule FMCSP with SDSTs, thus optimizing the completion time of jobs. To realize this goal, the minimization of the total flow time is considered the main performance criteria to be studied with makespan and total flow time as a bi-objective scheduling problem to optimize the performance of a cellular manufacturing system. The following objectives are further steps in reaching the main goal:

- Develop metaheuristic algorithms to minimize the total flow time (sum of completion times)

  - Develop metaheuristic algorithms based on Genetic Algorithm (GA), and Particle Swarm Optimization (PSO) to minimize the total flow time;
  - Develop a hybrid algorithm (PSO algorithm with SA) to improve the effectiveness of the algorithm especially for large-sized instances;
  - Study the robustness of the proposed metaheuristic algorithms;

- Propose an efficient algorithm to optimize total flow time and makespan simultaneously:
  - Develop Multiple Objective Particle Swarm Optimization (MPSO), Multiple Simulated Annealing (MOSA) and Improved Particle Swarm Optimization (IMPSO) algorithms to solve the studied problem;
  - Evaluate the solution quality of the proposed algorithms based on some standard quality indicators.

1.3 Work done and Research Assumptions

The research will address two problems based on the objective functions. The first problem is the minimization of the TFT; this problem is classified as: $F_m \setminus fmls, Se_{lki}, prum \setminus \sum_{j=1}^{N} C_j$. The second problem is to minimize the total flow and makespan simultaneously which is denoted as: $F_m \setminus fmls, Se_{lki}, prum \setminus C_{max}, \sum_{j=1}^{N} C_j$. The following assumptions are used in this research:

- The number of parts (jobs), their processing times, the number of part families, and their setup times are known in advance.

- The sequence of parts and part families are the same on all machines (permutation scheduling).
- Once a part starts to be processed on a machine, the process cannot be interrupted before completion (pre-emption is disallowed).
- Each machine can handle only one part (job) at a time.
- All buffers have unlimited size.
- The jobs belonging to each family should be processed without any preemption by other jobs of other families (group technology assumption).
- The ready time of all parts is zero, i.e, all parts in all part families are available for processing at the start time.
- Setup time depends on both the part to be processed and its preceding part. There is a minor setup time among parts within a family but there is a major (considerable) setup time among the part families.

1.4 Optimization Approaches

Since the problem is classified as NP-hard problems [20], the computation time required to find the optimal solution could be extremely high. Therefore, exact methods such as mathematical modeling are impractical. Nevertheless, introducing the mathematical model helps to get a better understanding of the problem and its constraints. Alternatively, approximate methods such as metaheuristics are commonly used. Metaheuristics are defined by

Osman, as "iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions" [21]. The successful implementation of the metaheuristics in scheduling fields inspired the author to propose several metaheuristics to optimize single and bi-objective performance measures (bi-criteria scheduling problem).

1.5 List of Publications

Results presented in this dissertation have been published or submitted for publishing, as follows:

- Ibrahem, A., Elmekkawy, T., and Peng, Q. (2014). Robust Metaheuristics for Scheduling Cellular Flowshop with Family Sequence-Dependent Setup Times. Procedia CIRP, 17C, 428-433. doi:10.1016-j.procir.2014.01.072

- Ibrahem, A., Elmekkawy, T., and Peng, (submitted in August 19, 2014). An Efficient Hybridized Particle Swarm optimization Algorithm for Scheduling a flowline manufacturing cell with Sequence Dependent Family Setup Times. Journal of Scheduling. Manuscript No. JOSH-D-14-00109.

- Ibrahem, A., Elmekkawy, T., and Peng, (submitted in Jan 29, 2015). Metaheuristics for solving the Bi-criteria Flowshop Manufacturing Cell Scheduling Problem with Sequence Dependent setup times. International Journal of Production Research. Manuscript ID is TPRS-2015-IJPR-0144.

- Ibrahem, A. Elmekkawy, T, and Peng, Q., (2013), A MultiObjective Genetic Algorithm for scheduling a Cellular Manufacturing Flowshop. 55th CORS Annual Conference, Vancouver, Canada

1.6 Thesis Outline

The outline of this thesis is organized as follows (Figure 1.1): Following the introduction in Chapter 1, the literature review is presented in Chapter 2. In Chapter 3, the mathematical model of the proposed problem is discussed. Chapter 4 presents the proposed metaheuristic algorithms to minimize the total flow time while Chapter 5 discusses the proposed metaheuristics algorithms for minimization of total flow time and makespan simultaneously. Finally, Chapter 6 presents conclusions and recommendations for future research directions.

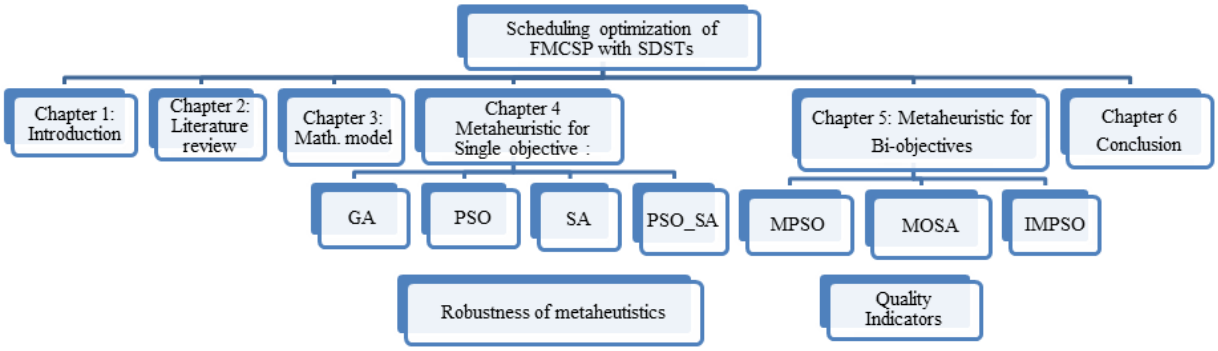

Figure 1.1: Thesis Outline

Chapter 2

Literature review

2.1 Background

Inevitably, efficient job scheduling is a crucial aspect of any manufacturing environment and it has a significant impact on the operation costs [22]. In many manufacturing and assembly facilities, each job has to undergo a series of operations. Often, these operations have to be done on all jobs in the same order implying that jobs have to follow the same route. If machines are assumed to be set up in series, the manufacturing environment is referred to as a flowshop [5]. Further, if changes in the sequence of the jobs are not allowed between machines, then it is called permutation flowshop. Flowshop scheduling has been an interesting area of scheduling since 1954 when Johnson published the first paper in this area [23]. In the flowshop environment, a set of N jobs is to be processed over a number of M machines. The sequence of all jobs over all machines is the same. Each job can be processed only on one machine at a time and only once on each machine. Moreover, each

machine can process only one job at a time. In FMCSPs, the set of N jobs are grouped into different part families according to their similar attributes or production techniques. The sequence of part families and the sequence of jobs within each part family are the same on all machines. Furthermore, the required setup time on a machine used for switching between jobs that belong to the same part family is usually considered a part of the processing time of the jobs or it can be neglected. Yet, the setup required for switching the process between part families is more crucial. The setup time depends on the part family to be processed and the preceding part family. For instance, if we consider a painting cell in which parts are grouped according to their color. The time needed to switch from one color to another depends on the sequence in which part families are scheduled. For example, the required setup time to switch from white to blue is shorter than that needed to change from blue to white. Such a problem is called Flowshop Manufacturing Cell Scheduling problem with Sequence Dependent Setup Times (FMCSPs with SDSTs). A comprehensive study of the scheduling problems with setup times is provided by Allahverdi et al [24, 25].

The objective of such a flowshop scheduling problem is to optimize performance criteria by finding the best sequence of the part families, and jobs with each family. The commonly used performance measures are the Makespan (MS), Total Flow Time (TFT), and Mean Flow Time (MTFT). Makespan in a flowshop problem can be defined as the completion time, which is the total time to process all the jobs. Makespan criterion is usually used to weigh the utilizations of machines. As long as the makespan is minimized, the production efficiency can be improved. The flow time refers to the time that a part spends from its entry into the system to completion of all the operations. The TFT is the sum of flow times

of all the parts in the system. By minimizing TFT, the work-in-process inventory can be reduced. In this dissertation, the focus will be on the TFT criterion as a single objective performance measure, and the TFT and MS wil then be optimized as Bi-objective criteria.

2.2 Complexity of the problem

Gupta and Darrow concluded that the two machine flowshop scheduling problem where the set-up time of a job depends on the immediately preceding job (sequence dependent job scheduling) is a strongly NP-hard problem [26]. Furthermore, [27] they proved that minimization of mean flow time in a multi-stage flow shop problem is an NP- hard problem. These problems form a subset of what we consider in this dissertation (two machines are a special case of multiple machines, and mean flow time is the average of the flow times). It can be concluded that the proposed problem is the NP-hard problem as well.

2.3 Classification of work on Flowshop Manufacturing Cell Scheduling Problems

A classification of the previous work on FMCSPs, also known as group scheduling or Flowline scheduling problem with SDSTs is shown in Figure 2.1 where the current research is highlighted in bold.

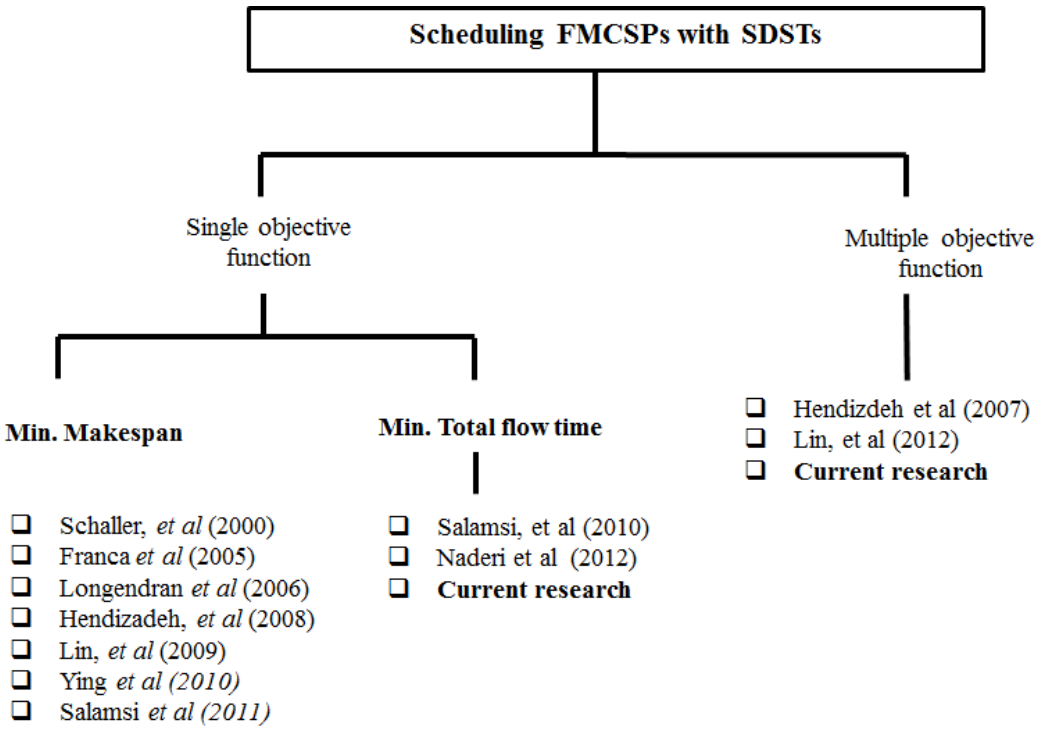

Figure 2.1: Classification of existing literature on a flowshop manufacturing cell scheduling problem

2.3.1 Makespan Minimization

Scheduling a flowshop of cellular manufacturing systems with family setup times was studied for the first time by Schaller, Gupta, and Vakharia [28], they developed several heuristic algorithms with minimization of the makespan as the criterion. They developed a lower bounding method to evaluate the solution quality of the proposed algorithms. Franca et al [29] proposed Genetic algorithm (GA) and a Memetic algorithm (MA) with local search for the makespan minimization. These scholars concluded that the solution quality of the MA outperformed the available algorithms. Hendizadeh et al [15] presented various TS based meta-heuristics for FMCSPs with SDSTs to minimize makespan. They proposed the

concepts of elitism and the acceptance of worse moves from SA to improve intensification and diversification. SA meta-heuristic algorithm proposed by Lin et al [30] to minimize the makespan. Lin et al [31] proposed a Multi-star Simulated Annealing (MSA) based meta-heuristic to minimize the makespan for the aforementioned problem and showing that the performance of their algorithm is better than the MA proposed by Franca [29]. Ying et al [32] presented an extensive computational investigation concerning the performance evaluation of non-permutation versus permutation schedules for the flowline manufacturing cell with sequence dependent setup times.

Logendran et al [33] investigated the two-machine problem in the presence of sequence-dependent setup times to minimize the makespan. A hybrid Ant Colony Optimization (ACO) algorithm proposed by Salmasi [34] to solve the studied problem in this thesis with minimization of makespan; they concluded that the ACO algorithm had a performance that was superior to the best available algorithm. Further, a lower bounding method was suggested for the flowshop sequence dependent groups scheduling problems by minimizing the makespan [35]. Salmasi showed that the proposed Lower Bound (LB) had a better performance than all the available ones in literature. Bouabda et al addressed the permutation flowline manufacturing cell with sequence dependent family setup times problem with the objective of minimizing the makespan [36], they proposed a new way of hybridizing by using branch and bound algorithm as the embedded local search algorithm inside an iterated local search algorithm. Neufeld et al [37] studied the permutation flowshop group scheduling problem to minimize makespan. They have shown that generating job sequences by minimizing the sum of inserted machine idle times instead of makespan on the first level (sequence of jobs

within part families) of scheduling and using the schedule heads on the second level (part families have to be sequenced) can lead to significant improvements for some test problems. Most of the work in literature has focused on the makespan minimization. Further, the performance of proposed algorithms has been evaluated based on the deviation from the lower bounds. In this research, the performance of proposed metaheuristic algorithms is evaluated based on the robustness of metaheuristics.

2.3.2 Total Flow Time Minimization

The minimization of the total flow time was studied for the first time by Salmasi et al [14]. They developed two meta-heuristics based on Tabu Search (TS) and Hybrid Ant Colony Optimization (ACO) algorithm to minimize TFT. They showed that the algorithm had a higher performance than the TS algorithm and it has been considered as the best available metaheuristic. They developed an efficient lower bound based on the Branch- and-Price for total flow time minimization. In 2012, Naderi and Salmasi [20] studied the proposed problem and developed two different mixed integer linear programming models showing that the models are effective in solving small and medium sized problems, and provide optimum solution given a reasonable time. They developed a hybrid genetic and simulated annealing algorithm, called (GSA) to solve the problem heuristically and proved that the proposed mathematical models and the proposed metaheuristic algorithm outperformed the available algorithms in the literature. As a result of the limited work done on the TFT minimization, more algorithms are needed to minimize the TFT. The contribution of the work is to develop several algorithms based on the implementation of pure metaheuristics (GA, PSO), and

hybrid algorithms combining PSO and SA in different schemes. Furthermore, based on the existing research, the effectiveness of algorithms is evaluated based on the deviation from the lower bounds. Yet, the robustness of the proposed algorithm should be considered to find the most robust algorithm to solve a variety of test problems with minimal variation [38]. In this work, the proposed algorithms are evaluated on the robustness rather than on deviation from lower bounds.

2.3.3 Multiobjective Optimization in Flowshop Manufacturing Cell

As mentioned before, the proposed problem is an NP-hard problem. Research effort has focused on finding the approximation algorithms that can provide a near-optimal solution at a relatively minor computational expense. Most of the currently available algorithms only deal with the optimization of a single criterion measure [8]. However, considering more than a single objective compromises the advantages of the studied objectives. For instance, minimization of the makespan improves production efficiency, while minimization of the total flow time reduces the work-in-process inventory. Therefore, minimizing both of them at the same time leads to accomplishing the benefits of both measures (efficiency and cost reduction). Li et al [39] considered the FMCSP with SDSTs for total tardiness and mean total flow time minimization. Hendizadeh et al [7] and Lin et al [8] studied the proposed problem to minimize makespan and total flow time as a bi-criteria scheduling problem. The former studied the problem for the first time, and they developed a Multi-Objective Genetic Algorithm (MOGA) to optimize the total flow time and makespan simultaneously. They compared their results to the lower bounded proposed by Schaller [40] for makespan

criterion. The latter developed a two-level multi-star simulated annealing (TLMSA) for the same problem to minimize makespan and total flow time (or total tardiness). As a result of the limited work done on the proposed problem, the author of this study was motivated to develop two multiobjective algorithms to generate approximated Pareto fronts for the proposed problem. The convergence and the diversity of the generated Pareto fronts are evaluated based on QI measures rather than the deviation from the lower bound.

2.3.4 Heuristics and Metaheuristics Approaches

Many combinatorial optimization problems are too complex to allow exact solution methods to give results in a reasonable amount of time [41]. An alternative is the use of heuristics that sacrifice a degree of optimality in return for near-optimal but acceptable solution time. Since many of the scheduling problems are a subset of combinatorial optimization problems, there is a great opportunity to exploit the power of the heuristics and metaheuristics for industrial engineering applications. The studied problem has been strongly shown to be an NP-hard problem in a strong sense. Considering SDSTs, several heuristic approaches have been proposed to find near-optimal solutions for larger instances in a reasonable computational effort [11]. Further, constructive heuristics are proposed to determine a job sequence for all part families as well as a family sequence. The most well-known heuristics are CMD [9] and NEH [10]. A branch and bound procedure for scheduling a FMCSP with SDSTs is presented by Schaller [42].

Metaheuristics have proven to be highly useful for approximately solving difficult optimization problems in practice [43]. Metaheuristics are a set of concepts that can be used

to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem [44]. Metaheuristics are characterized as high level strategies for exploring search spaces by using different methods. It is now interesting to ponder the question: What kind of metaheuristics can be implemented to solve a particular problem? The choice of the appropriate metaheuristic would be based on the complexity of the problem (NP-completeness, size and structure of the input instances). The guidelines for solving a given optimization problem are presented by Talbi [45]. The general procedure for solving an optimization problem using metaheuristics is as follows:

- Step 1: Define, formulate and model the problem
- Step 2: Speculate metaheuristic algorithms based on the complexity of the problem, and the requirements of the application (search time, quality of solution, robustness)
  - Single-solution based metaheuristics (TS, SA, VNS, etc.)
  - Population-based solution metaheuristics (GA, PSO, ACO, etc.)
- Step 3: Design a metaheuristic algorithm starts with defining the following:
  - Solution representation
  - Objective function, and constraints
- Step 4: Implement the proposed metaheuristic algorithm
  - From scratch or no use

  - Code reuse
  - Design and code reuse

- Step 5: Select the parametric levels of the algorithm
  - Off-line (e.g. Design of Experiment)
  - On-line (dynamic, adaptive)

- Step 6: Evaluate the performance of the developed metaheuristic
  - Experimental design (goal of the experiments, test instances, and factors)
  - Measurements (quality of solutions, computational effort, robustness)
  - Reporting (using box plots, and interactions)

The designing procedure is started by defining, and formulating the problem. Mathematical models aid in understanding the problem and its constraints. In the next step, the selection of the appropriate metaheuristic depends on the complexity of the problem and the problem size. In the design stage, the solution representation, objective function, and constraints should be considered carefully. Moreover, the implementation stage might be based on a new approach or implementing and improving an existing approach. In stage five, tuning of the parameters of the developed metaheuristic could be done as an off-line method based on the Design of Experiments (DOE). Alternatively, an on-line approach can also be used, which is based on the dynamic technique where the parametric levels are changed during the evolution of the algorithm. The last and perhaps the most important step in this procedure is the performance evaluation; the research reported in the literature

evaluates the quality of the solution based on the Relative Percentage Deviation (RPD) from the lower bounds. According to Talbi approach [45], the analysis of metaheuristics should be done in three steps: 1) experimental design; 2) measurement that includes: Quality of solutions, computational effort, and robustness; 3) reporting and interpretation of the results using box plots, and interactions. He gives more details for the design and implementation of efficient metaheuristics to solve complex optimization problems. The class of metaheuristic algorithms includes -but is not restricted to - simulated annealing, tabu search, evolutionary algorithms like genetic algorithms and evolution strategies, ant colony optimization, particle swarm optimization among others listed as examples of classical metaheuristics. These methods include any procedure for problem solving that employs a strategy for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures [43].

2.3.5 Hybridization of Metaheuristics

Over the last years, interest on hybrid metaheuristics has risen considerably in the field of optimization [46]. Hybridization refers to the class of optimization algorithms in which different techniques are employed together. The best results found for many real-life or classical optimization problems are obtained by hybrid algorithms [46]. Combinations of algorithms such as metaheuristics, mathematical programming, constraint programming and machine learning techniques have provided very powerful search algorithms. A discussion on taxonomies and overviews on hybridization [47] report that there are four different types of combinations:

- Combining metaheuristics with (complementary) metaheuristics .
- Combining metaheuristics with exact methods from mathematical programming approaches which are mostly used in operations research
- Combining metaheuristics with constraint programming approaches developed in the artificial intelligence community.
- Combining metaheuristics with machine learning and data mining techniques.

In this study, two metaheuristics based on Particle Swarm Optimization (PSO) which is a population-based metaheuristic, and Simulated Annealing (SA) which is a single solution based metaheuristic are combined into two different strategies. First, PSO is used to generate the initial solution of the SA algorithm. Second, SA is combined with PSO algorithm as a local search engine. The fundamental concept of the cooperation between PSO and SA is due to the following reasons: 1) The ability of PSO to locate high performance regions of vast search spaces quickly; 2) The simplicity of SA algorithm; and 3) The successful implementation of SA for solving scheduling problems. Therefore, PSO can be applied to locate promising regions. Then, it is highly recommended to apply a single solution based algorithm (such as SA) which is a powerful optimization method in terms of exploitation [46] to be combined with PSO as a local search. These two combinations of PSO and SA will be discussed in Chapter 4.

2.3.6 Intensification and Diversification

In designing metaheuristics, two conflicting criteria must be taken into account: exploration of the search space (diversification) and exploitation of the best solutions found (intensification) [45]. Promising regions are determined by the obtained good solutions. On one hand, in intensification, the promising regions are explored more thoroughly in the hope of finding better solutions. On the other hand, in diversification non-explored regions have been visited to be sure that all the regions of the search space are explored. Therefore, the proposed algorithm should compromise and balance between diversification and intensification criteria. Moreover, hybridization is implemented to balance between these criteria and to manage the cooperation between the operation of the search among the candidate solutions (populations or swarms), a diversifying agent, and the intensifying agent.
The hybrid approach implanted in this study to balance between intensification and diversification is shown in Figure 2.2. PSO is used to perform the global search, and SA algorithm is combined as a local search to improve quality of search.

2.3.7 Local Search

Local search (LS) techniques are reported to be effective for many optimization problems with arbitrary objective functions and constraints [48]. During the last decade many different types of local search algorithms for scheduling problems have been developed.

LS mainly can be divided into threshold algorithms including but not limited to SA, TS, Variable Neighborhood Search (VNS), and GA. However, these algorithms have to be carefully selected due to the highly problem-dependent effect. Parameters of an algorithm require

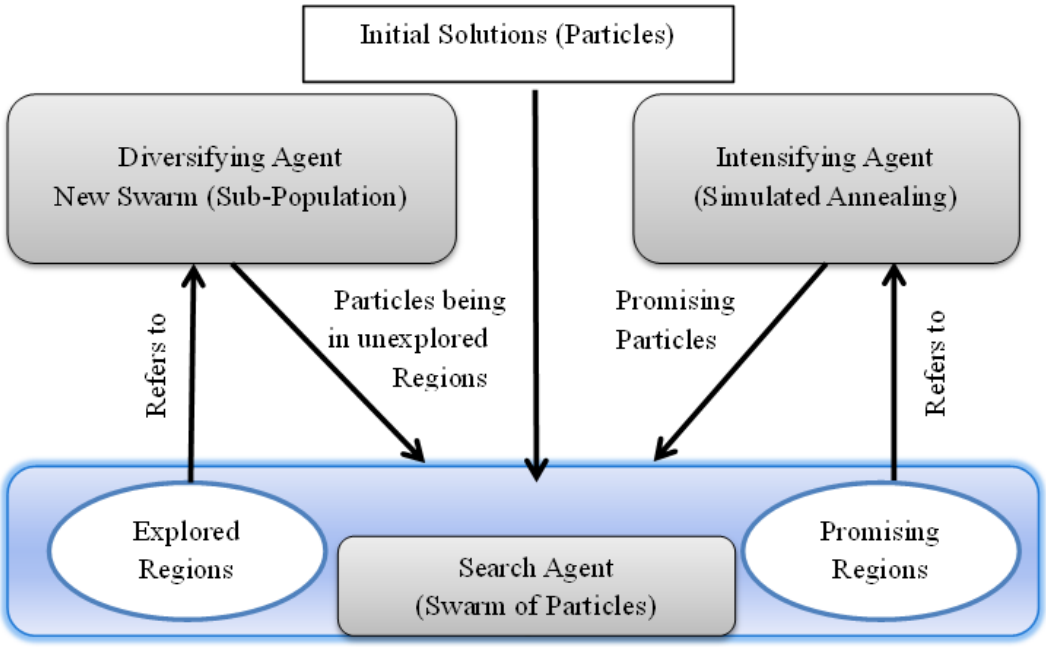

Figure 2.2: Hybridization of particle swarm optimization and simulated annealing

several tuning parameters to identify the best combination [49]. Several local search algorithms based on SA, Threshold Acceptance (TA), and GA local search in job shop scheduling problem were investigated by Aarts [50, 51]. They concluded that SA outperformed other algorithms in the sense that it finds better solutions within the same amount of running time. Furthermore, TA [52] has convergence properties which are similar to SA. The difference between SA and TA algorithms can be recognized in the acceptance condition of the new generated solution. For example, if we consider two solutions in the combinatorial optimization problem $\{i, j\}$ where i is an initial solution and j is generated from i with the cost functions, SA can be viewed as an iteration of Metropolis algorithm [53]. The solution is accepted according to a deterministic acceptance function defined as follows:

$$Pc(acceptancej) = \begin{cases} 1, & \text{if } f(j) \leq f(i) \\ e^{\frac{f(i)-f(j)}{kT}}, & \text{if } f(j) > f(i) \end{cases}$$

In TA, the deterministic acceptance function is defined as follows:

$$Pc(acceptancej) = \begin{cases} 1, & \text{if } f(j) \leq f(i) \\ 0, & \text{otherwise} \end{cases}$$

These two approaches will be described and discussed in detail in Chapters 4 and 5.

2.3.8 Parameter Tuning through Taguchi method

Selecting the best combination of the strategy/operating parameters of an algorithm is a crucial step in designing an efficient algorithm due to the significant effect on the quality of the solution [54]. Therefore, to have a better performance of the proposed algorithms it is necessary to set parameters properly. Taguchi [55, 56] proposed a novel approach to select the optimum parameters with the least possible number of experiments [57]. The required number of trials in the Taguchi method can be identified using the Orthogonal Arrays (OA) [58]. Taguchi classifies the factors into two main groups: controllable and noise (uncontrollable or costly to control) factors. Although elimination of the noise factors is impractical and often impossible, the Taguchi method seeks to minimize the effect of noise and to determine optimal level of important controllable factors based on the concept of robustness [20]. Moreover, Taguchi creates the relative significance of individual factors in

terms of their main effects on the objective function. Taguchi has created a transformation of the repetition data to another value which is the measure of variation. Signal-To-Noise Ratio (SNR) explains why this type of parameter design is called robust design [56, 58]. The term signal refers the desirable value (mean response variable) and noise refers to the undesirable value (standard deviation). Therefore, SNR indicates the amount of variation present in the response variable.

The Taguchi method has been employed to determine the best combination of the algorithm's parameters. A robust parameter combination of GA is reported [59] for flowshop scheduling problem using the Taguchi method to increase the efficiency of the proposed GA. Further, the Taguchi method combined with other algorithms to enhance the performance of the proposed algorithms. A crossover operator is conducted based on the OA and Taguchi method to select better subsequences for recombination of the selected solutions [60]. Further, Tseng et al utilized the OA-based crossover and OA-based mutation to enhance the capability of intensification [61]. Yildiz combined the properties of Taguchi method to develop a hybrid robust differential evolution method to improve the searching ability [62]. Naderi employed the Taguchi method to select the optimum parameters for the proposed GSA algorithm [20]. Yang et al developed a hybrid Taguchi-based genetic algorithm for solving multiobjective flowshop scheduling problem [63]. Seidar et al proposed a hybrid PSO for scheduling just-in-time single machine problem [64], they applied the Taguchi method to calibrate the parameters of the proposed algorithms. The Taguchi method is implemented in this study to select the best parameter combination of the SA, and study the main effect of each parameter on the performance of the algorithm (Section 4.7.2).

2.3.9 Performance Analysis and Robustness of Metaheuristics

In real-world applications, it is often desirable that a solution is not only of high performance but also of high robustness [65]. Thus, evaluating the performance of metaheuristic algorithms must be done in a rigorous manners. Thus, performance analysis aims to achieve the following goals:

- Comparing the performance of competing search metaheuristics such as genetic algorithm, simulated annealing, and particle swarm optimization
- Analyzing the algorithms parameters, and selecting the best combination of these parameters
- Improving the robustness of metaheuristics

Robustness can be defined in many ways. It includes insensitivity against small deviations in the input instances or parameters of the metaheuristic. Indeed, the lower the variability of the obtained solutions the higher the robustness of the algorithm [66]. Robustness can also be defined as a good performance over a wide range of instances of one test problem or even over a wide range of different test problems [67]. To design an efficient metaheuristic, the quality of solutions, computational effort, and robustness should be considered. Beyer et al presented a comprehensive survey on robust optimization [68] showing different approaches to perform robust optimization in practice including the methods of mathematical programming, deterministic nonlinear optimization, and direct search methods such as stochastic approximation and evolutionary computation. Samarghandi et al reported that the most

important feature of DOE is the ability to study the interaction effects between the considered factors [69]. They developed a DOE model used to determine the best combination of the sequencing and timetabling methods for no-wait job shop scheduling problem. Furthermore, the author presented a systematic approach to evaluate the robustness of two metaheuristics based on GA and PSO for FMCSP with SDSTs [38].

2.3.10 Multi-objective Performance Measures

Unlike the single objective optimization, there is a set of optimal solution called Pareto set or Pareto Front. The evaluation of the performance of the proposed algorithms of approximated Pareto sets differs from the single objective optimization problem. In multi-objective optimization, the proposed algorithms are evaluated based on a number of quality indicators. Moreover, to study the performance of a multi-objective optimization algorithm, two aspects are examined: effectiveness or the quality of outcome, and the efficiency or the required amount of computational efforts to generate such an outcome. Several performance measures have been defined in the literature to evaluate the effectiveness of algorithms. Addressing the efficiency of an algorithm often includes considering a fixed number of function evaluations or a certain amount of computational time. Moreover, these measures are implemented to evaluate the convergence of the generated Pareto fronts for a bi-objective problem. Interested readers may refer to [70] for a review of performance metrics in multi-objective optimization.

2.3.11 Summary

In summary, there has been limited work done on optimizing the completion time of a FMCSP with SDSTs especially for minimizing TFT, and minimizing both TFT and MS at the same time. Metaheuristics have proven to be highly useful for approximately solving difficult optimization problems in practice, but more metaheuristic algorithms are needed to cope with the proposed problem. Designing an efficient metaheuristic algorithm requires considering several aspect such as the complexity of the problem, selecting the appropriate combination of the parameters of the algorithm, and analyzing the effectiveness of the solutions. In this work, pure metaheuristics, and hybrid algorithms are proposed to minimize the TFT, and considering TFT and MS as a bi-criteria scheduling problem. Moreover, the Taguchi method is used to identify the best combination of the algorithmic parameters. Hybridizing of metaheuristics is proposed to balance between the diversification and intensification, and improve the quality of the solutions. The analysis of the effectiveness of the proposed algorithm is performed based on the robustness to identify the most robust algorithm that is able to solve a wide range of test problems with minimal variations. Moreover, quality indicator measures are used to evaluate the convergence of the approximated Pareto fronts.

Chapter 3

A Mathematical Model to Solve a Flowshop Manufacturing Cell with Family Sequence Dependent Setup

3.1 Background

Mathematical models are a class of exact methods that are used to find optimal solutions. In addition they are a useful means to formulate and to get a deep understanding of problems. Mathematical models have a crucial role in determining all the characteristics of the problem in terms of the objective function and constraints. Mathematical models play a key role in finding a benchmark in practical solution approaches such as lower and upper bounds. Salmasi et al [35] reported that manufacturing systems are sophisticated and complex. Mathematical models are very useful to better understand the systems so that

better solutions can be sought [71]. This chapter presents the mathematical model of the FMCSPs with SDSTs to minimize the TFT. First, the problem's definition, assumptions and constraints are presented in Section 3.2. Then, mathematical model of the proposed problem is discussed in Section 3.3. In section 3.4, the specification of the test problems is presented. Section 3.5 presents the algorithm for calculating TFT and an illustrative example.

3.2 Problem Formulation

In FMCSP, there are N_0 jobs which are grouped according to their similarity and production requirements. Therefore, there are F part families $\{1, 2, , F\}$ to be processed in a cell that has m machines $\{M_1, M_2, , M_m\}$. The ultimate goal is to find the best sequence of processing the part families as well as jobs within each family in order to minimize the total flow time. The assumptions considered in this problem are addressed in Section 1.3.

3.3 Mathematical Model

Salmasi et al [14] developed the first model for the flow shop sequence dependent group scheduling problem with minimization of total flow time as the criterion. The notations, parameters, and decision variables are presented below:

F: Number of families

N_f: Number of jobs in each family, $f = \{1, 2, ..., F\}$

N_0: Total number of jobs; $N_0 = \sum_{f=1}^{F} N_f$

m: Number of machines

$p_{f,j,i}$: The processing time for job j which belongs to family f on machine i

$Se_{l,f,i}$: The setup time of family f if it processed immediately after the family l on machine i

G_f: A set including the jobs belonging to the family $f \in \{1, 2, ..., F\}$

M: A large positive number

As mentioned before, the goal is to find the best sequence of a set of part families and jobs in each family to minimize the TFT. Since the problem is a SDST problem, setup time for any family to be processed on a machine is dependent on the current part family on a machine and on the part family that was processed right before it on the same machine. A binary variable is incorporated in the model to fulfil the SDST constraint. A dummy family (a reference part family) is assumed to be processed as the first family in a particular sequence. If a part family is assigned to be in the first position of the current sequence, the dummy family is used to calculate the required setup time for the part family on each machine. The job setup times are neglected or included in the job processing time. Therefore, there is no need to consider which job has to be processed in first position. Nevertheless, a binary variable is included in the model to express whether a job precedes/succeeds another job. The following decision variables are used in the model:

$$R_{lf} = \begin{cases} 1, & \text{if family f processed right after l where} f, l \in \{1, 2, ..., F\} \\ 0, & \text{otherwise} \end{cases}$$

$$W_{f,k,j} = \begin{cases} 1, & \text{if job j is processed after job k where} j,k \in G_f \\ 0, & \text{otherwise} \end{cases}$$

$$St_f, j, i : \quad \text{Starting time of job } j \text{ in family } f \text{on machine } i \tag{3.1}$$

$$E_f, i : \quad \text{Finish time of the last job of family } f \text{ on machine } i \tag{3.2}$$

$$C_f, j, i : \quad \text{Completion time of job } j \text{ that belongs to family } f \text{ on machine } i \tag{3.3}$$

The objective is to minimize (TFT) or sum of completion times. It is presented in equation (3.4). Each machine can handle only one part at a time, and every part can be processed by at most one machine at any given time without preemption. Completion time of a job on a machine should be greater than its completion time on the previous machine plus its processing time on that machine. This fact is supported by constraint (3.6). The difference between the completion times of two different jobs $\{j, k\}$ on a machine should be greater than the processing time of the second job k that is processed right after job j; constraints (3.7 and 3.8) are supporting this fact. The mathematical model that formulates the FMCSP with SDSTs with minimization of the total completion time (TFT) is as follows:

$$Minimize \sum_{j=1}^{N_0} C_{f,j,m} \tag{3.4}$$

Subject to:

$$C_{f,j,i} \geq C_{f,j,(i-1)} + p_{f,j,i}, \qquad \forall j, i \tag{3.5}$$

$$C_{f,j,i} \geq C_{f,k,i} + p_{f,j,i} - (1 - W_{f,k,j}) \times M, \qquad \forall f, j, k \in G_f \tag{3.6}$$

$$C_{f,k,i} \geq C_{f,j,i} + p_{f,j,i} - (W_{f,k,j}) \times M, \qquad j, k \in G_f \tag{3.7}$$

$$C_{f,j,i} \geq St_{f,j,i} + p_{f,j,i} \qquad j \in G_f, i \in \{1, 2, ..., m\} \tag{3.8}$$

$$E_{f,i} \geq C_{f,j,i}, \qquad \forall f, j, k \in G_f, i \in \{1, 2, ..., m\} \tag{3.9}$$

$$St_{f,j,i} \geq E_{l,i} + Se_{l,f,i} - (1 - R_{l,f}) \times M, \quad \forall l, f \in \{0, 1, .., F\}, f \neq l, i \in \{1, 2, .., m\} \tag{3.10}$$

$$\sum_{l=0, l \neq f}^{F} R_{l,f} = 1 \qquad \forall f, f \neq l \tag{3.11}$$

$$\sum_{f=1}^{F} R_{l,f} \leq 1 \qquad \forall f, f \neq l \tag{3.12}$$

$$\sum_{f=1}^{F} R_{0,f} = 1 \tag{3.13}$$

$$R_{l,f} + R_{l,f} \leq 1 \qquad \forall l, f > l \tag{3.14}$$

$$C_{f,j,i}, St_{f,j,i}, E_{f,i} \geq 0 \qquad j \in G_f, i \in \{1, 2, ..., m\} \tag{3.15}$$

$$W_{f,k,j}, R_{l,f} \in \{0, 1\}, \qquad C_{f,j,0}, E_{0,i} = 0 \tag{3.16}$$

Constraints (3.9 and 3.10) are incorporated into the model to ensure that every job must start and finish between starting $\{St\}$ and finishing times $\{E\}$ of the family that it belongs to. Constraint (3.11) ensures that processing on the next family always starts after processing on the current part family is done. The start time of processing a job belonging to a certain part family l on a machine i is greater than the finish time of the last family f on the machine

plus the setup of the family l (i.e. $Se_{l,f.i}$). Constraint (3.12) supports the fact that each family f should have exactly one family as the previous processed family. Constraint (3.13) is incorporated into the model to make sure that each family has at most one successor family. A reference family or family zero is considered when a family is processed as the first family in the sequence. Constraint set (3.14) is included into the model to make sure that the reference family is assigned to the first slot. Constraint set (3.15) supports the fact that each part family can be processed either before another part family or after that part family. Lastly, decision variables are given in constraints (3.16).

3.4 Test problem Specifications

In this study, two different test problems are used to evaluate the effectiveness of the developed algorithms in Chapter 4, and 5: First, Salmasis test problems [14]. These test problems are used to evaluate the proposed algorithm to minimize the TFT. Second, Schallers test problems [28] are used to evaluate the algorithms proposed for the bi-objective problem. The specifications of these test problems are presented in the following subsections.

3.4.1 Schaller test problems

The first set is developed by Schaller [28]. These test problems have been most widely used for FMCSPs [7], [8], [15], [29], [31], [37], [50] to test the proposed algorithms. These test problems examined study for the bi-objective optimization problem which is addressed in Chapter 5. Schaller generated the processing times of each machine randomly using a uniform distribution in the range of $[1, 10]$. To generate examples with different levels of

difficulty, three different classes of family setup times were generated as random integers from the following uniform distributions:

- Small Setups (SSU): U [1,20]
- Medium Setups (MSU): U [1,50]
- Large Setups (LSU): U [1,100]

The problem difficulty depends on whether there was a balance between average processing times and the average family setup times. This distribution implied that the ratio of mean family setup time to mean part processing time was approximately 2:1 (10.5:5.5), 5:1 (25.5:5.5), and 10:1 (50.5:5.5). In addition, the number of families and number of machines was between 3 and 10, and the range of number of jobs in a family was between 1 and 10. Schaller generated 30 sub-problem sets for each combination of problem parameters to consider a wide range of problem instances, Since 30 problem instances were generated for each sub-problem set, the total of test problems involved is 900.

3.4.2 Salmasi's test problems

Salmasi et al [14] reported two reasons for developing these test problems: 1) the test problems are more similar to real world industrial problems, and 2) these test problems are considered a benchmark to evaluate the performance of proposed algorithms with the total flow time minimization. Additionally, these test problems have been widely used in FMCSPs with SDSTs. These instances have been used in [14, 20, 34, 35, 72, 73] to test the effectiveness of several metaheuristics, and to solve problems to minimize TFT as a single

performance measure (Chapter 4). In this thesis, two-machine and six-machine test problems are considered . Specification of the test problems is briefly summarized as follows (see Figure 3.1):

- The processing time of each job on each machine is a random integer from a uniform discrete distribution [1, 20]
- The number of families is a random integer from uniform discrete distribution [2,5], [6,10], and [11,16] for small, medium, and large size problems, respectively
- The number of jobs in a family is a random integer from uniform discrete distribution [2,4], [5,7], and [8,10] for small, medium, and large size problems, respectively (for each classification of family size)
- The setup times of the families on each machine are classified into three different levels based on the ratio of setup times for all machine pairs and the number of machines in the cell as shown in Figure 3.1 for the two-machine instances and Table 3.2 for the six-machine instances. Moreover, Salmasi stated that the quality of solutions identified by metaheuristics strongly depends on the ratio of setup times of part families on successive machines, and it should be less than 0.8. Therefore, if setup times are generated based on the U[1, 50], and U[16, 67], then the average ratio of setup times is equal to $0.607([(1+50)/2]/[(17+67)]/2 = 0.607)$, which is less than 0.8 [34].

Thus, the problems can be classified into 27 different classes based on the number of families, maximum number of jobs in a family, and the ratio of setup times. Moreover,

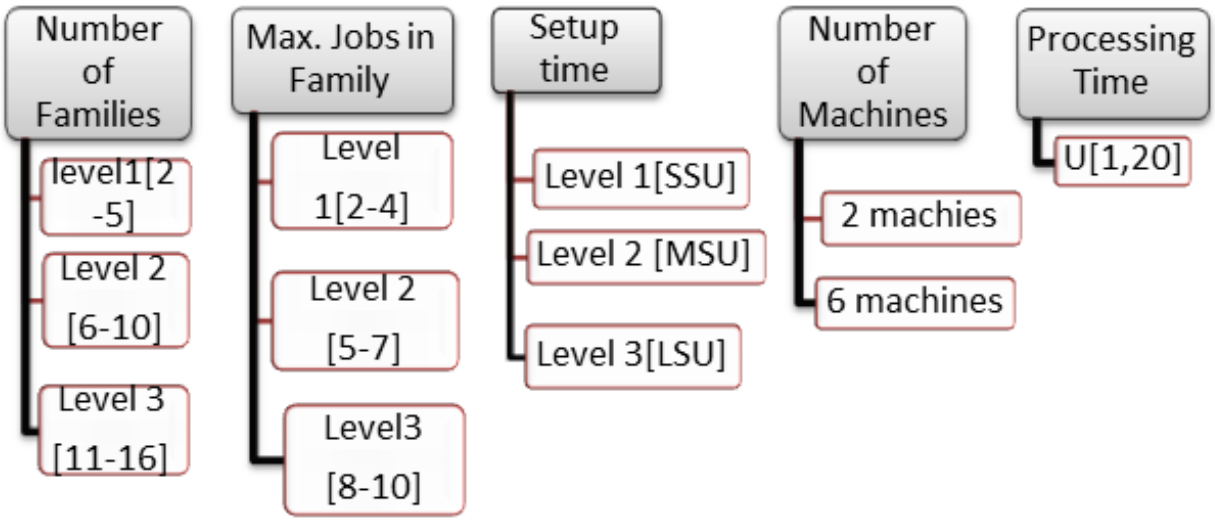

Figure 3.1: Test Problem specifications

Table 3.1: The setup time of each machine in two-machine instances

	No. of Families.	Max Job in Family	Total Jobs
M_1	U[1, 50]	U[1, 50]	U[17, 67]
M_2	U[17, 67]	U[1, 50]	U[1, 50]

Table 3.2: The setup time of each machine in six-machine instances

	No. of Families.	Max Job in Family	Total Jobs
M_1	U[1, 50]	U[1, 50]	U[300, 350]
M_2	U[17, 67]	U[1, 50]	U[170, 220]
M_3	U[45, 95]	U[1, 50]	U[92, 142]
M_4	U[92, 142]	U[1, 50]	U[45, 95]
M_5	U[170, 220]	U[1, 50]	U[17, 67]
M_6	U[300, 350]	U[1, 50]	U[1, 50]

two replicates are generated; therefore, the number of test problems for two-machine, and six-machine problems are as follows:

1. two-machine problems

   There are three levels of the number of families, three levels of the maximum number of jobs in a family, and the three levels of setup ratio. Since there are two replicas, the total number of the test problems is fifty four instances.

2. six-machine problems

   There are three levels of the number of families, three levels of the maximum number of jobs in a family, and the three levels of setup ratio. Since there are two replicas, the total number of the test problems is fifty four instances.

3.5 Calculating the Total flow time of jobs

Figure 3.2 shows the calculation of completion time of a job j ($j \neq 1$) that belongs to family f in a sequence dependent setup scheduling problem. $C_{f,j,i}$ refers to the completion time of a job j that belongs to part family f to be processed on machine i.

The completion time is given by:

$$C_{f,j,i} = max\{C_{f,j,(i-1)}, C_{f,(j-1),i}\} \qquad j \neq 1 \tag{3.17}$$

If the job is assigned to be in the first position of the job sequence, the completion time will

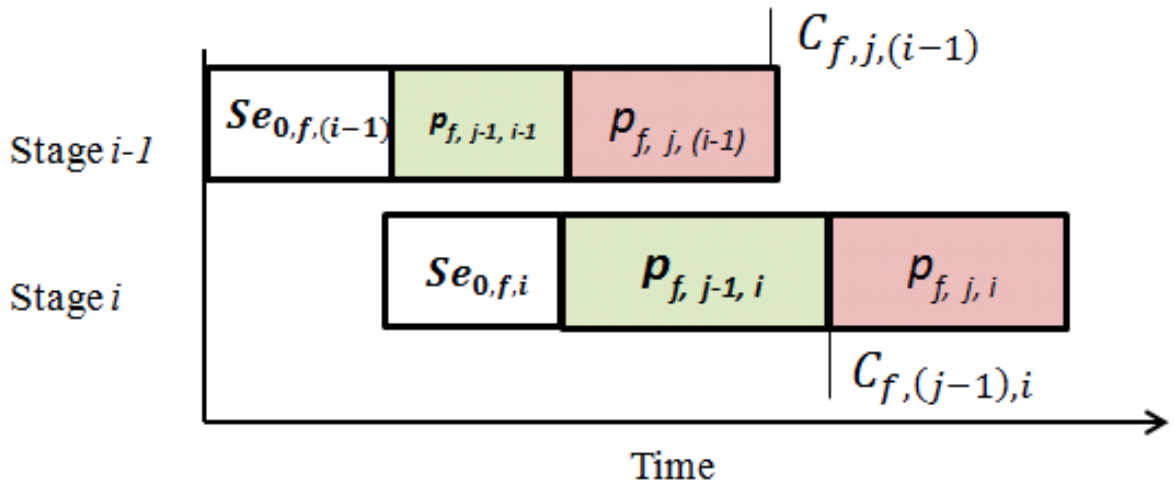

Figure 3.2: Completion time of a job in the case of sequence-dependent setup problem

be as follows:

$$C_{f,j,i} = max\{C_{f,j,(i-1)}, C_{f,(j-1),i} + Se_{l,f,i} + p_{f,j,i}\} \qquad j = 1 \tag{3.18}$$

where

$$C_{f,j,0} = 0 \qquad for\{j = 1, 2, ..., N_f\}, and \qquad \{f = 1, 2, ..., F\} \tag{3.19}$$

$$C_{0,0,i} = 0 \qquad \{i = 1, 2, ..., m\} \tag{3.20}$$

Completion time or flow time $F_{f,j,i}$ of a job j in a family f on a machine i can be calculated using the start time Stf, j, i, and the processing time $p_{f,j,i}$ according to following steps:

- Step 1: Starting time of first job in the first family on first machine is equal to the

setup $Se_{l,f,i}$ of this family when it takes first position in the sequence

$St_{1,1,1} = Se_{1,1,1}$

- Step 2: Starting time of the first job of the first family on machine $i \in \{1, 2, ..., m\}$

  $St_{1,1,i} = \max \{Se_{0,1,1} ; St_{1,1,i} + p_{1,1,i-1} \}$

- Step 3: The starting of job $j \in \{2, 3, ..., N_f\}$ of the first family on the first machine

  $St_{1,j,1} = \{St_{1,j-1,1} + p_{1,j-1,1}\}$

- Step 4: The starting job $j \in \{2, 3, ..., N_f\}$ of the first family on machine $i \in \{1, 2, ..., m\}$

  $St_{1,j,i} = \max \{St_{1,j-1,i} + p_{1,j-1.i} \ ; \ St_{1,j,i-1} + p_{1,j,i-1}\}$

- Step 5: Starting the first job in family $f \in \{2, 3, ..., F\}$ on the first machine

  $St_{f,1,1} = \{St_{(f-1),N_f,1} + p_{(f-1),N_{f-1},1} + Se_{f,(f-1),1}\}$

- Step 6: Starting time of job $j \in \{2, 3, ..., N_f\}$ in family $f \in \{2, 3, ..., F\}$ on the first machine

  $St_{f,j,1} = \{St_{f,j-1,1} + p_{f,j-1,1}\}$

- Step 7: Starting time of the first job in family $f \in \{2, 3, ..., F\}$ on machine $i \in \{1, 2, ..., m\}$

  $St_{f,1,i} = \max \{St_{(f-1),N_{f-1},i} + p_{(f-1),N_{f-1},i} + Se_{f,(f-1),1} \ ; \ Se_{f,f-1,1} + p_{f,1,i-1}\}$

- Step 8: Starting job $j \in \{2, 3, ..., N_f\}$ in family $f \in \{2, 3, ..., F\}$ on machine $i \in \{1, 2, ..., m\}$

  $St_{f,j,i} = \{St_{f,j-1,i} + p_{f,j-1,i}; \ St_{f,j,i-1} + p_{f,j,i-1}\}$

The completion time of a job is then obtained by adding the starting time of that job on the last machine to its processing time, computed by using the recursive relations described above. The TFT is then the sum of flow times of all jobs.

3.5.1 Illustrative example

For the sake of clarification, the following example is considered to illustrate the calculation method for TFT in a manufacturing cell that consists of four machines dedicated to produce 3 part families. Processing times for each job over each machine are tabulated in Table 3.3. Setup times of the part families over machines are shown in Table 3.4.

Table 3.3: Processing time of jobs over machines in a cell ($p_{f,j,i}$)

		$j=1$	$j=2$	$j=3$	$j=4$
$f=1$	$i=1$	3	10		
	$i=2$	3	10		
	$i=3$	3	8		
	$i=4$	7	3		
$f=2$	$i=1$	1	5	4	
	$i=2$	3	3	8	
	$i=3$	6	8	3	
	$i=4$	4	6	1	
$f=3$	$i=1$	3	5	5	1
	$i=2$	1	5	4	9
	$i=3$	6	3	9	6
	$i=4$	9	4	3	1

The TFT will be calculated for the case of the following sequences:

Sequence I: $F_1 - F_2 - F_3(J_1 - J_2)(J_1 - J_2 - J_3)(J_2 - J_3 - J_1 - J_4)$

Sequence II: $F_3 - F_2 - F_1(J_4 - J_1 - J_2 - J_3)(J_3 - J_2 - J_1)(J_2 - J_1)$

Sequence III: $F_3 - F_1 - F_2(J_1 - J_4 - J_2 - J_3)(J_2 - J_1)(J_1 - J_2 - J_3)$

TFT and the MS of these scenarios are calculated, and presented in Figures 3.3, 3.4, and

3.5. The TFT for each sequence can be calculated as follows:

Sequence I: $TFT = \sum_{j=1}^{N_0} C_{f,j,m} = 31 + 41 + 75 + 85 + 86 + 104 + 113 + 117 + 126 = 778$

Sequence II: $TFT = \sum_{j=1}^{N_0} C_{f,j,m} = 36 + 50 + 54 + 57 + 73 + 96 + 100 + 122 + 129 = 717$

Sequence III: $TFT = \sum_{j=1}^{N_0} C_{f,j,m} = 41 + 42 + 48 + 56 + 86 + 93 + 114 + 124 + 125 = 729$

This example shows that there is an inverse proportion between TFT and MS. For example, in sequence I, TFT was 778, and makespan was 126; whereas in sequence II, TFT became 717, and MS was 129. Among these different sequences, the second sequence is considered best sequence in the comparison with sequence II, and sequence III with TFT of 717, and MS of 129 for the total flow time and makespan. Note that, these scenarios represent only a small fraction of the total number of combinations. Total number of combinations = $(3! * 2! * 4! * 3! =)$ 1827 combinations. Finding the best sequence out of all these combinations is the focus of this thesis. Note that a higher number of families and jobs leads to a higher number of combinations to check. Therefore, it is imperative to use a kind of optimization algorithm to find the best possible sequence.

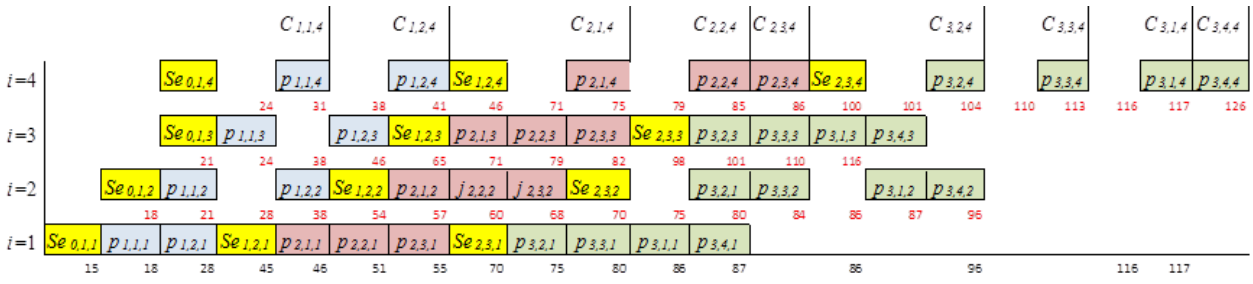

Figure 3.3: Gantt chart for sequence I

Table 3.4: Setup times of part families over machines ($Se_{l,f,i}$)

		$f=1$	$f=2$	$f=3$
$l=0$		15	18	19
$l=1$	$i=1$	NA	17	17
$l=2$		3	NA	14
$l=3$		19	15	NA
$l=0$		3	9	11
$l=1$	$i=2$	NA	16	18
$l=2$		5	NA	20
$l=3$		12	2	NA
$l=0$		9	8	10
$l=1$	$i=3$	NA	19	16
$l=2$		10	NA	16
$l=3$		16	16	NA
$l=0$		1	2	16
$l=1$	$i=4$	NA	5	4
$l=2$		19	NA	7
$l=3$		4	14	NA

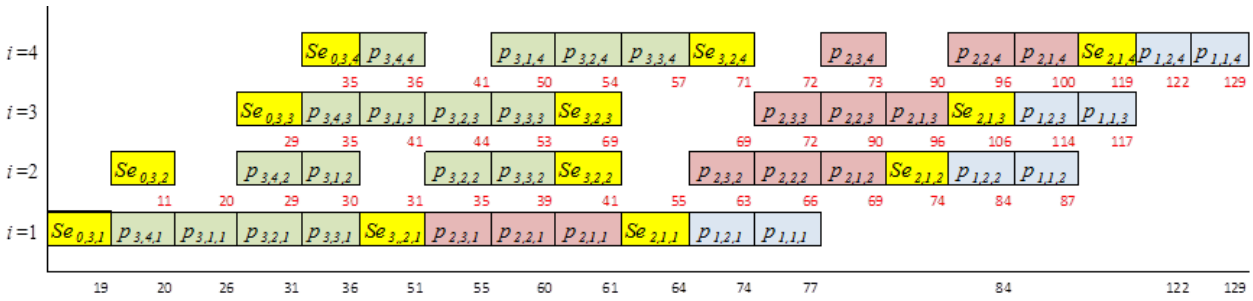

Figure 3.4: Gantt chart for sequence II

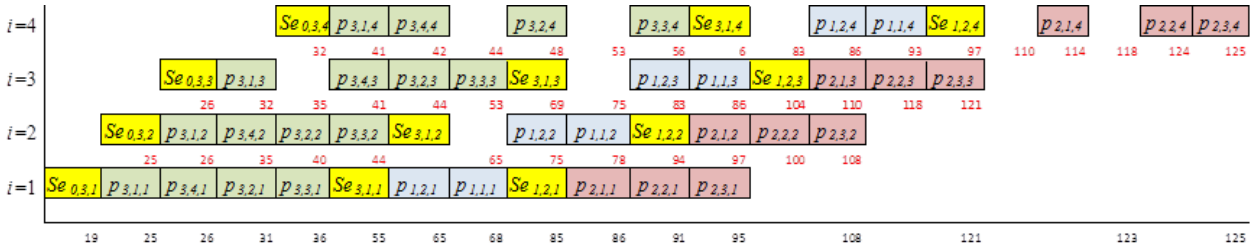

Figure 3.5: Gantt chart for sequence III

Chapter 4

Metaheuristics for scheduling a flowshop manufacturing cell to minimize the total flow time

4.1 Introduction

As mentioned in Sections 2.2, and 2.4, the research problem is NP-hard; the computation time required to find the optimal sequence of the families as well as the jobs in each family could be extremely high [27]. The most promising approach for solving combinatorial optimization problems is metaheuristics [74]. This Chapter addresses several metaheuristic algorithms to solve the FMCSPs with SDSTs for minimizing TFT in a reasonable time. First, a pure metaheuristics approach based on GA, and PSO is proposed. A comparison of performance of the proposed GA and PSO based on the robustness of metaheuristics

is presented by Ibrahem et al [38]. Second, two hybrid metaheuristics based on PSO and SA (named as SA, and PSO-SA) are proposed. The rest of the chapter is organized as follows: First, problem definition and solution representation are presented in Section 4.2. Metaheuristics approaches based on GA and PSO are presented in Section 4.3. Hybrid metaheuristics based on SA and PSO are discussed in Section 4.6, 4.7, and 4.8. Performance analysis and discussion of the proposed algorithms are presented in Section 4.9. Finally, summary and conclusion remarks are reported in Section 4.10.

4.2 Problem definition and Solution representation

This chapter addresses scheduling a FMCSP with SDSTs to minimize TFT. Using the triplet notation [5], the problem can be notated as: $F_m \setminus fmls, Se_{lki}, prum \setminus \sum_{j=1}^{N} C_j$. The solution of the studied problem is achieved in two phases or levels:

1. Sequencing of part families

2. Sequencing of jobs within each part family

The solution representation consists of $F + 1$ segments; the first segment F represents the sequence of part families on each machine, the other segments correspond to the sequence of jobs within each part family [14,15,30,34,75,76]. The sequence of part families and the parts within each part family are the same on all machines (permutation flowshop). As shown in Figure 4.1, for a feasible schedule, a solution π of FMCSP takes the following structure:

$$\Pi = \left\{\Pi_{[1]}, \Pi_{[2]}, ..., \Pi_{[f]}, \Pi_{[f+1]}, ..., \Pi_{[F]}\right\}$$

where

$$\Pi_{[f]} = \left\{\Pi_{[f][1]}, \Pi_{[f][2]}, ..., \Pi_{[f][j]}, ..., \Pi_{[F][N_f]}\right\}$$

is the sequence of the jobs in each part family. Figure 4.1 shows solution structure of part families as well as jobs in each part family:

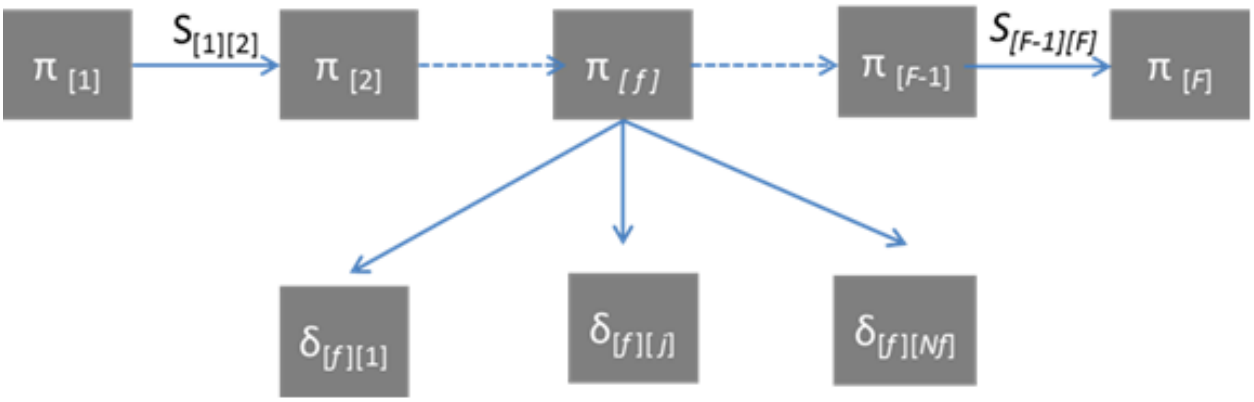

Figure 4.1: The solution structure

The algorithm used for calculation of TFT with a numerical example has been discussed in Section 3.5. Further, the major assumptions made in this study are mentioned in Section 1.3. The ultimate goal is to find an efficient algorithm to minimize the TFT.

4.3 Metaheuristic Approach

In this chapter, metaheuristics are developed for FMCSP with SDSTs to minimize TFT. Metaheuristics based on GA, PSO, and SA are developed to solve the problem heuristically First, pure metaheuristics based on GA and PSO (population solution-base) are proposed. Second, SA (single solution-base) is proposed. Moreover, a hybridization of PSO with SA as

a local search is proposed to improve the search ability of the PSO algorithm. The algorithms will be discussed in the following sections.

4.4 Proposed Genetic Algorithm (GA)

Genetic Algorithm (GA) has successfully been applied to the scheduling problems. It consists of individuals, which composes a population of solutions. These individuals are evolved by mutation and reproduction processes. The best fitted solutions of the population survive while the worse fitted will be replaced [29]. The basic steps of the proposed GA are shown in Algorithm 4.1

4.4.1 Solution representation

The main goal is to find the sequence of the part families as well as the sequence of jobs in each part family. There are two levels of sequences: part family, and job sequence. For instance, consider the following problem with four part families with 3, 4, 2, and 3 jobs in each family, respectively. Jobs are assigned to be processed on a cell that has three machines. The pseudo matrix for a sequence of part families as well as jobs within these part families is as follows: $\{ F_{[4]}, F_{[3]}, F_{[1]}, F_{[2]}\}$ The sequence of jobs in each part family may be of the form: $\{ F_{[4]} : [J_{42} - J_{41} - J_{43}], F_{[3]} : [J_{31} - J_{32}], F_{[1]} : [J_{13} - J_{11} - J_{12}], F_{[2]} : [J_{22} - J_{24} - J_{23} - J_{21}]\}$

The solution is presented as: X = $\begin{pmatrix} J_{42} & J_{41} & J_{43} & \\ J_{31} & J_{32} & & \\ J_{13} & J_{11} & J_{12} & \\ J_{22} & J_{24} & J_{23} & J_{21} \\ F_4 & F_3 & F_1 & F_2 \end{pmatrix}$

This matrix would be transformed to integer numbers, X = $\begin{pmatrix} 2 & 1 & 3 & \\ 1 & 2 & & \\ 3 & 2 & 1 & \\ 2 & 4 & 3 & 1 \\ 4 & 3 & 1 & 2 \end{pmatrix}$

The first four rows are used to represent sequence of jobs within part families: $F_{[1]}$, $F_{[2]}$, $F_{[3]}$, $F_{[4]}$, whereas the last row represents sequence of the part family.

4.4.2 Selection

The proposed algorithm starts with an initial solution that is generated randomly from the population with size $|S|$. After that, the fitness function is estimated based on the objective function which is used to minimize TFT. The fitness function is equal to the reciprocal of TFT. For each solution Π the normal probability can be determined by the following formula [20]:

$$\frac{\frac{1}{TFT(\Pi)}}{\sum_{X=1}^{|S|} \frac{1}{TFT(X)}} \tag{4.1}$$

The solutions that display the higher probability values based on equation 4.1 will be

more suitable to be selected for reproduction. Based on the fitness values, two solutions are randomly picked for the reproduction process using the crossover and mutation operators.

Algorithm 4.1 The steps of the proposed Genetic Algorithm

Require: : Set the algorithmic parameters $Pop.Size, Max.CPUtime, MR, CR$

- Step 1: Initialize the population by generating initial solutions
- Step 2: Assess each solution in the population based on TFT and assign each of them a fitness value
- Step 3: Select individuals for reproduction that show high probability based on equation (4.1)
- Step 4: Recombine individuals generating new ones
- Step 5: Mutate the new individuals
- Step 6: If the stopping criterion is met, STOP, otherwise, replace old individuals with the new ones, and return to step 3

4.4.3 Crossover Operator

The crossover operator is used to find better solutions by recombining good genes from different parent chromosomes. The crossover operator combines the selected solutions to build two new ones. For instance, a random solution of the aforementioned problem can be represented in two segments: the sequence of the families, and the sequence of jobs in each family. Crossover is performed similar to the crossover presented by [7, 29]. A new offspring is constructed from two parents as follows: A segment of parent one is randomly selected and copied in the same position as the new offspring. The rest of the first part of the offspring is completed or filled from the second parent (see Figure 4.2).

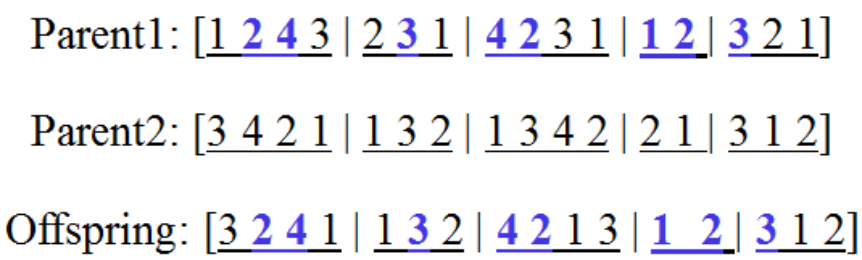

Figure 4.2: The crossover operator

4.4.4 Mutation Operator

The mutation operator generates offspring from a single parent. First, two random positions in each part of a parent are chosen and then their positions are swapped. The offspring is compared with its corresponding parent. This is different from the traditional GA, where either computation time or the number of generations is selected as the termination criterion [77]. The parameters of GA that are used as follows: initial population size ($Pop.Size$) is 30, the maximum CPU ($Max - CPU$) time is 60 sec, mutation rate (MR) is 2, and crossover rate (CR) is 0.9.

4.4.5 Results of the proposed GA algorithm

The proposed GA algorithm is coded using C++ language and run on a PC with an Intel core I7 (2.93 GHz) CPU and 4.0 GB memory. The performance of the proposed algorithm is tested using test problems which are proposed by Salmasi (presented in Section 3.4). The computational results of minimum, average, and maximum TFT are summarised in Tables 4.1, 4.2, and 4.3, for two-machine test problems and Tables 4.4, 4.5, and 4.6 for the six-machine problems. The comparison (in terms of robustness) of GAs performance with the

other proposed PSO algorithms is presented in Section 4.5. In this section, the performance of proposed GA algorithm is compared with the best available existing algorithms in the literature. Results show that GA performs as the best available algorithm (ACO) for solving the small size problems. However, as shown in Figure 4.3, a small deviation from the best is noticed at the large size problem.

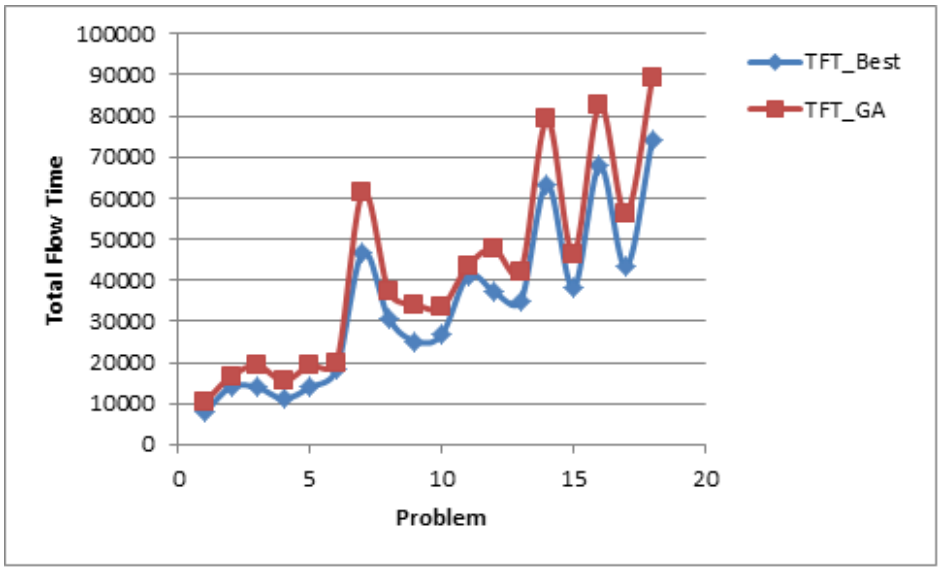

Figure 4.3: Performance of proposed GA vs. ACO for large test problems

Table 4.1: Results of small test problems for two machine problems obtained by GA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**		
				Min.	Ave.	Max
1	4	4	13	2084	2386	2581
2	3	4	10	1451	1455.667	1458
3	3	4	8	812	976	1058
4	2	3	5	454	475.3333	486
5	5	4	16	2963	3427	3659
6	3	4	8	1014	1188.667	1276
7	5	7	28	1691	1965.667	2103
8	4	6	22	4439	5033	5330
9	3	7	17	2860	2930.667	2966
10	2	7	13	1424	1588.667	1671
11	5	7	31	8649	9419	9804
12	4	7	25	5727	5759.667	5776
13	5	10	31	8680	9912	10528
14	2	8	15	2205	2205	2205
15	5	9	35	8687	9207	9467
16	5	10	40	11247	11655.67	11860
17	4	9	31	6878	7222.667	7395
18	4	10	28	5394	7203.333	8108

Table 4.2: Results of medium test problems for two machine problems obtained by GA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**		
				Min.	Ave.	Max
19	6	4	17	3373	3608.667	3997
20	8	4	25	6296	7592	7916
21	9	3	22	4913	5404.667	6087
22	9	4	27	7452	7801.667	8273
23	10	4	33	12432	13869.33	15412
24	6	4	16	3099	3594.667	4004
25	9	7	60	31122	33058	34989
26	10	7	62	29887	31661.67	34482
27	6	7	33	8834	9620	10984
28	10	7	52	20331	22350.33	24033
29	6	7	32	8702	9969.667	10736
30	10	7	49	21886	23035	23699
31	8	9	41	15570	16615.33	17351
32	8	10	51	20247	22760.75	24540
33	10	10	71	26929	29080.33	31904
34	6	8	29	6093	6579.5	7028
35	9	9	58	10058	11130.25	12850
36	6	10	31	8432	9888	11824

Table 4.3: Results of large test problems for two machine problems obtained by GA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**		
				Min.	Ave.	Max
37	11	4	29	10376	10618	10706
38	13	4	40	16627	17502.33	18330
39	16	4	46	19294	19304	19316
40	13	4	40	15493	15557.5	15622
41	14	4	40	19334	19708.4	19837
42	15	4	46	19767	24457.4	27607
43	16	7	79	61509	62998	63674
44	13	7	63	37090	38212	39984
45	16	7	66	34111	35836.33	39251
46	12	7	65	33505	33915	34469
47	15	7	76	43612	44646.33	46543
48	14	7	69	47766	49204.25	51211
49	11	10	75	41965	43360.67	45646
50	15	10	99	79109	83652.33	86017
51	12	10	83	46462	48926	51323
52	16	10	106	82747	84275.67	86003
53	15	10	79	56230	59652.67	62345
54	16	10	108	89051	93669.5	97612

Table 4.4: Results of small test problems for six machine problems obtained by GA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**		
				Min.	Ave.	Max
1	5	3	11	10269	10981.4	11719
2	3	4	9	5609	6428.2	7107
3	4	3	10	1816	1980.4	2148
4	2	4	7	892	972.8	1089
5	4	4	15	13073	13502.6	13936
6	2	4	8	4812	4874.2	4913
7	5	7	29	30444	33255.6	34403
8	4	6	21	18352	19303	20059
9	2	7	11	1672	1935	2105
10	5	7	25	6418	7062.8	7981
11	4	6	15	12034	14204	16534
12	5	7	20	20244	23479.75	26539
13	2	10	13	6267	6768.6	8530
14	3	10	29	20913	21712	22494
15	4	10	28	8584	9215	10027
16	4	9	21	5854	6197.6	6444
17	4	9	28	26010	27301.6	28669
18	3	9	21	15334	16946.2	18841

Table 4.5: Results of medium test problems for six machine problems obtained by GA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**		
				Min.	Ave.	Max
19	9	4	31	51610	55401	56984
20	8	4	19	26737	29269.2	32700
21	6	4	22	6184	6874.2	7400
22	9	4	25	8686	9485.4	10059
23	8	4	24	37377	40757.6	42349
24	6	4	16	18387	20537.6	21612
25	6	7	28	32849	35801.4	42011
26	7	7	29	36904	38367.2	39871
27	8	6	33	13037	14029.6	15258
28	9	7	38	16096	16733.4	17967
29	7	7	34	43376	48906.2	53132
30	6	7	31	36465	40544.2	44297
31	9	9	50	85490	93976	108831
32	8	10	49	71955	79537.4	86463
33	7	10	46	19463	20423.8	21558
34	10	10	65	38717	41703.4	43270
35	10	10	56	112580	123227.2	135153
36	8	10	46	67329	74458.4	82492

Table 4.6: Results of large test problems for six machine problems obtained by GA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**		
				Min.	Ave.	Max
37	16	4	45	123838	129720	135152
38	12	4	32	69917	72004.8	73934
39	14	4	37	18892	19593.8	20103
40	13	4	40	20435	21137	21391
41	11	4	34	70920	73453.2	76257
42	14	4	45	119508	122893.8	127141
43	13	6	47	105030	118239	132664
44	16	7	63	175467	187036	195624
45	15	7	71	50681	51750.4	52986
46	14	7	60	38794	41860	46414
47	15	7	69	188962	197447	208509
48	16	7	78	241381	247648.4	258768
49	13	10	95	246788	260160.2	268624
50	14	10	80	213914	227866.2	240581
51	13	10	72	47856	50580	52099
52	15	10	77	58525	59803.4	61150
53	16	9	100	304645	318877.6	333040
54	15	10	117	368921	378431.2	389252

4.5 Particle Swarm Optimization (PSO) Algorithm

Particle Swarm Optimization (PSO) is a novel iterative computational evolution model that was developed by Kennedy and Eberhart [78]. An overview of important work and research directions on particle swarms as well as applications are presented by Poli et al [79]. Liu proposed an effective particle swarm optimization (PSO) based memetic algorithm (MA) for the permutation flow shop scheduling problem to minimize the makespan [80]. A novel PSO algorithm for the permutation flowshop scheduling is applied by Lian et al [81] to minimize makespan. Tseng developed a PSO algorithm for the scheduling of multiprocessor tasks in a multistage hybrid flowshop with makespan minimization [82]. Kuo proposed a hybrid PSO model named (HPSO) that combines random-key encoding scheme, individual enhancement (IE) scheme, and PSO to solve the flowshop scheduling problem to minimize makespan [83]. RameshKumar et al [84] proposed a Discrete Particle Swarm Optimization (DPSO) algorithm to solve permutation flowshop scheduling problems with the objective of minimizing the makespan . Lin et al [85] presented a hybrid algorithm combined PSO algorithm with SA technique, and multi-type individual enhancement scheme to solve the job-shop scheduling problem. Liu et al [86] proposed a hybrid PSO with estimation of distributed algorithms to solve permutation flowshop scheduling problem to minimize the makespan. Gohar and Salmasi proposed several hybrid metaheuristic algorithms based on PSO and SA to heuristically solve the flexible flow-line scheduling problem by considering constraints for the beginning and terminating times of processing the jobs. The objective again was to minimize the makespan [87].

Simply, PSO algorithm simulates birds swarm behaviour, and makes every particle in the swarm move according to its experience and the best particles experience to find a new better position. After the evolution, the best particle in the swarm is seen as the best solution for the input problem. The population of PSO is called swarm and each individual or particle which is a potential solution is known with its current position and current velocity. The new position of each individual particle is obtained by assigning a new position as well as a new velocity to the particle. Each particle gains a different position, and the value of each position is evaluated based on the value of the objective function. The main advantage of this approach is that every particle always remembers its best position in the experience. When a particle moves to another position, it must refer to its best experience and the best experience of all particles in the swarm. The best position of each particle that has been gained so far during the previous steps is called the best particle (p-best). The best position gained by all particles so far is called the global best (g-best). The new position as well as the new velocity of each particle are obtained based on the previous positions, the p-best, and the g-best. Considering an n-dimension search space, there are S particles (swarm size) cooperating to find the global optimum in the search space. In a swarm of S particles, the ith particle is associated with the position vector $\{x_{i1}, x_{i2}, ..., x_{in}\}$ and the velocity $\{v_{i1}, v_{i2}, ..., v_{in}\}$.

The p-best and g-best are updated each iteration based on generation of new swarms. Each particle uses its own search experience and the global experience by the swarm to update

the velocity and flies to a new position based on the following equations:

$$v_j(t+1) = wv_j(t) + C_1 r_1^*(P_j^t - x_j(t)) + C_2 r_2^*(G^t - x_j(t)) \tag{4.2}$$

$$x_j(t+1) = v_j(t+1) + x_j(t) \tag{4.3}$$

Where w is the inertia weight, it controls the influence of the previous velocity of particles, C_1 and C_2 are called acceleration coefficients that provide weight to the social influence. The parameters r_1 and r_2 are uniformly distributed random variables in the range between $[0, 1]$. For the tth iteration, P_i^t and G^t are the p-best (for i^{th} particle) and g-best particles respectively. The values of these parameters are updated in each iteration based on the following equations:

$$w = w^{min} + \frac{w^{max} - w^{min}}{1 + e^{\frac{-a(maxI-I)}{maxI}}} \tag{4.4}$$

$$C_1 = C_1^{min} + \frac{C_1^{max} - C_1^{min}}{1 + e^{\frac{-a(maxI-I)}{maxI}}} \tag{4.5}$$

$$C_2 = C_2^{min} + \frac{C_2^{max} - C_2^{min}}{1 + e^{\frac{-a(I)}{maxI}}} \tag{4.6}$$

The maximum and minimum values of the above parameters are presented in Table 4.7, where I is the current iteration, $maxI$ is the pre-set value of maximum number of iterations, and is constant equal to 10. Particles fly in the search space based on equation (4.2), and equation (4.3). Every particle always remembers its best position in the experience. When a particle moves to another position, new velocity is calculated according to the previous velocity and the distance of its position from both p-best and g-best. However, the new

velocity is limited to the range to control the extreme traveling of particles outside the search space. Particles gain their new positions according to the new velocity and the previous position equation (4.2) and equation (4.3). Liu et al [88] implemented Ranked Order Value (ROV) to convert the continuous position value of the particles to job sequence to solve permutation flowshop scheduling problem. In this study, ROV is implemented to convert the position of particles to part families sequences as well the sequence of jobs in each part family.

Table 4.7: The maximum and minimum values of PSO parameters

Parameter.	Max. values	Min. values
W	2	0.4
C_1	2	0.4
C_2	2	0.4
Position value	0	4
Velocity	-4	4

4.5.1 Initial Swarm

The algorithm starts with generating the initial velocity and position for j^{th} particle in the swarm of size $N\{j \in \{1, 2, ..., N\}$ according to the following equations:

$$X_{0i} = X_{min} + r_1(X_{max} - X_{min}), \tag{4.7}$$

$$V_{0i} = V_{min} + r_2(V_{max} - V_{min}), \tag{4.8}$$

The subscript 0 refers to the starting point, i.e., before the iterations are started. X_{min}, and X_{max} represent the minimum and maximum position values. V_{min}, and V_{max} represent

the minimum and maximum velocities. Also r_1 and r_2 are random variables in the range between $[0, 1]$. Therefore, the initial position of particle X_i will be as follows:

$$X_i = \begin{pmatrix} X_{i11} & X_{i12} & \cdots & X_{i1,N_1} \\ X_{i21} & X_{i22} & \cdots & X_{i2,N_2} \\ \vdots & \vdots & \ddots & \vdots \\ X_{if1} & X_{if2} & \cdots & X_{if,N_f} \\ X_{i(F+1)1} & X_{i(F+1)2} & \cdots & X_{i(F+1)F} \end{pmatrix} \tag{4.9}$$

Particle flies in the search space with velocity matrix V_i

$$V_i = \begin{pmatrix} V_{i11} & V_{i12} & \cdots & V_{i1,N_1} \\ V_{i21} & V_{i22} & \cdots & V_{i2,N_2} \\ \vdots & \vdots & \ddots & \vdots \\ V_{if1} & V_{if2} & \cdots & V_{if,N_f} \\ V_{i(F+1)1} & V_{i(F+1)2} & \cdots & V_{i(F+1)F} \end{pmatrix} \tag{4.10}$$

Where F refers to number of families and N_f is maximum number of jobs of family f.

4.5.2 Ranked Order Value (ROV)

Ranked Order Value (ROV) is based on the random representation of permutation of jobs [83], [88]. ROV is included in the proposed PSO algorithms to convert the continuous position of particles to a permutation of families as well as jobs. Basically, ROV is applied based on Smallest Particle Value (SPV); firstly handled and assigned a smallest rank value 1. Then, the second SPV is assigned a rank value 2. Similarly, all the position values will be dealt with to convert the position information of a particle to a family and a job sequence. For example, if we have particles information about a sequence of 4 families with these values of $X_i = \{0.12, 0.22, 0.06, 1.14\}$, based on ROV method, the sequence of these families is $\{3, 2, 1, 4\}$. ROV is used to convert all random numbers generated to a sequence of families as well as the jobs in each iteration (after updating the velocity and position of particles in the swarm). The steps of the PSO algorithm are executed as follows in Algorithm 4.2.

Algorithm 4.2 The steps of the proposed PSO algorithm

Require: Initialize parameters { swarm size n, maximum Iteration I_{max}, C_{1max}, C_{1min}, $C_{2max}, C_{2min}, V_{max}, V_{min}, W_{max}, W_{min}, X_{max}, X_{min}$}

- Step 1: Set iteration $t = 0$
- Step 2: If $t = 0$
  Generate initial positions X_i^t and V_i^t initial velocities for $i \in \{1, 2, ..., n\}$ according to equations (4.7, and 4.8)
  Else
  Generate a new swarm by updating the velocity V_i^t and position X_i^t of particles according to equations (4.2, and 4.3)
- Step 3: Apply the (ROV) on X_i^t to find the sequence of families as well as jobs in families.
- Step 4: Calculate the objective function $f\ (X_i^t)$ for each particle $i \in \{1, 2, ..., n\}$
  - o Step 4.1 For each particle in the swarm the p-best (P_i^t) is calculated as:

$$P_i^t = \underset{X_i^j}{\operatorname{argmin}} f(X_i^j) \qquad for \quad j \in \{1, 2, ..., t\}$$

  - o Step 4.2 The g-best can be calculated as:

$$G^t = \underset{P_i^t}{\operatorname{argmin}} f(P_i^t) \qquad for \quad i \in \{1, 2, ..., n\}$$

- Step 5 Set $t = t + 1$
- Step 6 If $t = I_{max}$, STOP; else, go to step 3.

4.5.3 Results of the proposed PSO algorithm

The proposed PSO algorithm is coded using C++ language and run on a PC with an Intel core I7 (2.93 GHz) CPU and 4.0 GB memory. The performance of the proposed algorithm has been evaluated using test problems used by Salmasi (Section 3.4.2). The computational results of minimum, average, and maximum TFT are summarized in Tables 4.8, 4.9, and

4.10 for two-machine test problems, and Tables 4.11, 4.12, and 4.13 for the six-machine problems. The analysis of performance versus the proposed particle swarm optimization based on robustness will be presented in Section 4.5. The performance of proposed PSO algorithm is compared with the best available existing algorithms in the literature. Results show that PSO is outperformed by the best available ACO algorithm proposed by [14] for solving the small size problems. However, there is a very small deviation from the best in solving the large size problem (Figure 4.4). Deviation of results from the best available algorithm in literature can be explained by the Average Relative Deviation (ARD_{TFT}) using the following formula:

$$ARD_{TFT} = \left(\frac{TFT_{Algorithm} - TFT_{BEST}}{TFT_{BEST}} \right) \tag{4.11}$$

Where $TFT_{Algorithm}$ refers to the amount of flow time obtained by the proposed algorithm and TFT_{best} is the best TFT obtained by the ACO algorithm developed by [14] which is considered as the best available algorithm (benchmark). Results indicate that the Average Relative Deviation ARD_{TFT} of GA, SA, PSO, and PSO-SA algorithms are: 0.25, 0.28, 0.10, and 0.05 respectively. Therefore, PSO-SA shows the least deviation from the best available algorithm.

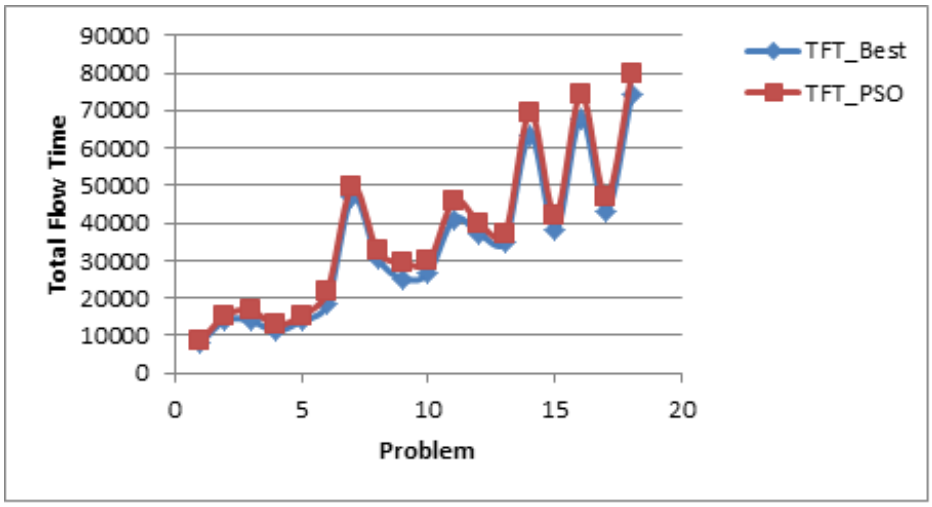

Figure 4.4: Performance of proposed PSO vs. ACO for large test problems

Table 4.8: Results of small test problems for two machine problems obtained by PSO algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
1	4	4	13	2011	2018.5	2026	1.27
2	3	4	10	1425	1428.5	1432	0.95
3	3	4	8	843	843	843	0.91
4	2	3	5	454	454	454	0.64
5	5	4	16	2913	2920.5	2928	1.54
6	3	4	8	986	986	986	0.91
7	5	7	28	1684	1688.5	1693	2.28
8	4	6	22	4310	4320	4330	1.53
9	3	7	17	2599	2604.5	2610	1.17
10	2	7	13	1263	1266.5	1270	0.93
11	5	7	31	8892	8994	9056	2.11
12	4	7	25	5494	5527.5	5561	1.58
13	5	10	31	8129	8151.5	8174	2.21
14	2	8	15	2036	2052	2068	0.96
15	5	9	35	8146	8153	8160	2.17
16	5	10	40	10511	10605.8	10701	2.31
17	4	9	31	6423	6509	6595	1.78
18	4	10	28	5611	5642	5673	1.8

Table 4.9: Results of medium test problems for two machine problems obtained by PSO algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
19	6	4	17	3114	3170.4	3277	1.94
20	8	4	25	6071	6224.2	6555	2.84
21	9	3	22	4437	4521	4605	3.06
22	9	4	27	6000	6316.8	6562	3.21
23	10	4	33	10446	10865.6	11351	3.84
24	6	4	16	3104	3182.2	3259	1.91
25	9	7	60	28763	29160.25	29403	4.15
26	10	7	62	27045	27438.4	27984	4.62
27	6	7	33	7783	7899.4	7996	2.43
28	10	7	52	17730	18441.6	19173	4.53
29	6	7	32	8229	8264.4	8342	2.48
30	10	7	49	18295	18676.75	19421	4.51
31	8	9	41	12912	13241.6	13551	3.77
32	8	10	51	18661	19097.4	19710	3.81
33	10	10	71	23844	24251	25450	6.05
34	6	8	29	5970	6109.6	6285	2.45
35	9	9	58	8054	8104.2	8175	4.58
36	6	10	31	8203	8401.8	8519	2.9

Table 4.10: Results of large test problems for two machine problems obtained by PSO algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
37	11	4	29	8301	8681.2	9092	4.31
38	13	4	40	15399	15997.5	16811	5.54
39	16	4	46	16708	28457	82973	7.72
40	13	4	40	12773	13064.2	13717	5.59
41	14	4	40	15402	15836	16054	6.41
42	15	4	46	21507	22385.6	23444	7.59
43	16	7	79	49639	51460.4	52882	8.91
44	13	7	63	32388	33477.667	34834	6.42
45	16	7	66	29503	30619	32085	8.62
46	12	7	65	29677	30079.33	30415	5.96
47	15	7	76	45988	47891.2	49846	8.11
48	14	7	69	39961	40517.8	41124	7.18
49	11	10	75	36834	37181.6	37451	5.77
50	15	10	99	69564	70770	72254	8.89
51	12	10	83	42122	42468.33	43068	6.46
52	16	10	106	74326	75343.4	77103	10.13
53	15	10	79	46906	48640.2	49784	9.03
54	16	10	108	79639	81793.67	82968	9.77

Table 4.11: Results of small test problems for six machine problems obtained by PSO algorithm

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
1	5	3	11	10256	10256.4	10258	3.5
2	3	4	9	5583	5583.2	5584	2.19
3	4	3	10	1787	1789	1797	2.57
4	2	4	7	879	879	879	1.36
5	4	4	15	12873	12882.6	12891	2.99
6	2	4	8	4762	4762	4762	1.42
7	5	7	29	30236	30286	30306	4.77
8	4	6	21	18265	18273.8	18298	3.37
9	2	7	11	1546	1563.4	1578	1.66
10	5	7	25	5986	6097.6	6269	4.56
11	4	6	15	11791	11799	11815	3.34
12	5	7	20	17290	17342.6	17404	4.58
13	2	10	13	6199	6200.8	6202	1.98
14	3	10	29	20807	20832.4	20860	3.47
15	4	10	28	8331	8418.8	8484	4.15
16	4	9	21	5372	5456.5	5581	3.74
17	4	9	28	25901	25993.2	26096	4.94
18	3	9	21	15289	15319	15344	2.9

Table 4.12: Results of medium test problems for six machine problems obtained by PSO algorithm

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
19	9	4	31	48007	48495	48924	9.36
20	8	4	19	25518	25636.8	25861	7.33
21	6	4	22	5841	5872	5903	5.02
22	9	4	25	7775	7957	8140	8.49
23	8	4	24	34149	34317.2	34610	7.66
24	6	4	16	17988	18012.2	18062	4.9
25	6	7	28	28947	29033.5	29108	6.04
26	7	7	29	34488	34495	34504	7.01
27	8	6	33	10872	11080.8	11365	7.91
28	9	7	38	14294	14645.66667	14837	10.09
29	7	7	34	43581	43746	43934	7.49
30	6	7	31	35526	35583.2	35634	7.25
31	9	9	50	74357	75273	76649	11.34
32	8	10	49	66893	67689	68836	9.99
33	7	10	46	18263	18575.2	18841	8.26
34	10	10	65	35536	36381	36773	13.28
35	10	10	56	102700	103959.2	107183	13.31
36	8	10	46	63135	63473	63883	9.88

Table 4.13: The results of large test problems for six machine problems obtained by PSO algorithm

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
37	16	4	45	114369	115082.4	116511	21.8
38	12	4	32	60969	61994	63700	13.6
39	14	4	37	16442	16585	16779	16.26
40	13	4	40	17634	18171.2	18773	14.59
41	11	4	34	64909	65344.33	65942	11.54
42	14	4	45	109041	110287.2	111888	17.02
43	13	6	47	95180	97387.4	99339	16.14
44	16	7	63	150487	155010.2	159551	24.6
45	15	7	71	44383	45795.4	46643	20.67
46	14	7	60	33555	34089.33	34716	18.55
47	15	7	69	165031	167255	169519	21.76
48	16	7	78	211837	215081.6	222637	24.1
49	13	10	95	221596	223292.2	224898	19.72
50	14	10	80	180983	187778.2	193824	20.93
51	13	10	72	44750	45327.333	45983	18.27
52	15	10	77	50930	51262.33	51809	23.47
53	16	9	100	277755	279233.33	282011	25.83
54	15	10	117	345381	349734.8	355785	24.28

4.6 Hybrid Algorithms

As mentioned in Section 2.4, PSO and SA are combined into two different hybridization strategies. As shown in Figure 4.5, the initial solution of the SA algorithm is generated by PSO using the initial particle generation as mentioned in Section 4.5.1. Then, a new solution is generated by manipulation of the family as well as the job sequences. In the second scheme (named as PSO-SA), a local search based on Simulated annealing (SA) is combined and implemented in each iteration to enhance the quality of g-best in PSO.

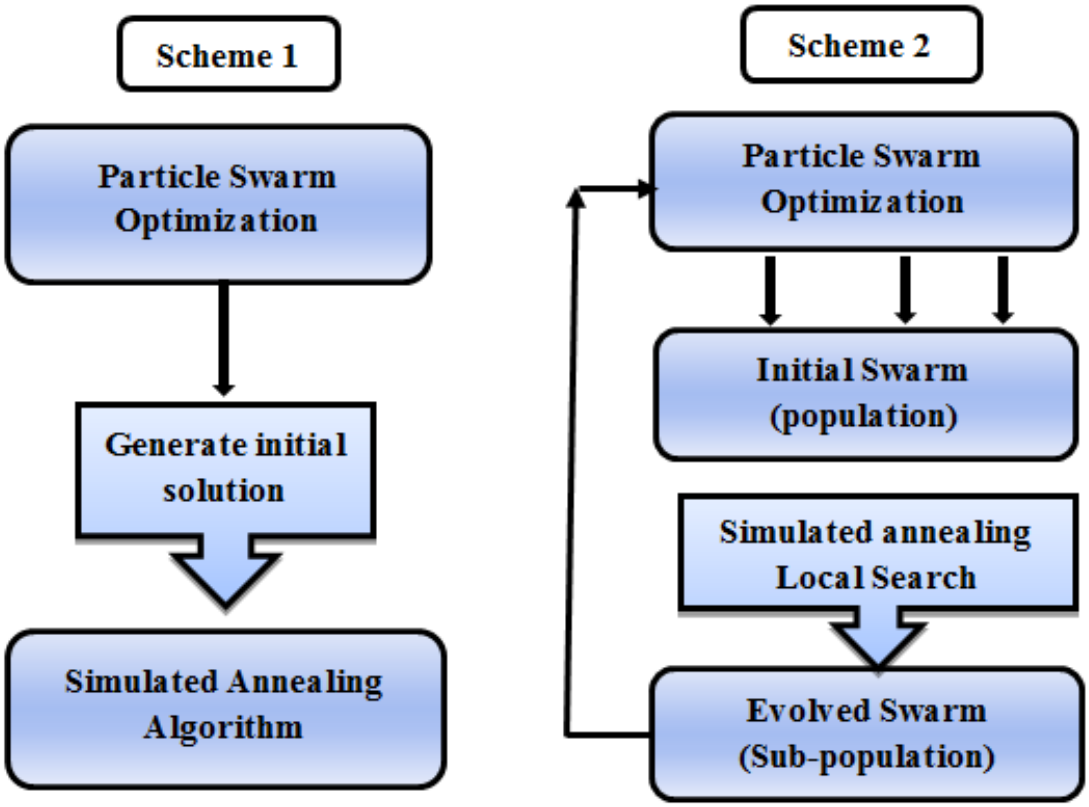

Figure 4.5: Hybridization schemes of PSO and SA

4.7 Simulated Annealing (SA)

SA is a meta-heuristic approach that can provide optimal (or near-optimal) solutions to combinatorial optimization problems. Since its introduction by Kirkpatrick et al (1983) [53]. SA has been applied to a vast number of optimization problems. SA approach starts from an initial sequence (current solution), and then moves successively among the neighboring sequences to generate another solution. Basically, SA is a two step process: perturb (generate a new solution), and then evaluate the quality of the new solution [89]. Eglese provided an overview to implement the SA algorithm [90]. For the sake of getting an overview about simulated annealing algorithm readers are referred to references [91, 92]. A metaheuristic based on SA is proposed by Vakharia et al [93] to schedule part families as well as jobs with each part family for FMCSP with SDST. Sridhar et al [94] proposed an algorithm based on SA for scheduling the FMCSP (without considering the setup times) to minimize the total flow time.

Annealing is the process through which slow cooling of metal produces uniform, low energy-state crystallization, whereas fast cooling produces poor crystallization. The optimization procedure of SA algorithm to find a near global minimum mimics the crystallization cooling procedure. SA procedure starts with a random initial solution as the current solution. Then, the algorithm generates a new solution from the predetermined neighbourhood. The initial solution is randomly generated by using the initial position and velocity that is described in Section 4.5.1. The steps of SA algorithm are shown in Algorithm 4.3. The fitness value of the new solution is then compared with the current solution to determine if

the new one is better. For minimization problems, if the fitness value of the new solution is smaller than that of the current one, the new solution is automatically accepted and becomes the current solution from which the search continues. The algorithm will then proceed with further iterations. Larger fitness values for next solution may also be accepted as the current solution under certain conditions to escape from a local minimum.

Algorithm 4.3 The steps of the proposed SA algorithm

Require: Initialize SA parameters and set a solution X to be current solution { T_0, T_F, $\alpha = 0.95, I_{iter}$, $N_{non-imp.}$}

- Step 1: Set iteration $t = 0$
- Step 2: For F=1 to Family size
  Set $T = T_0$, No. of Moves = 1, and counter = 1
  - Step 2.1: while $T > T_F$ do:
    - Step 2.1.1: if F=1, Generate by swapping the family sequence
      Else
      Generate a new neighbour solution Y from current solution X by minor swapping on job sequence of Fth family, and calculate the difference in objective functions of X, and Y to be $\triangle E$= $TFT(Y)$ - $TFT(X)$
    - Step 2.1.2: IF $\triangle E \leq 0$, then replace the current solution with the new solution and go to Step 2.1.4; otherwise, go to Step 2.1.3
    - Step 2.1.3: Generate a random number RN in the range $[0, 1]$
      IF$RN < \frac{T}{T^2+(\triangle E)^2}$, then replace solution with solution Y and go to Step 2.1.4
    - Step 2.1.4: Increment No. of Moves by one. IF No. of Moves $<$ $N_{non-imp.}$, then go to Step 2.1.1; Else, go to Step 2.1.5
    - Step 2.1.5: IF the best solution is improved, then restart the counter from zero; otherwise, increase the counter by one unit
    - Step 2.1.6 IF counter = I_{iter}, decrease the temperature using geometric schedule T= αT , make No. of Moves =1, and go to Step 2.1.
- Step 3: Return the current solution in the end as X_{best} with TFT_{best}

4.7.1 SA procedure

The procedure of the proposed SA starts with setting the levels of parameters. The current state or temperature T is set to the initial value T_0. An initial solution X is randomly generated and it is considered as the current solution. For each iteration, the next solution Y is generated from the current solution by perturbation on job sequence or on family sequence using the swapping and insertion techniques. Consider E_i as the energy state of the fitness value of current solution and E_j is the fitness value of the new solution obtained as a result of the manipulation of the job sequence within a family F. Let $\triangle E = E_j$ - E_i; which refers to the difference between $\triangle E = TFT(Y)$ - $TFT(X)$. If $\triangle E \leq 0$, the probability of replacing current solution X with the new solution Y is 1. If $\triangle E > 0$, then, probability of accepting the new solution depends on the Cauchy $\left(\frac{T}{T^2+(\triangle E)^2}\right)$ function. The new solution replaces the current solution, if $\left(\frac{T}{T^2+(\triangle E)^2}\right)$ is greater than some random number RN between 0 and 1. Then, temperature T is reduced based on cooling rate after running maximum number of iterations ($Iite$) from the previous decrease, according to the formula (the typical value of α is 0.95). The algorithm is terminated if T is lower than T_f. During the search evolutions, the best solution with the least total flow time is recorded. The algorithm is also terminated if the current solution is not improved in non-improving successive reductions in temperature. Following the termination of the SA procedure, the near- global optimal schedule is X_{best} with total flow time TFT_{best}.

4.7.2 Selecting the parameters setting of the SA algorithms using the Taguchi method

Selecting levels of an algorithm's parameters is a very crucial step to develop an efficient algorithm. Suitable values for the parameters of a metaheuristic algorithm significantly affects its performance [87]. SA algorithm begins with six parameters, namely number of iterations (I_{iter}), initial temperature (T_0), final temperature (T_F), cooling schedule ($C.S$), cooling rate ($C.R$), and number of non-Improving ($N_{non-imp.}$) or the allowable number of reductions in temperature where the best objective function value obtained is not improved during the SA procedure. Note that the Boltzman coefficient constant ($\alpha = 0.95$) is used in the cooling rate reduction. In fact, performance of SA is very sensitive to the choice of the levels of the mentioned parameters. The selection of the best combination of these parameters should be conducted in a scientific manner. In this study, the Taguchi method is implemented to identify the best combination among the candidate values for each parameter that is listed in Table 4.14. Among the experimental designs provided by Taguchi, L27 (3^6) design is selected to investigate the effect of these parameters and identify the best combination to be used in the execution of the SA algorithm; It represents 27 experiments with six three-level factors Table 4.15. The full factorial design represents a classical approach that considers all possible combinations of the input parameters. In this case 729 (3^6) experiments would be performed. Yet, 27 experiments will prove to the same conclusion.

The experiments were performed using the proposed SA and RPD which is the ratio of the deviation from the corresponding LB. The results are shown in Table 4.15. Further, Signal-

Table 4.14: List of the SA parameters and the corresponding values

Parameter	Candidate values (levels: 1, 2, 3)
$InitialTemp(T_0)$	10, 20, 30
$FinalTemp(T_f)$	1, 5, 10
$Coolingschedule(C.S)$	Boltzmann, Cauchy
$CoolingRate(C.R)$	Linear, Geometric, and Logarithmic
$No.ofIterations(Iter)$	50, 150, 300
$No.ofnon-Improving(Non-Imp.)$	10, 20, 30

Table 4.15: Results of the L27 design

Initial Temp.	Final Temp.	Cooling schedule	Cooling rate	No. non Improvement	No. Iterations	Lower Bound	Total Flow Time	Deviation from LB	SNR
1	1	1	1	1	1	1857.2	2581	0.39	8.185
1	1	1	1	2	2	964.2	1252	0.299	10.499
1	1	1	1	3	3	8831	9715	0.1	19.991
1	2	2	2	1	1	3567.6	5357	0.502	5.993
1	2	2	2	2	2	23298.6	30990	0.33	9.626
1	2	2	2	3	3	12513	16856	0.3471	9.191
1	3	3	3	1	1	7462.4	9481	0.27	11.357
1	3	3	3	2	2	23915	34286	0.434	7.257
1	3	3	3	3	3	22054	45863	1.08	-0.665
2	1	2	3	1	2	1363.5	2031	0.49	6.204
2	1	2	3	2	3	705	715	0.014	36.963
2	1	2	3	3	1	14806	19972	0.349	9.146
2	2	3	1	1	2	3810	5181	0.36	8.877
2	2	3	1	2	3	11637	15528	0.334	9.516
2	2	3	1	3	1	10327.1	14268	0.382	8.367
2	3	1	2	1	2	20350.6	23588	0.159	15.976
2	3	1	2	2	3	24770.7	172930	5.981	-15.53
2	3	1	2	3	1	30250	86292	1.853	-5.356
3	1	3	2	1	3	10256	10548	0.028	30.912
3	1	3	2	2	1	4502	4812	0.069	23.241
3	1	3	2	3	2	11678	11955	0.023	32.498
3	2	1	3	1	3	4728.3	6533	0.382	8.366
3	2	1	3	2	1	34477	44083	0.279	11.099
3	2	1	3	3	2	74057	91356	0.234	12.631
3	3	2	1	1	3	60370	89327	0.48	6.3813
3	3	2	1	2	1	19517	39777	1.038	-0.325
3	3	2	1	3	2	32389	48258	0.49	6.197

To-Noise Ratio (SNR) is calculated to indicate the amount of variation in the response variable. For example, in case of the minimization problem, the signal-to-noise ratio is maximized. The corresponding SNR is given by the following formula [20]:

$$SNR = -10\log_{10}\left(MS\right)^2, \text{where MS is the mean square}, MS = \left[\frac{1}{n}\sum_{i=1}^{n} y_i^2\right] \tag{4.12}$$

The main effect plot of the parameters is shown in Figure 4.6. It shows that not improving, and the number of iterations and the cooling rate are insignificant compared to the final and initial temperatures, and cooling schedule. Therefore, considering the interaction among these parameters (as shown in Figure 4.7) is a crucial step to get a better understanding of

the performance of the parameters at different levels. For instance, selecting the high level of the initial temperature with the least level of final temperature is most preferred regardless of the cooling scheduling function Figures 4.8, and 4.9.

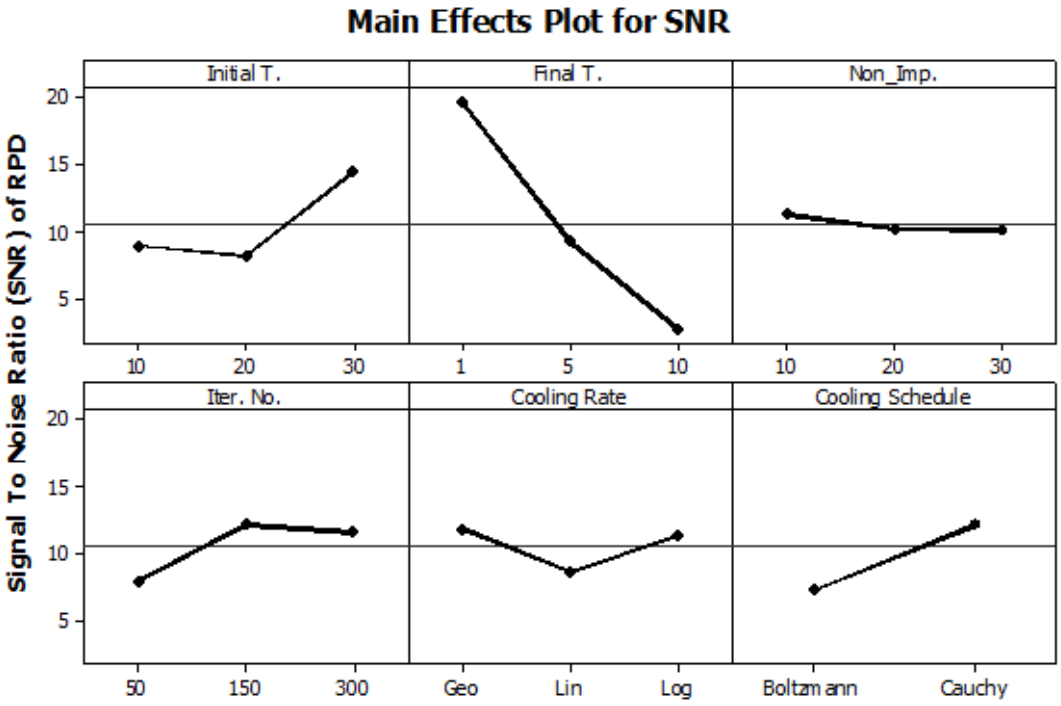

Figure 4.6: The main effect plot of the SA algorithm

The selected cooling schedule is strongly affected with the cooling rate and the number of iterations. The interesting point in selecting the cooling schedule can be highlighted from Figure 4.10. If Cauchy function is considered as a cooling schedule, the best cooling rate is geometric rate (Geo.). However, linear rate (Lin) is preferred if Boltzmann function is considered as a cooling schedule. Therefore, Cauchy function is used instead of the Boltzmann function (commonly used in SA) with geometric cooling rate.

The other parameters are as follows: T_f =1, T_0=20, I_{iter} =150, and $N_{non-imp.}$ = 15.

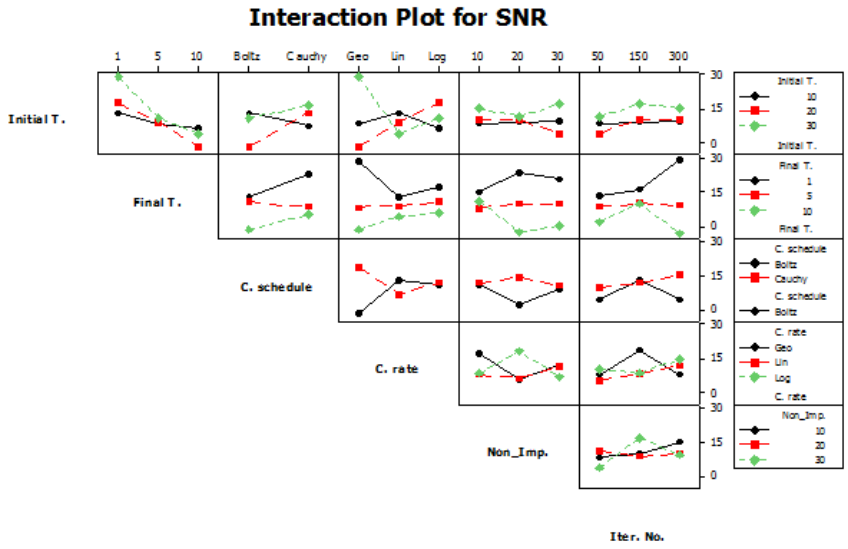

Figure 4.7: The interaction plot of the parameter vs. the signal-to-noise ratio

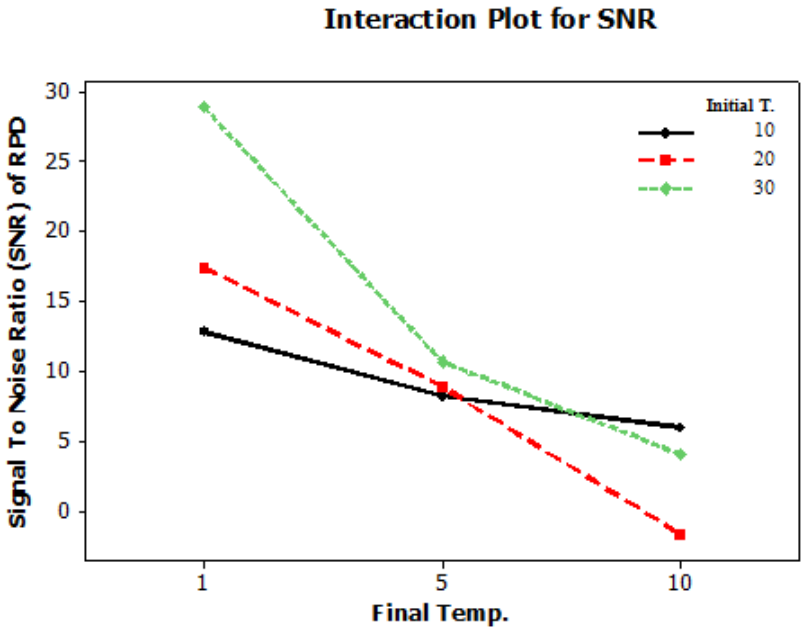

Figure 4.8: The interaction between the initial and final temp.

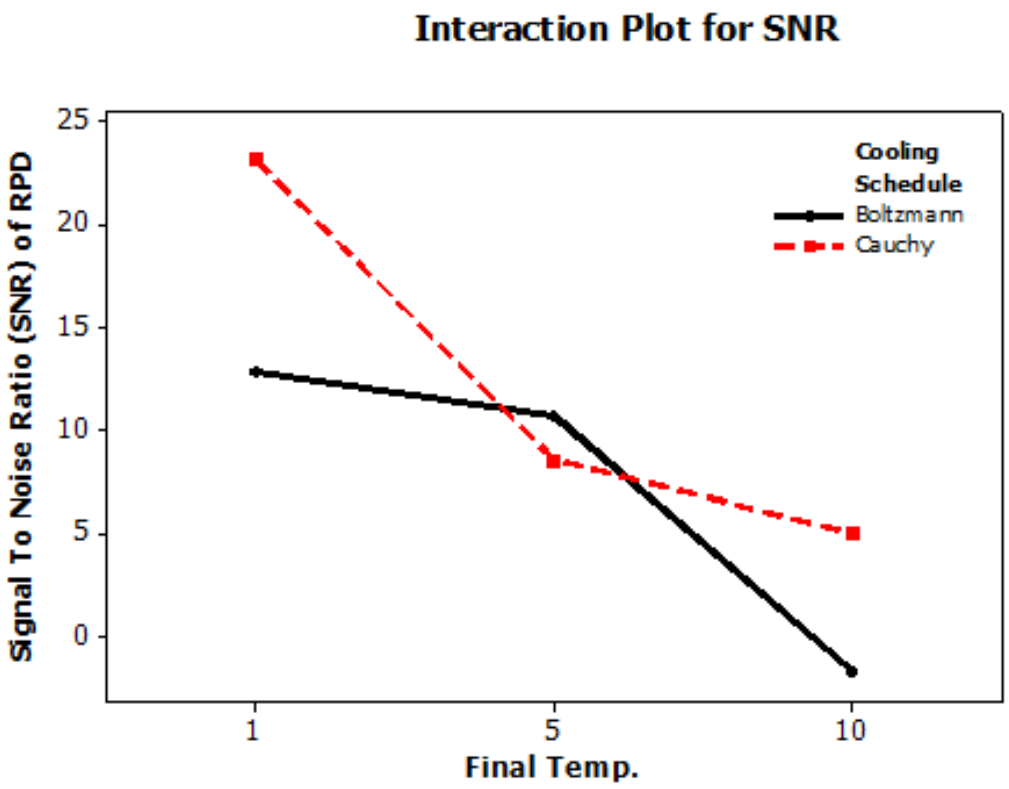

Figure 4.9: The interaction between the final temp. and cooling schedule

4.7.3 Results of proposed SA algorithm

The proposed SA algorithm is coded using C++ language and run on a PC with an Intel core I7 (2.93 GHz) CPU and 4.0 GB memory. The performance of the proposed algorithm is evaluated using test problems proposed by Salmasi (Section 3.4). The computational results of Minimum, Average, and Maximum are summarized in Tables 4.16, 4.17, and 4.18, for the two-machine problems and Tables 4.19, 4.20, and 4.21 for the six-machine problems. The analysis of performance against the proposed particle swam optimization based on the robustness is presented in Section 4.5. The performance of proposed SA algorithm shows a small deviation from the best available existing algorithms in the literature. Figure 4.11,

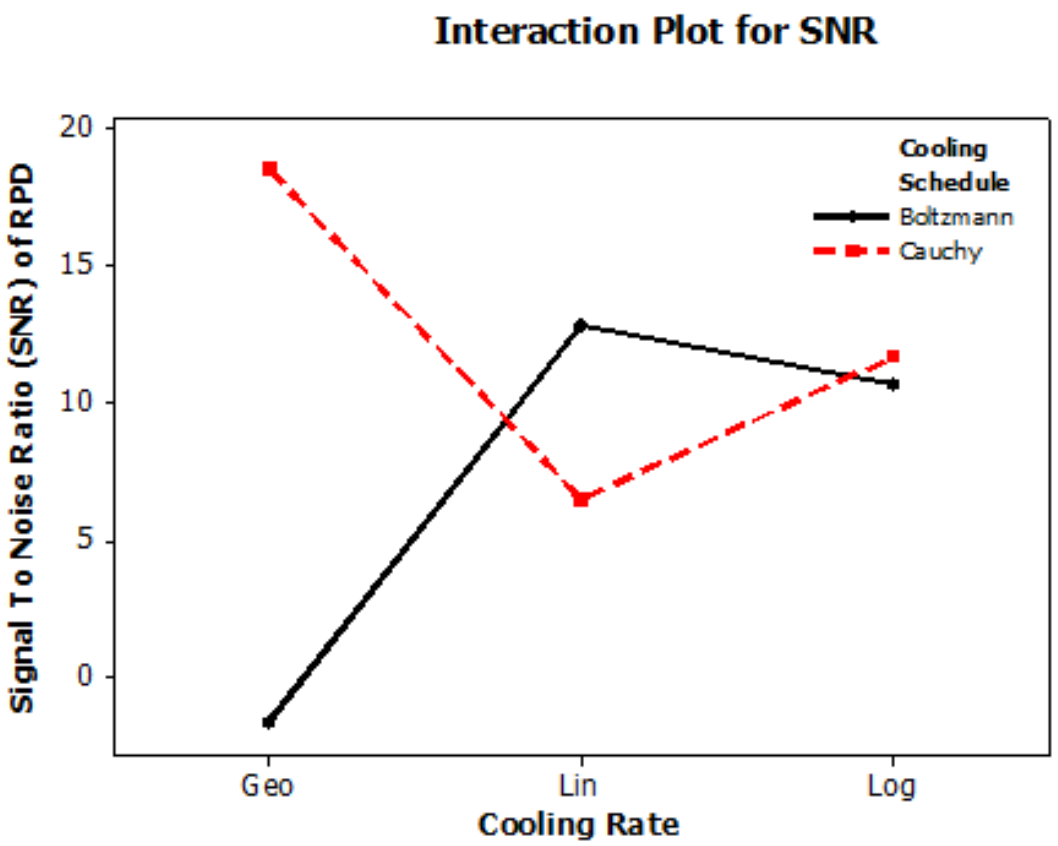

Figure 4.10: Iteration between cooling schedule and cooling rate

highlights differences between solutions found by the proposed SA and the best available existing algorithm (ACO) proposed by [14] in solving the small size problems. In addition, as shown in Figure 4.12, the difference between solutions is increased for the large size problem.

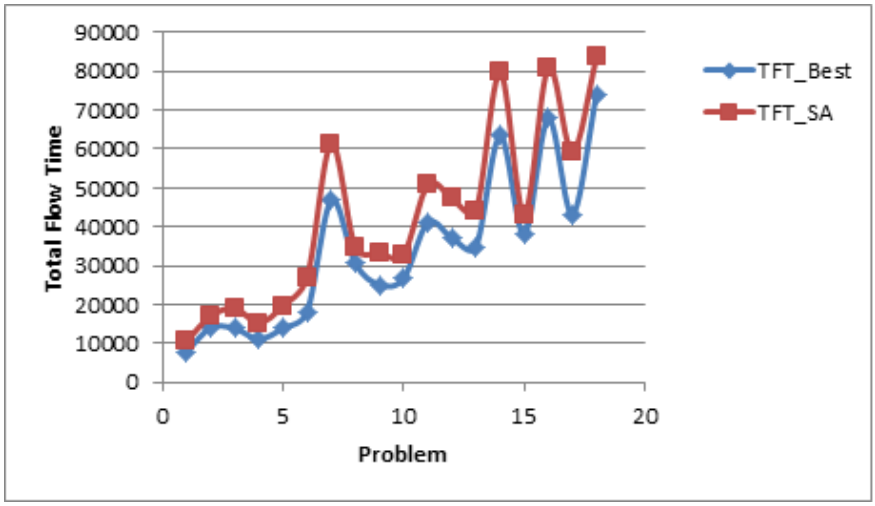

Figure 4.11: Comparison between the proposed SA and ACO for small size problems

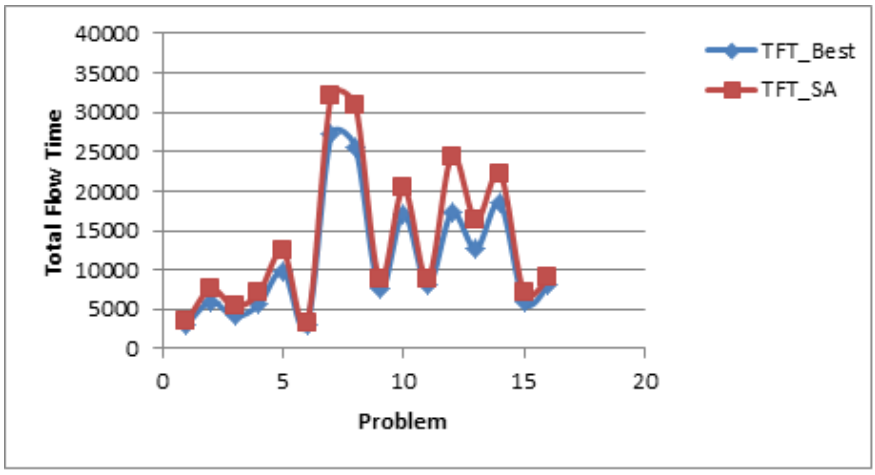

Figure 4.12: Comparison between the proposed SA and ACO for large size problems

Table 4.16: The results of small test problems for two machine problems obtained by SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
1	4	4	13	2187	2337.8	2553	1.8
2	3	4	10	1432	1443.4	1458	1.2
3	3	4	8	812	871.4	887	1.18
4	2	3	5	454	460.4	486	0.62
5	5	4	16	3525	3583.6	3636	2.55
6	3	4	8	1001	1129.2	1275	1.16
7	5	7	28	1848	2010.6	2090	3.79
8	4	6	22	4420	4845.6	5304	2.31
9	3	7	17	2735	2852.6	2951	3.18
10	2	7	13	1369	1528.2	1640	0.87
11	5	7	31	9160	9632.8	9767	3.37
12	4	7	25	5679	5706.4	5718	2.32
13	5	10	31	8651	9369	9845	6.06
14	2	8	15	2168	2178	2188	1.63
15	5	9	35	8326	8966.4	9277	4.8
16	5	10	40	10964	11595	11779	6.03
17	4	9	31	7228	7279.4	7329	3.97
18	4	10	28	6181	7245.2	8100	3.91

Table 4.17: The results of medium test problems for two machine problems obtained by SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
19	6	4	17	3455	3732.4	3919	3.66
20	8	4	25	7569	7799	7878	9.62
21	9	3	22	5365	5369.4	5375	11.59
22	9	4	27	7095	7550.2	7780	11.87
23	10	4	33	12418	12451.2	12496	10.16
24	6	4	16	3325	3509.2	3855	3.73
25	9	7	60	32135	32820.6	33804	9.2
26	10	7	62	30842	30872.4	30946	13.34
27	6	7	33	8863	9629.4	10495	4.99
28	10	7	52	20367	21115.6	21639	13.03
29	6	7	32	8827	9821	10234	5.11
30	10	7	49	24433	24491	24545	10.96
31	8	9	41	16301	17189	17793	5.02
32	8	10	51	22242	22263.4	22290	9.87
33	10	10	71	27014	27055	27082	19.04
34	6	8	29	7017	7038	7050	5.56
35	9	9	58	11567	11575.4	11580	12.41
36	6	10	31	9053	9359.2	9490	6.78

Table 4.18: The results of large test problems for two machine problems obtained by SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
37	11	4	29	10633	10645	10660	13.65
38	13	4	40	17086	17819	19024	20.61
39	16	4	46	19099	19144.4	19202	34.08
40	13	4	40	15342	15429.6	15543	16.16
41	14	4	40	19706	19736.6	19773	24.52
42	15	4	46	27068	27309.6	27466	29.01
43	16	7	79	61432	62212.8	63362	41.24
44	13	7	63	34556	35962.4	37112	26.61
45	16	7	66	33244	33874.8	34681	41.32
46	12	7	65	32877	33887	34350	24.53
47	15	7	76	50863	50945	51057	38.32
48	14	7	69	47402	47526.4	47611	30.58
49	11	10	75	44213	44267.2	44408	34.83
50	15	10	99	79734	80920.6	81955	62.61
51	12	10	83	43251	45283.8	47554	48.63
52	16	10	106	81025	81965	83043	64.6
53	15	10	79	59283	59402.2	59460	49.11
54	16	10	108	83656	84910.8	86648	71.22

Table 4.19: The results of small test problems for six machine problems obtained by SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
1	5	3	11	10260	10472.8	10526	5.22
2	3	4	9	5599	5610.6	5626	2.14
3	4	3	10	1890	1963	1997	3.3
4	2	4	7	890	999.4	1077	1.23
5	4	4	15	13097	13908.8	15004	3.8
6	2	4	8	4795	4866.4	4916	1.25
7	5	7	29	30653	31763	33166	8.49
8	4	6	21	18583	19210.6	19773	4.34
9	2	7	11	1640	1660.2	1684	1.61
10	5	7	25	6761	6871.2	6996	7.59
11	4	6	15	11889	11948.8	12014	4.28
12	5	7	20	18872	20097.6	20824	7.31
13	2	10	13	6285	6285	6285	2.35
14	3	10	29	20973	21376.2	22204	5.49
15	4	10	28	9118	9279.2	9613	6.9
16	4	9	21	5532	5769.6	6025	5.69
17	4	9	28	26469	27557	28292	7.01
18	3	9	21	15289	16420	17858	4.58

Table 4.20: The results of medium test problems for six machine problems obtained by SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
19	9	4	31	49675	50653.8	51521	22.41
20	8	4	19	27663	29025	29924	17.13
21	6	4	22	6139	6428	6547	6.66
22	9	4	25	9081	9206.6	9366	22.07
23	8	4	24	37179	38348.4	40236	17.61
24	6	4	16	18690	19576.8	19904	8.6
25	6	7	28	29107	30311.2	30619	10.97
26	7	7	29	39626	42407.4	44101	14.13
27	8	6	33	12100	12775.2	13813	18.78
28	9	7	38	15637	16186.4	16712	18.97
29	7	7	34	49561	50366.4	50952	13.69
30	6	7	31	39457	41341.8	42061	10.05
31	9	9	50	86452	88914.6	91343	25.43
32	8	10	49	78070	82914.8	85649	27.79
33	7	10	46	19128	19409	19591	20.46
34	10	10	65	38058	38551.8	39073	44.47
35	10	10	56	105494	108521.6	109827	41.46
36	8	10	46	70810	75622.8	78886	29.04

Table 4.21: The results of large test problems for six machine problems obtained by SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
37	16	4	45	136671	137601.6	137867	96.04
38	12	4	32	69539	69864.8	69983	45.68
39	14	4	37	19051	19082.2	19111	67.65
40	13	4	40	20036	20158	20316	56.91
41	11	4	34	71000	72338.6	74025	38.93
42	14	4	45	123021	124175.4	124514	71.18
43	13	6	47	102844	104724.6	106220	61.35
44	16	7	63	185387	189291.2	193206	105.89
45	15	7	71	51014	51821.8	52491	91.67
46	14	7	60	35936	36235.2	36549	69.39
47	15	7	69	205343	206693	210320	92.37
48	16	7	78	232966	233032.2	233135	123.38
49	13	10	95	248389	248437.8	248496	74.06
50	14	10	80	223129	226959.2	230937	85.97
51	13	10	72	46656	47897.6	49119	72.09
52	15	10	77	53939	54023.4	54084	103.02
53	16	9	100	312725	314725	316020	124.73
54	15	10	117	370509	378919.6	385824	118.01

4.8 Hybrid Particle Swarm Optimization algorithm

The main goal of hybridizing the PSO algorithm is to improve the quality of the obtained solution using a pure PSO metaheuristic. SA is basically combined with PSO as a local search to find better solutions. At the first stage, PSO is proposed (Section 4.5) to solve the studied problem, and it showed a better performance than GA. Yet, PSO algorithm finds very good solutions that matched the best existing algorithms for small and medium instances (number of families less than 10 families, and maximum number of jobs in a family is less than 6 jobs). For large problems, the performance of the PSO algorithm deviates from the best solutions due to the large solution space, and the higher probability to be trapped in local optima. Therefore, PSO has been combined with SA to improve the quality of solutions and reduce the variation from the lower bounds for the large problem instances. The main role of combining SA algorithm with PSO is to enhance the intensification and search for a better solution within the neighbourhoods (this combination is named PSO-SA).

In the PSO-SA, each particle flies in the solution space seeking for a better position based on its best experience (p-best), and the global experience (g-best) based on the new velocity and new position (equations 4.7, and 4.8). Therefore, particles are enhanced to move toward the global optima. A local search engine based on SA is implemented to search on the neighbourhood of the g-best particle each iteration by using swapping of the job sequences in families. Note that swapping is implemented only to the job sequence for simplicity. Local search is conducted on the g-best particle in each iteration. The procedure of the proposed PSO-SA algorithm is similar to the proposed PSO algorithm. As compared to algorithm 4.2,

the PSO-SA algorithm shown in algorithm 4.4 differs only in step 6, where a local search based on SA (denoted by L.S) is incorporated with PSO to improve g-best particle that improves the searching and finding promising regions in next iterations of PSO.

Algorithm 4.4 The steps of the proposed PSO-SA algorithm

Require: Initialize parameters { swarm size n, maximum Iteration I_{max}, C_{1max}, C_{1min}, $C_{2max}, C_{2min}, V_{max}, V_{min}, W_{max}, W_{min}, X_{max}, X_{min}$}

- Step 1: Set iteration $t = 0$
- Step 2: If $t = 0$
  Generate initial positions X_i^t and V_i^t initial velocities for $i \in \{1, 2, ..., n\}$ according to equations (4.7, and 4.8)
  Else
  Generate a new swarm by updating the velocity V_i^t and position X_i^t of particles according to equations (4.2, and 4.3)
- Step 3: Apply the (ROV) on X_i^t to find the sequence of families as well as jobs in families.
- Step 4: Calculate the objective function $f\ (X_i^t)$ for each particle $i \in \{1, 2, ..., n\}$
  - Step 4.1 For each particle in the swarm the p-best (P_i^t) is calculated as:

$$P_i^t = \underset{X_i^j}{\operatorname{argmin}} f(X_i^j) \qquad for \quad j \in \{1, 2, ..., t\}$$

  - Step 4.2 The g-best can be calculated as:

$$G^t = \underset{P_i^t}{\operatorname{argmin}} f(P_i^t) \qquad for \quad i \in \{1, 2, ..., n\}$$

- Step 5 Set $t = t + 1$
- Step 6 Set $G^t = L.S(G^t)$
- Step 7 If $t = I_{max}$, STOP; else, go to step 2.

4.8.1 Results of the PSO-SA algorithm

The proposed PSO-SA algorithm is coded using C++ language and run on a PC with an Intel core I7 (2.93 GHz) CPU and 4.0 GB memory. The computational results of minimum, average, and maximum of the TFT are summarized in Tables 4.22, 4.23, and 4.24, for the two-machine problems and Tables 4.25, 4.26, and 4.27 for the six-machine problems respectively. The analysis of performance of PSO-SA versus the other proposed algorithms is presented in Section 4.5. The performance of the proposed PSO-SA algorithm is compared with the best available algorithms in the literature. PSO-SA performs similar to the best available algorithm (ACO) [14] for solving most of the test problems for the six-machine test. However, there is a very small deviation from the best in solving the large size problems for two-machine test problems as shown in Figure 4.13.

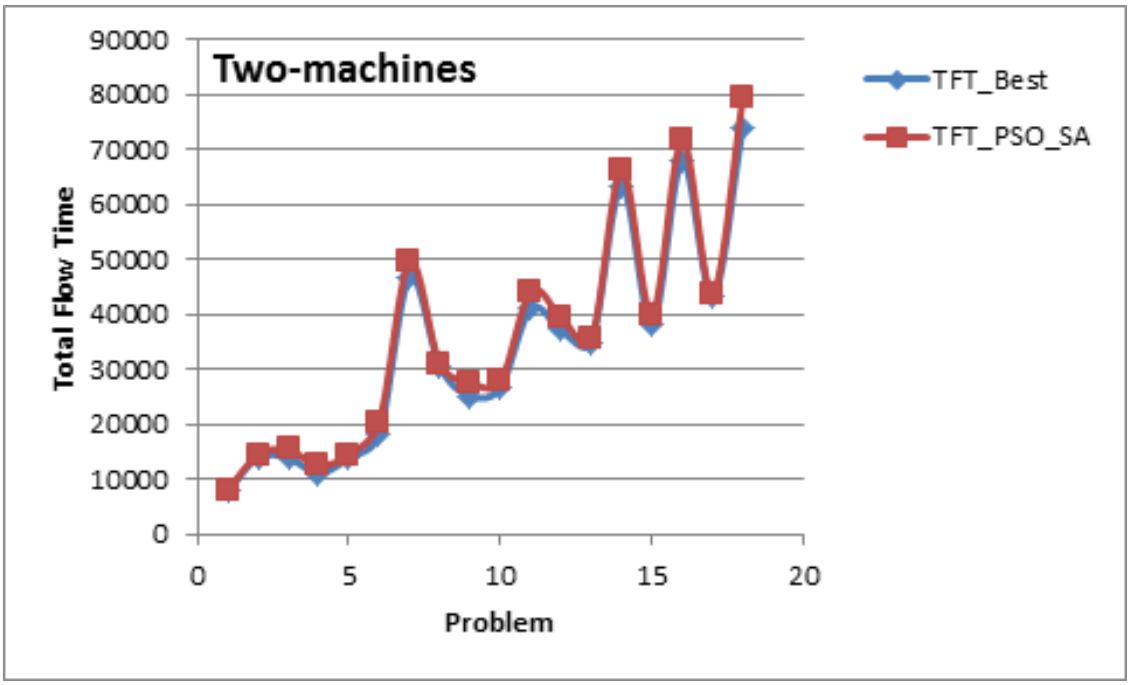

Figure 4.13: Performance of the proposed PSO-SA vs. ACO for 2-machine problems

Table 4.22: Results of small test problems for two machine problems obtained by PSO-SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
1	4	4	13	2011	2011	2011	12.63
2	3	4	10	1425	1425	1425	8.33
3	3	4	8	812	827.5	843	8.67
4	2	3	5	454	454	454	5.68
5	5	4	16	2912	2938	2977	15.99
6	3	4	8	986	986	986	4.56
7	5	7	28	1684	1689	1709	23.44
8	4	6	22	4297	4301	4302	14.14
9	3	7	17	2592	2592	2592	10.75
10	2	7	13	1263	1263	1263	7.86
11	5	7	31	8847	8864.4	8882	15.23
12	4	7	25	5457	5495.8	5579	12.13
13	5	10	31	8079	8103.7	8119	21.16
14	2	8	15	2023	2026.6	2029	8.64
15	5	9	35	8035	8066.4	8111	21.69
16	5	10	40	10346	10437	10532	23.62
17	4	9	31	6381	6441.4	6622	20.43
18	4	10	28	5583	5626	5677	20.81

Table 4.23: Results of medium test problems for two machine problems obtained by PSO-SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
19	6	4	17	3106	3281	3452	19.08
20	8	4	25	6039	6195.2	6483	31.02
21	9	3	22	4195	4411.8	4576	34.61
22	9	4	27	5730	6095	6537	36.08
23	10	4	33	10004	10300.4	10769	43.56
24	6	4	16	3043	3146	3225	18.50
25	9	7	60	27717	28313	28620	46.84
26	10	7	62	25760	26390.4	27649	41.02
27	6	7	33	7650	7832.25	8242	23.66
28	10	7	52	17616	18514	18972	39.37
29	6	7	32	8066	8128.4	8226	24.09
30	10	7	49	17717	18135	18658	51.52
31	8	9	41	12815	12962.6	13045	44.55
32	8	10	51	18536	18942	19353	47.42
33	10	10	71	22237	22910.4	23409	81.14
34	6	8	29	5907	6040.6	6239	26.00
35	9	9	58	7719	7879	8259	52.23
36	6	10	31	8129	8174.8	8212	34.13

Table 4.24: Results of large test problems for two machine problems obtained by PSO-SA algorithm

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
37	11	4	29	8148	8340.3	8637	27.81
38	13	4	40	14460	15183.2	16297	37.43
39	16	4	46	15457	16151.5	17157	54.96
40	13	4	40	12535	12848.6	13238	36.84
41	14	4	40	14553	14871.6	15170	41.14
42	15	4	46	20501	21625.2	22255	47.62
43	16	7	79	49599	50240.7	50598	67.59
44	13	7	63	31051	32194.4	33431	37.63
45	16	7	66	27586	29104.8	30356	67.58
46	12	7	65	27831	28808.9	29834	38.63
47	15	7	76	44210	45612.2	48457	59.12
48	14	7	69	39555	40384.8	41559	51.01
49	11	10	75	35553	36394.8	37586	42.05
50	15	10	99	66384	68045	70723	72.00
51	12	10	83	40116	41557.4	42833	44.05
52	16	10	106	71784	73093.8	75419	78.39
53	15	10	79	43908	46308.2	49471	66.96
54	16	10	108	79404	81432.3	83151	81.57

Table 4.25: Results of small test problems for six machine problems obtained by PSO-SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
1	5	3	11	10256	10261	10266	22.14
2	3	4	9	5583	5583	5583	13.60
3	4	3	10	1787	1787	1787	17.20
4	2	4	7	879	879	879	8.50
5	4	4	15	12867	12867	12867	15.02
6	2	4	8	4762	4762	4762	9.28
7	5	7	29	30226	30233	30240	32.50
8	4	6	21	18235	18235.7	18236	21.64
9	2	7	11	1546	1551.7	1562	10.55
10	5	7	25	5977	6050.3	6159	29.74
11	4	6	15	11783	11787	11791	21.06
12	5	7	20	17273	17282	17300	29.33
13	2	10	13	6198	6199.3	6201	12.54
14	3	10	29	20795	20816.7	20832	22.68
15	4	10	28	8292	8325.3	8355	28.45
16	4	9	21	5331	5367.7	5421	24.98
17	4	9	28	25875	25909	25932	26.28
18	3	9	21	15245	15270.66667	15311	19.97

Table 4.26: Results of medium test problems for six machine problems obtained by PSO-SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
19	9	4	31	47870	48417.8	49225	53.34
20	8	4	19	25487	29988.8	47870	40.25
21	6	4	22	5821	5886.8	6096	25.77
22	9	4	25	7704	7793	7907	46.33
23	8	4	24	34128	34176.6	34310	41.83
24	6	4	16	17977	17997	18027	25.81
25	6	7	28	28877	28925.6	29064	32.63
26	7	7	29	34478	34545.1	34851	40.03
27	8	6	33	10298	10504	10931	62.01
28	9	7	38	13701	14068.6	14440	72.12
29	7	7	34	43483	43517.2	43560	55.09
30	6	7	31	35359	35386	35417	43.70
31	9	9	50	74112	74593.9	75463	93.61
32	8	10	49	66741	66794.8	66824	77.26
33	7	10	46	17709	18052	18291	63.62
34	10	10	65	34596	34963	35401	109.61
35	10	10	56	101711	102175.2	102688	107.61
36	8	10	46	62876	63096.4	63704	79.37

Table 4.27: Results of large test problems for six machine problems obtained by PSO-SA algorithm.

Pro. No.	No. of Families.	Max Job in Family	Total Jobs	**Total Flow Time**			CPU Time(Sec)
				Min.	Ave.	Max	
37	16	4	45	112412	113395.6667	114244	191.7
38	12	4	32	60230	60533.3	60935	95.05
39	14	4	37	15199	16078.3	16929	141.13
40	13	4	40	17108	17601	18422	119.25
41	11	4	34	64619	65199.9	65923	89.39
42	14	4	45	108378	109312.2	110597	148.23
43	13	6	47	94329	95787.2	96895	136.03
44	16	7	63	148372	151411.4	155966	225.66
45	15	7	71	42343	43549.4	44963	201.67
46	14	7	60	32456	33643	35055	171.61
47	15	7	69	162365	165186.8	167450	197.25
48	16	7	78	210727	212962.5	216698	231.1
49	13	10	95	218407	218966.2	219598	197.83
50	14	10	80	180513	183222.6	186367	206.37
51	13	10	72	43331	43770.6	44231	188.63
52	15	10	77	49949	51288.4	52617	245.62
53	16	9	100	277566	279902.2	284628	288.46
54	15	10	117	344017	346843.2	350002	281.72

4.9 Performance Analysis and Discussion

Designing an efficient algorithm aims to obtain a high quality solution in reasonable computational efforts. Thus, evaluating the performance of a proposed metaheuristics must be done in a rigorous manner to find the most efficient one to cope with the studied problem. Basically, analysis of the developed GA, PSO, SA, and PSO-SA is aimed to find the best algorithm based on the following criteria:

1. Relative Percentage Deviation from the lower bounds; the best algorithm is the one that shows minimal

2. Robustness of metaheuristic via Design of Experiment:

    - Analyzing the interaction among test problems parameters and the proposed algorithms.
    - Evaluate the ability of the proposed algorithms to solve a wide range of test problems with less variation.

4.9.1 Relative Percentage Deviation (RPD)

Relative Percentage Deviation (RPD) measures the response based on the deviation of the total flow time from the corresponding lower bound. The RPD is calculated using the following formula:

$$RPD = \left(\frac{TFT - LB}{LB}\right) \times 100 \tag{4.13}$$

In the case of the two machine test problems (as shown in Figure 4.14), SA and GA have the same performance for the small, medium, and large instances. However, there is a significant difference between the PSO and PSO-SA compared with GA and SA. PSO-SA matched the best available algorithm proposed by [14] for small and medium test problems. Further, as shown in Figure 4.15, the same trend can be recognized for the six machine instances.

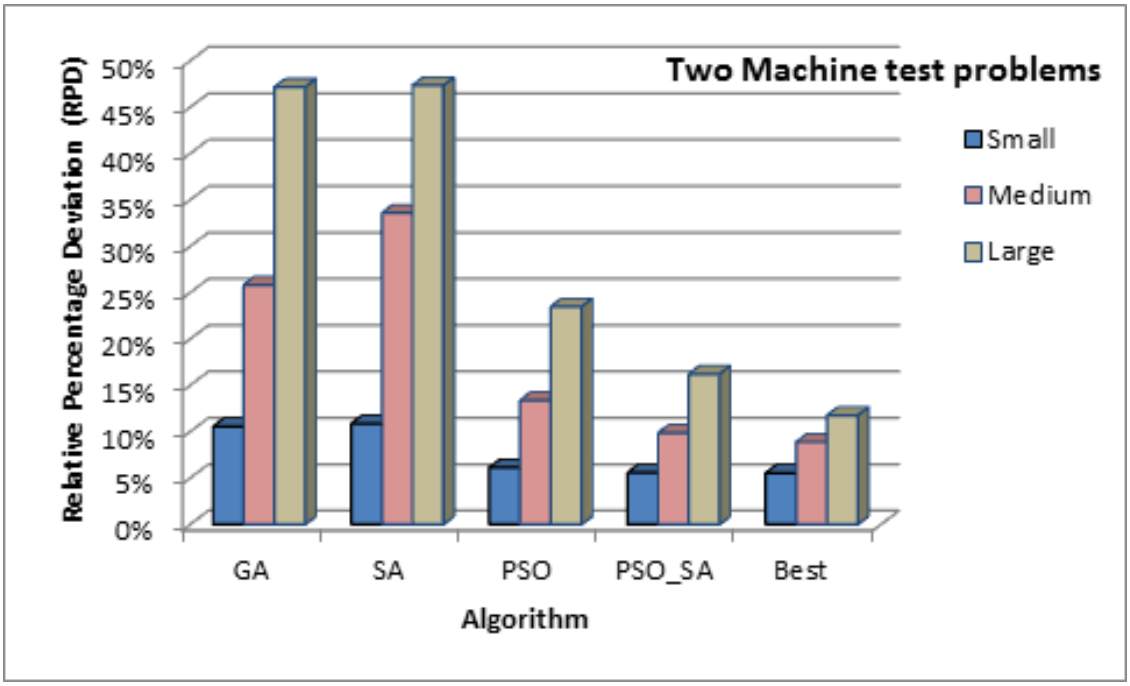

Figure 4.14: RPD for the two machine test problems

The reason that PSO-SA is not as efficient in the large instance is due to the complexity of the search space and the high possibility to be trapped in local optima. Yet, PSO-SA is the best algorithm among the proposed metaheuristics based on the RPD comparison.

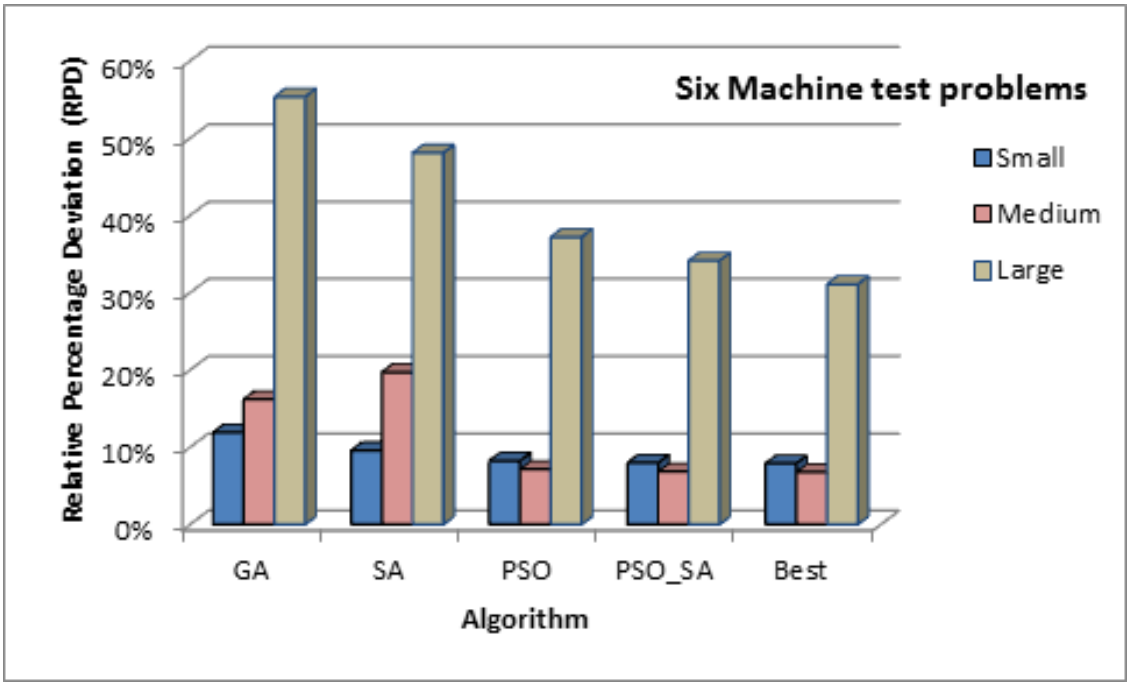

Figure 4.15: RPD for the six-machine test problems

4.9.2 Robustness of metaheuristics

As has been mentioned in Section 2.7, robustness means the insensitivity against small deviations in the input instances or parameters of a metaheuristic. In this section, the performance of the proposed metaheuristics is compared based on robustness. The analysis of metaheuristic is conducted by means of Design of Experiment (DOE) to get a better understanding of the performance of the proposed algorithms against problem size or structure. The comparison is conducted to answer the following questions:

1. Which algorithm is more robust?

2. What are the factors affecting the solution quality?

An experimental design is used to answer these questions and justify which algorithm outperforms the others. The factors to be studied are: Number of machines, number of families,

and the maximum number of jobs in a family. The levels corresponding to each parameter are presented in Table 4.28. The analysis is performed at 5 % significance level using the *Minitab* software as shown in Table 4.29.

Table 4.28: The levels of the problem's factors

Parameter	Candidate values (levels: 1, 2, 3)
$InitialTemp(T_0)$	2-4, 5-7, 8-10
$FinalTemp(T_f)$	2-6 , 6-10, 11-16
$No.of Machines$	2, −, 6

Table 4.29: The experimental design layout and results

Algorithm	No. of Families	Max. Jobs In Fam.	No. Machine	RPD/100	Algorithm	No. of Families	Max. Jobs In Fam.	No. Machine	RPD/100
GA	L1(2-5)	L1(2-4)	2	0.052	GA	L1(2-5)	L1(2-4)	6	0.069
GA	L1(2-5)	L1(2-4)	2	0.036	GA	L1(2-5)	L1(2-4)	6	0.03
GA	L2(6-10)	L1(2-4)	2	0.213	GA	L2(6-10)	L1(2-4)	6	0.375
GA	L2(6-10)	L2(5-7)	2	0.377	GA	L2(6-10)	L2(5-7)	6	0.308
GA	L2(6-10)	L2(5-7)	2	0.283	GA	L2(6-10)	L2(5-7)	6	0.07
GA	L3(11-16)	L2(5-7)	2	0.244	GA	L3(11-16)	L2(5-7)	6	0.154
GA	L3(11-16)	L3(8-10	2	0.543	GA	L3(11-16)	L3(8-10	6	0.119
GA	L3(11-16)	L3(8-10)	2	0.401	GA	L3(11-16)	L3(8-10)	6	0.988
SA	L1(2-5)	L1(2-4)	2	0.038	SA	L1(2-5)	L1(2-4)	6	0.065
SA	L1(2-5)	L1(2-4)	2	0.037	SA	L1(2-5)	L1(2-4)	6	0.018
SA	L2(6-10)	L1(2-4)	2	0.182	SA	L2(6-10)	L1(2-4)	6	0.3
SA	L2(6-10)	L2(5-7)	2	0.504	SA	L2(6-10)	L2(5-7)	6	0.298
SA	L2(6-10)	L2(5-7)	2	0.324	SA	L2(6-10)	L2(5-7)	6	0.149
SA	L3(11-16)	L2(5-7)	2	0.303	SA	L3(11-16)	L2(5-7)	6	0.167
SA	L3(11-16)	L3(8-10	2	0.572	SA	L3(11-16)	L3(8-10	6	0.121
SA	L3(11-16)	L3(8-10)	2	0.375	SA	L3(11-16)	L3(8-10)	6	0.841
PSO	L1(2-5)	L1(2-4)	2	0.023	PSO	L1(2-5)	L1(2-4)	6	0.058
PSO	L1(2-5)	L1(2-4)	2	0.007	PSO	L1(2-5)	L1(2-4)	6	0.01
PSO	L2(6-10)	L1(2-4)	2	0.133	PSO	L2(6-10)	L1(2-4)	6	0.262
PSO	L2(6-10)	L2(5-7)	2	0.244	PSO	L2(6-10)	L2(5-7)	6	0.235
PSO	L2(6-10)	L2(5-7)	2	0.161	PSO	L2(6-10)	L2(5-7)	6	0
PSO	L3(11-16)	L2(5-7)	2	0.032	PSO	L3(11-16)	L2(5-7)	6	0.004
PSO	L3(11-16)	L3(8-10	2	0.229	PSO	L3(11-16)	L3(8-10	6	0.024
PSO	L3(11-16)	L3(8-10)	2	0.241	PSO	L3(11-16)	L3(8-10)	6	0.719
PSO-SA	L1(2-5)	L1(2-4)	2	0.023	PSO-SA	L1(2-5)	L1(2-4)	6	0.058
PSO-SA	L1(2-5)	L1(2-4)	2	0.002	PSO-SA	L1(2-5)	L1(2-4)	6	0.009
PSO-SA	L2(6-10)	L1(2-4)	2	0.116	PSO-SA	L2(6-10)	L1(2-4)	6	0.252
PSO-SA	L2(6-10)	L2(5-7)	2	0.176	PSO-SA	L2(6-10)	L2(5-7)	6	0.231
PSO-SA	L2(6-10)	L2(5-7)	2	0.106	PSO-SA	L2(6-10)	L2(5-7)	6	0
PSO-SA	L3(11-16)	L2(5-7)	2	0.024	PSO-SA	L3(11-16)	L2(5-7)	6	0.001
PSO-SA	L3(11-16)	L3(8-10	2	0.161	PSO-SA	L3(11-16)	L3(8-10	6	0.02
PSO-SA	L3(11-16)	L3(8-10)	2	0.164	PSO-SA	L3(11-16)	L3(8-10)	6	0.663

Figure 4.16 shows the performance of all proposed metaheuristics averaged over all types of problem.

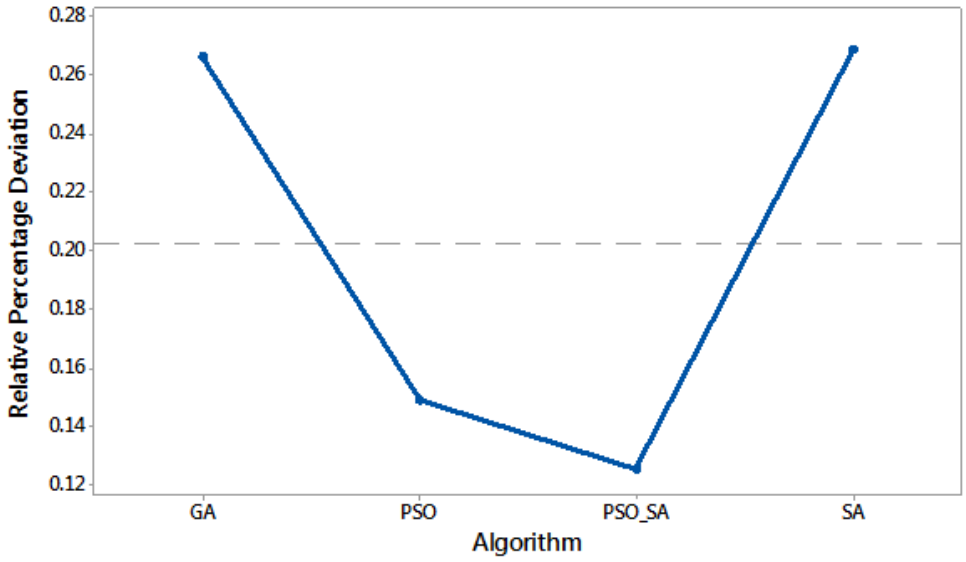

Figure 4.16: The performance of the proposed metaheuristic algorithms

Results and discussion

1. **Main effects and Interactions**

   The main effects of parameters, and interactions between these parameters are presented in Table 4.28. As shown in Figure 4.17, the number of machines in a cell is insignificant as compared to the number of families, and maximum number of jobs in a family. Note that Figure 4.17 indicates averaging performance of the 4 algorithms proposed for the tested instances.

   Moreover, studying the interaction among the factors gives a better understanding of the effect of several simultaneous factors. Figure 4.18 compares the RPD of the proposed algorithms as a function of family size. PSO-SA is better than PSO even though they have the same performance at the small-size problems (with less than five families). PSO-SA is noticeably better than SA and GA. It should also be noted

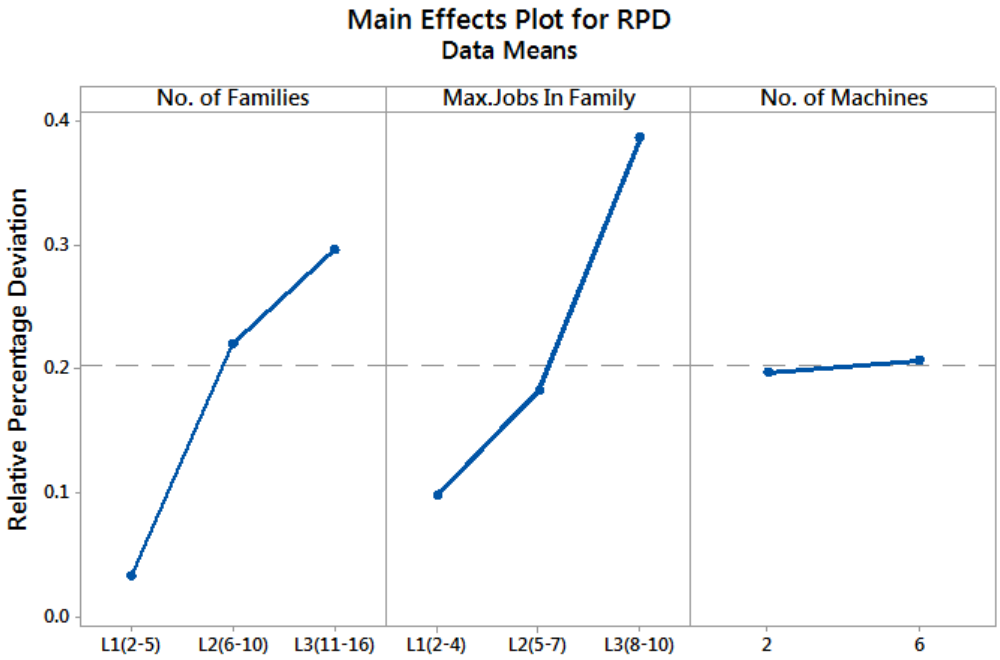

Figure 4.17: The performance of the proposed metaheuristic algorithms

that the metaheuristic algorithm based on PSO-SA is more efficient than the other proposed algorithms.

Figure 4.19 compares the RPD of the proposed algorithms as a function of the maximum number of jobs. It shows that the 4 algorithms have the same performance at the lower level of jobs in a family (maximum number of jobs in a family is less than four jobs). However, as far as the number of maximum jobs in a family is increased (greater than 5 jobs in a family), there is a significant difference in the performance and PSO-SA is superior.

2. **Interval plot**

In real-world applications, it is often desirable that a solution be not only of high performance, but also of high robustness that reflects the ability of the developed

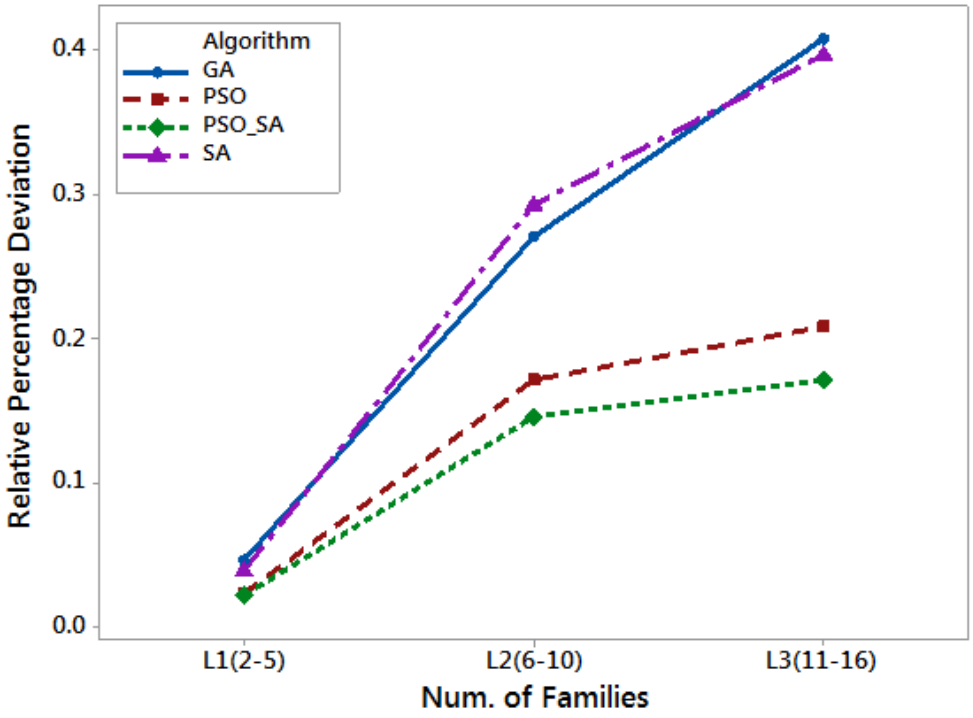

Figure 4.18: Interaction plots of number of familes vs. the proposed algorithms

algorithm to solve a wide range of test problems with minimal variation. Figure 4.20 shows that the minimal variation is obtained by the PSO-SA algorithm. The greater variation in the results of the all test problems is produced by the SA algorithm. Furthermore, for two-machine problems, the variation in RPD reaches the maximum value for SA algorithm, as shown in Figure 4.21 PSO-SA gives some solutions that are better than solutions that are obtained by PSO and GA.

In addition, there is considerable difference in the variation of RPD between the two metaheuristics for the six-machine problems as shown in Figure 4.22, indicating that metaheuristics based on PSO performs better than GA and SA. Therefore, PSO-SA is more efficient than GA in solving the cellular flowshop to minimize the TFT.

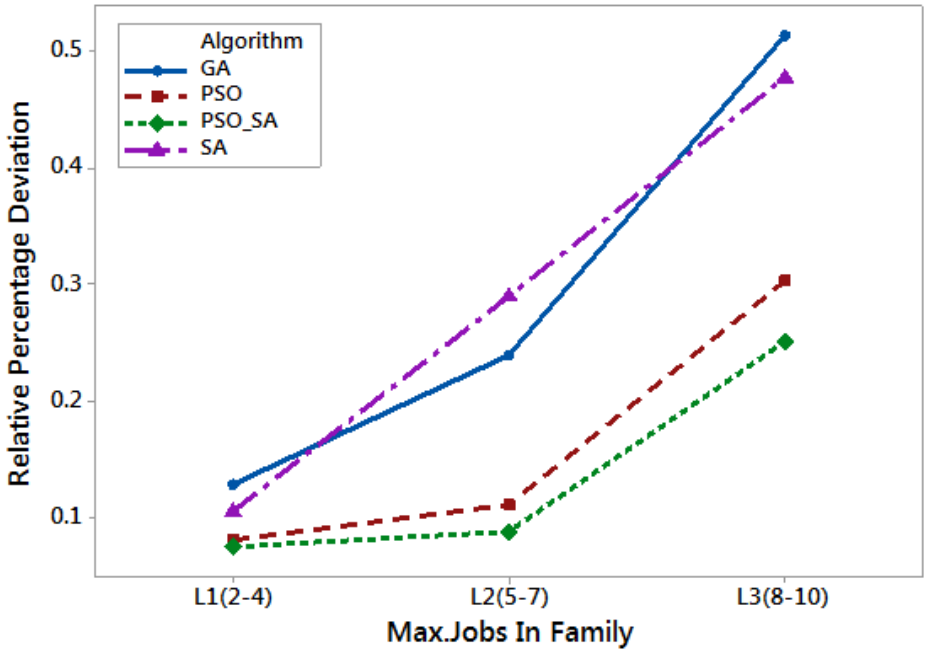

Figure 4.19: Interaction plots of maximum jobs in family vs. the proposed algorithms

3. **Analysis of Variance (ANOVA)**

   Furthermore, based on the P-values, analysis of variance (ANOVA) shows that the algorithm used to solve test problems has a significant effect on the model since its P-value is close to zero.

   The most significant factor is the maximum number of jobs in a family. Yet, as shown in Table 4.30, most other factors are insignificant since their values are greater than 0.005.

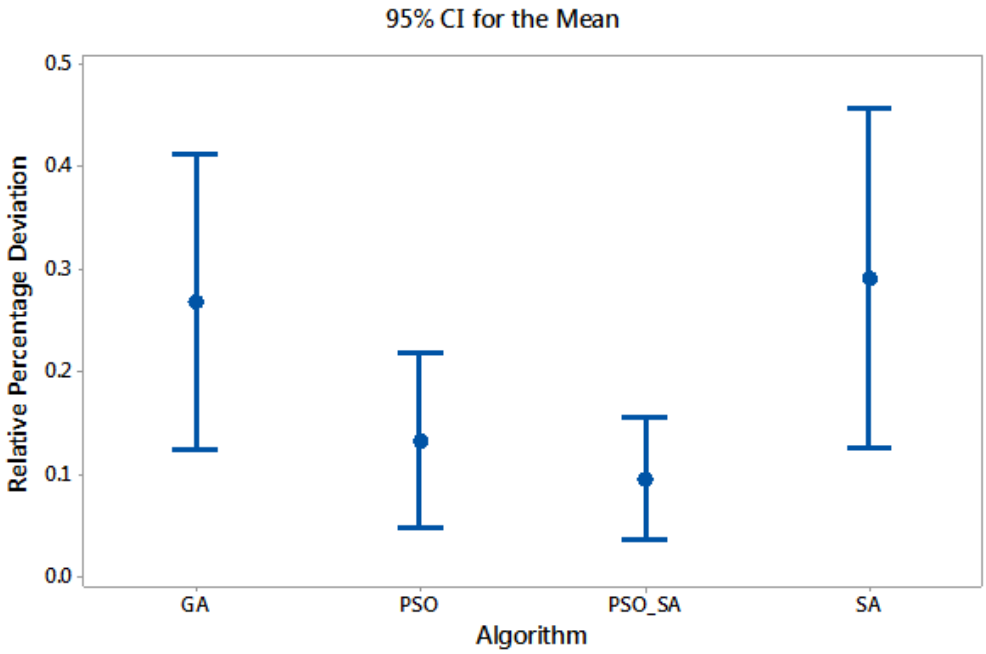

Figure 4.20: Interval plot of RPD of the all test problems

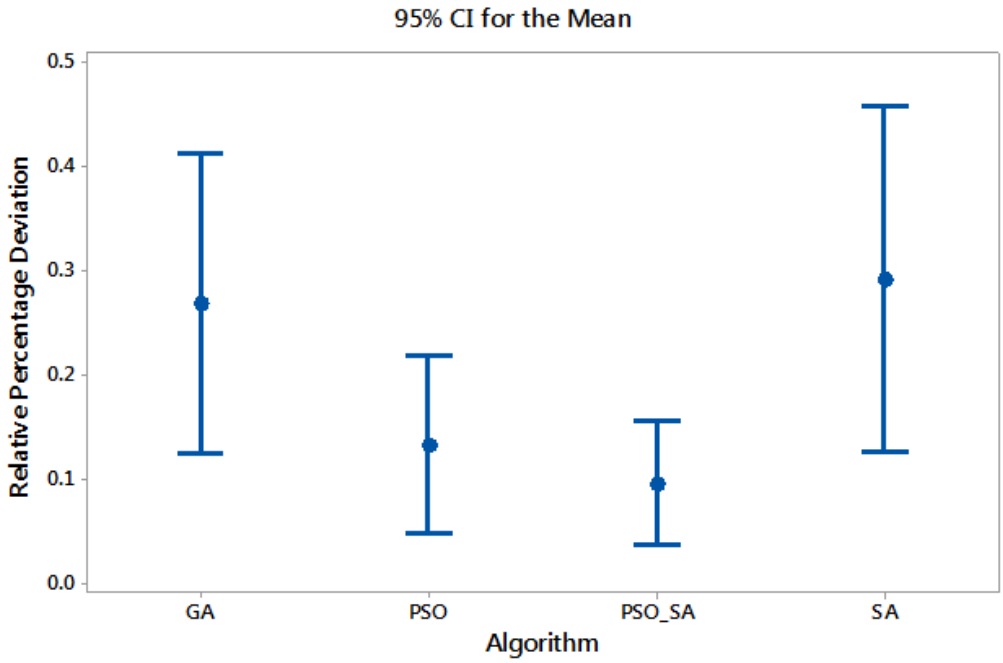

Figure 4.21: Interval plot for two-machine problems

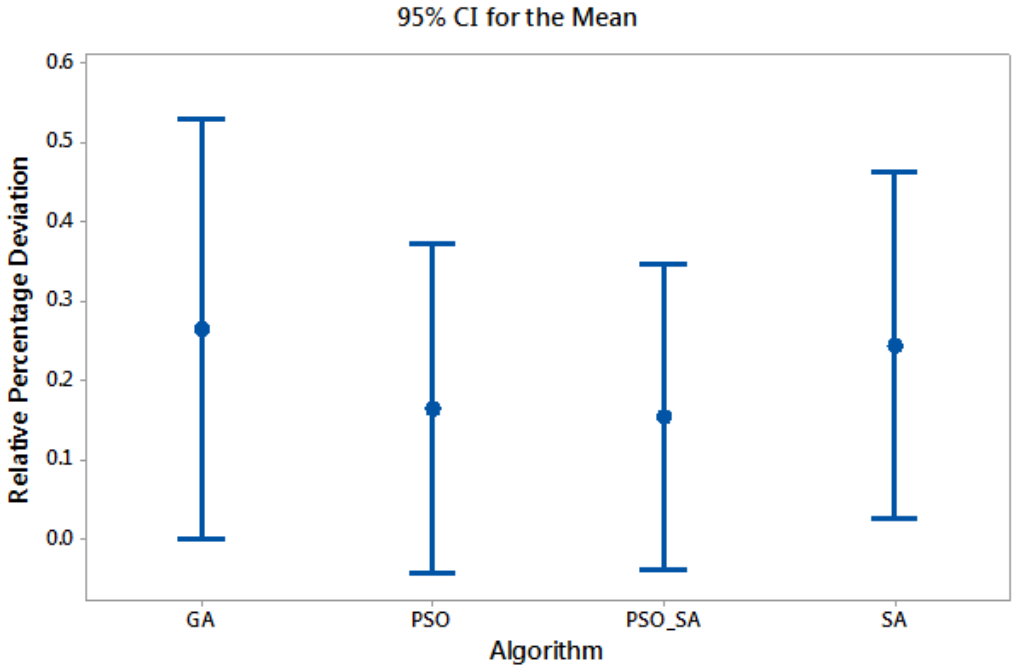

Figure 4.22: Interval plot for six-machine problems

Table 4.30: The levels of the problem's factors

Source	DF	Adj. Sum of Squares	Adj. Mean Square	F-Value	P-Value
Algorithm	3	0.27630	0.092100	2.96	0.040
Machines	1	0.00139	0.001387	0.04	0.834
No. of Families	2	0.20152	0.100761	3.24	0.047
Setup Time	2	0.83501	0.417505	13.41	0.000
Error	55	1.71298	0.031145		
Total	63	2.86893			
Model Summary	S	R-sq	R-sq(adj)	R-sq(pred)	
	0.176480	40.29 %	31.61 %	19.80 %	

4.10 Summary and conclusions

This chapter presents several metaheuristics based on PSO, SA and GA to solve a FM-CSP with SDSTs to minimize the total flow time. The Taguchi method is implemented to identify the best levels of the SA algorithm and investigate the main effect of each parameter. The performance of the proposed algorithms is evaluated according to the best available algorithm based on RPD from lower bounds. Moreover, robustness of proposed algorithms is

considered using Design of Experiments (DOE) technique to find the most robust algorithm. It was found that the proposed PSO-SA algorithm is more robust than SA, GA, and PSO as seen in the problems examined in this study.

Chapter 5

Metaheuristics for Solving the Bi-criteria Scheduling Problem of Flowshop Manufacturing cell with Sequence Dependent Setup Times

5.1 Introduction

Unlike single objective optimization problems, Multiple Objective Optimization problems (MOPs) involve several objectives and therefore, the search space is more complicated. Instead of a single optimal solution for a single objective optimization problem, there is a set of solutions, generally known as Pareto-optimal solutions. In the absence of preference information (selecting the most desirable objective), these solutions are optimal in the wider

sense that no other solutions in the search space are superior to them when all objectives are considered. Section 5.2 presents principles of multi-objective optimization problem. This is followed by a brief discussion about the traditional methods for solving MOPs, and Evolutionary Algorithms (EAs) to generate an approximated Pareto-optimal solution or Pareto Front (PF). Then, the proposed algorithms based on Multiple Particle Swarm Optimization (MPSO), Multi-objective Simulated Annealing (MOSA), and the Improved Multiple Particle Swarm Optimization (IMPSO) to minimize TFT and MS as bi-objective problem will be discussed in detail. The results of the developed algorithms are compared based on several Quality Indicators (QI) measures. Finally, summary, and conclusion remarks are addressed.

5.2 Principles of Multi-objective optimization problem

MOP is defined as a vector of decision variables which satisfies constraints within a feasible region to optimize a vector of objective functions [95]. Unlike mono objective, there is a set of solutions called Pareto solutions or Pareto front which is found through the use of Pareto Optimality Theory [96]. A general multi-objective optimization problem includes a set of n parameters (decision variables), a set of M objectives, and a set of k constraints. Objective functions and constraints are functions of the decision variables. In general, the minimization problem of m objectives can be presented as follows:

$$\begin{aligned} &\text{Minimize} \quad f_m(x), \quad m = 1, \ldots, M. \\ &\text{subject to} \\ &\qquad g_j(x) \leq 0, \ j = 1, \ldots, k. \\ &\qquad h_l(x) \leq 0, \ l = 1, \ldots, p. \end{aligned} \tag{5.1}$$

$f_i(x)$is the ith objective function; $g_j(x)$ and $h_l(x)$ are inequality constraint and equality constraints respectively.

5.2.1 Pareto Dominance

The optimal solutions in multi-objective optimization problem can be defined based on the concept of domination. This section addresses the definition of the non-dominated set or Pareto front. A solution $x^{[1]}$ is said to dominate other solution $x^{[2]}$; if both of the following conditions are true:

1. The solution $x^{[1]}$ is no worse than $x^{[2]}$ in all objectives. Thus, the solutions are compared based on their objective function values.

2. The solution $x^{[1]}$ is strictly better than $x^{[2]}$ in at least one objective.

$$f_i(x) \leq f_i(y) \quad \forall i \in 1, \ldots, M. \tag{5.2}$$

and

$$\exists \quad i \in \{1, \ldots, M\} : \ f_i(x) < f_i(y) \tag{5.3}$$

All decision vectors that are not dominated by any other feasible decision vector are called non-dominated sets or front (Pareto-optimal set). Achieving the exact Pareto front of an arbitrary problem is usually quite difficult. Nevertheless, reasonably good approximations of true Pareto Front are generally acceptable within a limited computational time [96]. Thus, the ultimate goal for multi-objective optimization problem is to find a set of solutions which lie on the Pareto front and find a set of solutions which are diverse enough to represent the entire range of the Pareto front. There are several solution techniques for solving MOPs which are classified into Classical methods (CMs), and Evolutionary Algorithms (EAs). CM is basically converting the MOP to a single objective problem such as weight method, constraint method, goal programming. On the other hand, EA is established as an alternative to the CM due to its ability to find multiple optimal solutions in one single simulation run [97]. Most of the recent metaheuristics focus on population algorithms that are considered to be more suited to solve MOPs [96]. The following sections provide a brief discussion about CMs for solving MOPs.

5.3 Classical Methods

Classical methods for generating the Pareto-optimal set combine the objectives into a single, parameterized objective function. However, the parameters of this function are not set by the decision makers, but systematically varied by the optimizer. Several optimization runs with different parameter settings are performed in order to achieve a set of solutions which approximates the Pareto-optimal set.

5.3.1 *Weighting method*

In this method, the original multi-objective optimization problem (M objective) is converted to a single objective optimization problem by forming a linear combination of the objectives. A weight w_i is given to each objective and the new objective function will be as follows:

$$\begin{aligned} &\text{Manimize} \quad f(x) = \sum_{m=1}^{M} w_m f_m(x), \qquad w_m \in [0,1] \\ &\text{subject to} \\ &\qquad\qquad g_j(x) \leq 0, \qquad j = 1, \ldots, J \\ &\qquad\qquad h_k(x) = 0, \qquad k = 1, \ldots, K \\ &\qquad\qquad x_i^{(l)} \leq x_i \leq x_i^{(u)} \qquad i = 1, \ldots, n \end{aligned} \tag{5.4}$$

Where w_i are called weights, and $\sum_{i=1}^{m} w_i$. Solving of the equation 5.4 with different weights produces a set of solutions [98]. These solutions are considered as a Pareto set of the problem.

5.3.2 *Constraint Method*

In this method, MOP is reformulated by keeping one of the objectives and restricted the rest of the objectives [98]. $M-1$ out of M objectives can be transformed into constraints. The remaining objective is considered as the objective function to be optimized.

Manimize $f_\mu(x)$,

subject to

$$\begin{aligned} f_\mu(x) \geq \epsilon_m \quad & m = 1, \ldots, M, \text{and} \quad m \neq \mu; \\ g_j(x) \leq 0, \quad & j = 1, \ldots, J \\ h_k(x) = 0, \quad & k = 1, \ldots, K \\ x_i^{(l)} \leq x_i \leq x_i^{(u)} \quad & i = 1, \ldots, n \end{aligned} \tag{5.5}$$

The lower bounds values ϵ_i are given different combinations to find multiple Pareto solutions. Some difficulties in defining the weights and other parameters affect the efficiency of the Weighting and Constraint method. The main disadvantage of the classical approaches is that they cannot generate all Pareto-optimal solutions with non-convex trade-off surfaces [97]. A number of difficulties in the classical methods in finding multiple Pareto solutions are reported [98] as follows:

- Only one Pareto solution can be found in one simulation run
- Not all Pareto solutions can be found in nonconvex MOPs
- Problem knowledge such as weights is required

5.4 Evolutionary Algorithms

Evolutionary multi-objective optimization was proposed by Schaffer (1985) for the first time [99]. Later, a few different EAs are proposed by Kursawe [100], Fonseca and Fleming [101]; Horn, et al [102], and Srinivas, et al [103], and Zitzler, et al [104]. The main advantage

of using evolutionary algorithms to search in complex solution spaces is the flexibility in combination with robust performance and global search characteristics [97]. Evolutionary Algorithms (EAs) are becoming an alternative technique to classical methods due to the following reasons: 1) EAs are relatively simple to implement, and 2) EAs are flexible and have wide-spread applicability [98]. Moreover, EAs are considered as a method for exploring the Pareto-optimal front in multi-objective optimization problems that are too complex to be solved by exact methods [105]. Therefore, the complexity of solution spaces can be handled, and multiple alternative trade-offs can be generated in a single optimization run. Evolutionary multi-objective approaches are based on evolutionary algorithms, where a population of solutions is used and evolved using the principles of natural evolution to obtain a set of solutions as close as possible to Pareto optimal solutions [106]. In recent years, many PSO algorithms have been proposed for solving multi-objective optimization problems, and results show that PSO is very suitable for multi-objective problems [107]. Pan proposed a novel Multi-Objective Particle Swarm Optimization (MOPSO) algorithm for solving no-wait flow shop scheduling problems with makespan and maximum tardiness criteria. The job permutation is represented by the particles; the dominance is applied to compare different solutions [108]. Further, Pan combined a local search algorithm to the MOPSO algorithm to balance between global exploration and local exploitation. Moreover, MOPSO was proposed by Chandrasekaran, et al to solve the flowshop scheduling problem to search a set of non-dominated sets close to the Pareto front [109]. Leong et al proposed the integration of a dynamic population strategy based on multiple particle swarm optimization (MOPSO) algorithm [110]. They reported that an adaptive local archive is developed to improve the

diversity within each swarm. Durillo et al analyzed the search capabilities of six typical MOPSO algorithms using three different quality indicators to find the best algorithm [111]. Tang et al presented an improved particle swarm optimization algorithm for the hybrid flowshop scheduling problem to minimize total weighted completion time [112]. Wang et al proposed PSO based hybrid multi-objective algorithm; they implemented crossover operator of genetic algorithm as the particle updating strategy. Moreover, a local search heuristic based on scatter search was applied to improve the non-dominated solutions [113]. Tasgetiren et al proposed MOPSO algorithm to solve the Permutation Flowshop Sequencing Problem (PFSP) (FCMSP without any setup time, and part family) with the objective of minimizing makespan and the total flow time of jobs [114].

5.5 Proposed Algorithms

In fact, two major challenges must be addressed when an evolutionary algorithm is applied to multi-objective optimization: First challenge is to accomplish fitness assignment and selection to guide the search towards the Pareto front. The second challenge is to maintain a diverse population in order to prevent premature convergence and achieve a well distributed and well spread nondominated set [105]. Metaheuristics based on MPSO, MOSA, and IMPSO are proposed to generate the approximated Pareto front sets. First, the MPSO algorithm is developed to minimize the total flow time and makespan in a cellular flowshop scheduling problem. Then, a Multi-Objective simulated annealing algorithm is developed to solve the same problem. Lastly, a hybrid algorithm based on an Improved Multi-objective Particle Swarm Optimization (named IMPSO-TA) is proposed. In this algorithm, IMPSO

proposed by [1] is combined with (TA) algorithm proposed by [52] as local search engine to provide efficient Pareto fronts. The analysis of the performance of the proposed MPSO, MOSA, and IMPSO algorithms will be discussed in detail. Moreover, QI measures are used to evaluate the effectiveness of the proposed algorithms in Section 5.10.

5.6 Multi-objective Particle swarm optimization Algorithm

In this study, metaheuristics based on MPSO, IMPSO, and MOSA are proposed to generate the approximated Pareto front sets. In the area of multi-objective optimization problems, PSO is attracting much research interest because of the simple computational model introduced by PSO and exploration capabilities of the algorithm [115].

As has been mentioned in Chapter 4, the neighbourhood search in PSO is that each particle flies with guidance of best particle p-bests and the global best g-bests. Updating these values is very critical and it has a significant effect on the solution quality. In PSO algorithms for mono objective optimizations, determination of local and global best particles is based on the extreme value of the objective function. However, in multi-objective optimizations problems, each particle might have a set of different g-bests from which just one can be selected in order to update its position. This set of g-bests is usually stored in a different place from the swarm that is called external archive named as $\{NDS\}$ set. This is an externally stored archive or elite set that was used to store the non-dominated solutions found so far. The set of solutions contained in the external archive. Global best (g-best) particle is randomly selected from an external archive. Velocities and Positions of particles are updated each iteration. Solutions are updated according to the non-domination check (Section 5.2.1), and

Algorithm 5.1 The steps of the proposed MPSO algorithm

Require: Initialize parameters { swarm size n, maximum Iteration I_{max}, C_{1max}, C_{1min}, $C_{2max}, C_{2min}, V_{max}, V_{min}, W_{max}, W_{min}, X_{max}, X_{min}$}

- Step 1: Set iteration $t = 0$
- Step 2: If $t = 0$ Generate initial positions X_i^t and V_i^t initial velocities for $i \in \{1, 2, ..., n\}$ according to equations (5.6, and 5.7)
  Else
  Generate a new swarm by updating the velocity V_i^t and position X_i^t of particles according to equations (5.8, and 5.9)
- Step 3: Set the initial Non Dominated Set $NDS = \{\}$ (The Null Set)
- Step 4: Apply the (ROV) on X_i^t to find the sequence of families as well as jobs in families.
- Step 5: Calculate the objective function f_{TFT} (X_i^t) and f_{MS} (X_i^t) for each particle $i \in \{1, 2, ..., n\}$
- Step 6: Check whether the particle (X_i^t) qualifies to be included in NDS based on the discussion in Section 5.2.1
  - o Step 6.1 For each particle in the swarm the p-best (P_i^t) is calculated as:

$$P_i^t = \underset{X_i^j}{\operatorname{argmin}} f(X_i^j) \qquad for \quad j \in \{1, 2, ..., t\}$$

$$f_i^t = \left(0.5 f_{TFT}(X_i^t) + 0.5 f_{MS}(X_i^t)\right)$$

  - o Step 6.2 The g-best G^t is randomly selected from the NDS set.
- Step 7 Set $t = t + 1$
- Step 8 If $t = I_{max}$, STOP; and set $NDS^{I_{max}}$ as the Pareto front; else, go to step 2.

stored in the external archive . The final set of non-dominated solutions in the external archive is reported as the final output of the algorithm.

5.6.1 Initialization

The MPSO algorithm starts by randomly generating several particles $X_0 = \{x_1, ..., x_N\}$ (subscript 0 refers to initial state before starting iterations) where N is the swarm size. Each particle is considered as a potential solution that refers to the sequence of the families and jobs in each family. Initial position and velocity of particles are determined using the following equations:

$$X_{0i} = X_{min} + C_1(X_{max} - X_{min}), \tag{5.6}$$

$$V_{0i} = V_{min} + C_2(V_{max} - V_{min}), \tag{5.7}$$

where $i \in \{1, 2, ...N\}$, X_{min} and X_{max} represent the minimum and maximum position values respectively. V_{min} and V_{max} represent the minimum and maximum velocities respectively. C_1 and C_2 are random variables in the range $[0, 1]$. The values of these parameters are given in Table 4.7.

5.6.2 Pareto Search

After initializing the first swarm, an External Archive is defined to store the Non-dominated Solution Set (NDS) or the Pareto front. This archive (named as NDS-Set) will be filled by initial non-dominated solution obtained from the initial swarm. In the tth iteration ($t > 0$), the velocity and the position are updated based on the following equations:

$$v_j^{t+1} = wv_j^t + C_1 r_1^*(P_j^t - x_j(t)) + C_2 r_2^*(G^t - x_j(t)) \tag{5.8}$$

$$x_j^{t+1} = v_j^{t+1} + x_j^t \tag{5.9}$$

Where w is the inertia weight that controls the influence of the previous velocity of particles, r_1 and r_2 are uniformly distributed random variables in the range $[0, 1]$, C_1 and C_2 are the acceleration constants. As has been mentioned in Section 4.5, the values of these constants are updated according to the equations 4.4, 4.5, and 4.6. Unlike the standard PSO for single objective problem, g-best (G^t) at the tth iteration is selected randomly from NDS. They guide every particle toward the local best and the global best solutions during the search process. The Ranked Order Value (ROV) [83] is applied to find the sequence of the families as well as the jobs in each family. This procedure will be repeated till the maximum number of iterations is reached. The final NDS archive will be reported as a final solution of the Pareto front. The algorithm is outlined in Algorithm 5.1. Note that the p-best, and g-best are calculated to update the velocity of the particles in the swarm for the next iteration.

5.6.3 Computational results

The proposed MPSO algorithm is coded using C++ language and run on a PC with an Intel core I7 (2.93 GHz) CPU and 4.0 GB memory. The performance of the proposed algorithm has been evaluated using test problems used by Schaller (Section 3.4). The computational results of Minimum, Average, and Maximum TFT and MS are summarised in Tables 5.1, 5.2, 5.3 for small problems, and in Table 5.4 for large problems with small, medium, and large setup times.

Table 5.1: Results of the proposed MPSO for small setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	66	66.3	67	172	174.3	175	66	3.1	0.00%
	3	3	9	86	86	86	485	487	491	86	4.19	0.00%
	3	3	9	100	102.3	104	527	532	544	100	4.13	0.00%
	3	3	5	75	82.5	86	249	261.5	275	75	3.23	0.00%
	3	3	13	117	117.7	118	872	876.7	886	117	4.53	0.00%
	3	3	12	124	124.7	125	831	834.7	842	124	4.9	0.00%
	3	3	15	125	126.3	127	1108	1108.7	1110	122	4.46	2.46%
	3	3	15	125	131.4	138	1145	1168.1	1213	125	4.86	0.00%
	3	4	9	102	104.75	106	580	595.5	610	101	4.22	0.99%
	3	4	15	154	154.3	155	1383	1384.5	1387	151	5.87	1.99%
	3	4	13	135	136.4	139	1073	1080.8	1108	130	5.41	3.85%
	3	4	13	126	128.2	130	1047	1071.2	1100	123	5.36	2.44%
	3	4	9	116	116	116	674	674	674	115	4.61	0.87%
Small	3	4	7	96	96.7	98	402	403	405	96	4.23	0.00%
	3	4	14	128	130	133	1108	1153.4	1180	128	5.83	0.00%
	3	4	14	120	120.7	121	1103	1109.7	1123	118	5.81	1.69%
	3	4	15	140	146.8	150	1306	1320.2	1334	140	6.41	0.00%
	3	4	7	80	84.5	88	403	407.8	410	80	4.26	0.00%
	3	4	14	131	131.5	132	1056	1128	1187	129	5.63	1.55%
	3	4	12	111	113.7	118	778	857.8	902	109	5.08	1.83%
	4	4	8	104	106.3	111	563	564	566	104	5.71	0.00%
	4	4	14	139	139	139	1105	1105	1105	138	7.31	0.72%
	4	4	13	132	134.3	140	984	1111.8	1156	132	7.71	0.00%
	5	5	15	154	155.4	157	1343	1395	1459	153	10.35	0.65%
	5	5	15	154	158	166	1290	1366	1460	149	11.21	3.36%
	5	6	14	158	168.3	177	1396	1411.4	1422	154	13.4	2.60%
	6	5	15	170	179.6	186	1563	1607	1682	164	13.42	3.66%
										Ave :	6.12	1.06%
										max	13.42	3.85%

Table 5.2: Results of the proposed MPSO for medium setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	116	116	116	329	329	329	116	3.2	0.0%
	3	3	9	150	158.7	176	882	1010	1074	149	4.23	0.7%
	3	3	9	152	159.4	164	756	824.3	934	152	4.18	0.00%
	3	3	5	117	117	117	405	405	405	114	3.24	2.60%
	3	3	13	177	180.3	182	1073	1098	1148	177	4.49	0.00%
	3	3	12	200	203.6	207	1368	1378	1391	199	4.85	0.50%
	3	3	15	161	164.8	171	1467	1478.7	1498	161	4.46	0.00%
	3	3	15	164	190.1	204	1422	1478.5	1568	164	4.94	0.00%
	3	4	9	150	150.7	151	755	761.33	774	150	4.6	0.00%
	3	4	15	211	215.6	221	1921	2048.2	2142	207	5.72	1.90%
	3	4	13	192	201.3	207	1633	1667.9	1731	192	5.4	0.00%
	3	4	13	187	190.2	192	1561	1573.4	1615	186	5.47	0.50%
Medium	3	4	9	178	179.8	182	1118	1122	1125	174	4.65	2.30%
	3	4	7	156	156	156	633	633	633	156	4.2	0.00%
	3	4	14	179	179.3	180	1402	1555.3	1632	179	5.88	0.00%
	3	4	14	187	200	219	1730	1738.5	1747	186	5.87	0.50%
	3	4	15	172	172	172	1643	1645.7	1647	172	6.41	0.00%
	3	4	7	119	119	119	593	594	596	119	4.21	0.00%
	3	4	14	199	202.6	212	1707	1833	1880	199	5.63	0.00%
	3	4	12	161	161	161	1335	1335.3	1336	160	5.07	0.60%
	4	4	8	160	184.8	204	851	859.8	888	160	5.79	0.00%
	4	4	14	196	196.7	198	1459	1465.7	1469	196	7.52	0.00%
	4	4	13	204	208.3	219	1360	1392.7	1432	199	7.61	2.50%
	5	5	15	233	245.5	254	1926	2088	2242	233	10.76	0.00%
	5	5	15	211	211	211	1857	1857.7	1858	211	11.24	0.00%
	5	6	14	254	265.4	287	2238	2354.2	2489	254	13.12	0.00%
	6	5	15	256	267.4	271	2369	2378.2	2395	251	13.91	2.00%
										Ave :	6.17	0.53 %
										max	13.91	2.63 %

Table 5.3: Results of the proposed MPSO for large setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	162	162	162	439	439	439	162	3.21	0.0 %
	3	3	4	162	162	162	439	439	439	162	3.21	0.00%
	3	3	9	263	274.3	297	1473	1473.7	1474	263	4.23	0.00%
	3	3	9	257	263.4	273	1336	1543.6	1683	257	4.34	0.00%
	3	3	5	254	269.3	311	900	926.8	936	254	3.32	0.00%
	3	3	13	278	281.3	289	2076	2382	2486	278	4.61	0.00%
	3	3	12	251	285	323	1855	2170	2371	251	4.96	0.00%
	3	3	15	253	253	253	2477	2477	2477	253	4.49	0.00%
	3	3	15	269	273	281	2392	2507.3	2565	266	4.86	1.10%
	3	4	9	237	241.8	244	1318	1360.5	1480	236	4.79	0.40%
	3	4	15	290	306	342	2810	2868.8	2933	290	5.78	0.00%
	3	4	13	268	274	280	2057	2089.6	2124	266	5.45	0.80%
	3	4	13	263	263.7	264	1830	1831.3	1834	261	5.66	0.80%
	3	4	9	264	269.5	281	1503	1790.7	1851	264	4.74	0.00%
	3	4	7	251	251.3	252	1002	1002.7	1003	251	4.31	0.00%
	3	4	14	285	285	285	2090	2091	2093	284	5.92	0.40%
Large	3	4	14	291	326	338	2084	2101	2149	291	5.91	0.00%
	3	4	15	286	286.7	287	2895	2897	2901	286	6.5	0.00%
	3	4	7	202	202.7	204	988	988.7	989	201	4.3	0.50%
	3	4	14	309	313.3	318	2524	2886.8	3157	309	5.77	0.00%
	3	4	12	249	254.3	257	1396	1530.7	1800	246	5.13	1.20%
	4	4	8	284	284	284	1351	1352	1354	284	5.86	0.00%
	4	4	14	325	331	344	2852	3189	3348	323	7.45	0.60%
	4	4	13	331	348.3	365	2289	2694.3	3121	331	7.72	0.00%
	5	5	15	395	409.8	422	3320	3366.3	3415	392	10.61	0.80%
	5	5	15	372	394.8	410	3266	3503.6	4281	372	11.5	0.00%
	5	6	14	421	439.8	460	2688	2944.3	3058	412	13.3	2.20%
	6	5	15	440	498.7	520	4044	4226.2	4909	436	13.82	0.90%
										Ave :	6.24	0.36%
										max	13.82	2.18 %

Table 5.4: Results of the proposed MPSO for Large setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			MakeSpan Lower Bound	CPU Time (sec)	Deviation for Makespan	
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max				
	3	3	17	144	144	144	1435	1436.3	1439	144	0.00%	4.957	
	4	4	21	185	188.4	192	2165	2240.6	2298	180	2.78%	8.096	
	5	5	22	279	281.5	283	5046	5083.5	5102	271	2.95%	13.96	
	6	5	19	201	205.8	209	2241	2302.3	2388	192	4.69%	13.65	
Small	5	6	39	307	320.5	333	6771	6855.3	6947	301	1.99%	17.37	
	6	6	39	341	342.6	345	7578	7665.6	7731	321	6.23%	20.72	
	8	8	37	376	386.9	394	7998	8291.9	8615	349	7.74%	34.54	
	10	10	59	539	544.8	557	18115	18395	18728	491	9.78%	66.89	*Ave.* = 4.52%
	3	3	17	208	214.2	217	2214	2229.8	2251	207	0.48%	5.19	
	4	4	21	270	281	286	3081	3134.4	3334	269	0.37%	8.24	
	5	5	22	355	372.1	387	6616	6830.6	7080	355	0.00%	14.54	
	6	5	19	291	301.2	311	3001	3112.2	3231	283	2.83%	13.7	
Medium	5	6	39	394	397.7	402	8202	8230	8269	378	4.23%	17.97	
	6	6	39	429	440.4	451	9348	9659.8	9890	416	3.13%	20.96	
	8	8	37	487	508	521	10240	10489	11362	465	4.73%	35.72	
	10	10	59	682	699.8	720	23158	23507	23877	647	5.41%	66.91	*Ave.* = 2.65%
	3	3	17	311	321	341	2987	3045	3074	311	0.00%	5.1	
	4	4	21	367	367	367	4620	4624.7	4634	366	0.27%	8.26	
	5	5	22	493	502.4	517	8662	9477.4	9835	492	0.20%	14.35	
	6	5	19	417	456	494	5005	5267.2	5619	410	1.71%	13.76	
Large	5	6	39	579	602.8	629	12831	13935	14865	539	7.42%	17.92	
	6	6	39	611	611.8	612	13766	13964	14535	584	4.62%	21.6	
	8	8	37	725	752.2	767	14010	14074	14133	696	4.17%	35.27	
	10	10	59	1043	1068.7	1095	33728	35639	38331	937	11.31%	67.21	*Ave.* = 3.71%
											Ave. =3.63%	*Ave.* = 22.79	

5.7 Multi-objective Simulated Annealing Algorithm

SA is often distinguished from the evolutionary algorithms that are used in solving the multi-objective optimization problems because SA does not involve candidate solutions [116]. Yet, MOSA has been proposed to solve the multi-objective optimization problems. For instance, the two-machine flowshop scheduling problem is addressed by Mansouri et al [117] to minimize setups and makespan. SA algorithm provides excellent solutions to single and multiple objective optimization problems with a substantial reduction in computation time [118]. Pareto SA was presented Czyzzak et al [119] to generate an approximation of the Pareto front for multiple-objective combinatorial optimization. Furthermore, the problem of permutation flowshop scheduling is considered by [120], in which they presented MOSA algorithm with the objectives of minimizing the makespan and total flow time of jobs.

The proposed MOSA mainly consists of the following steps: first, initialize the search parameter, and generate the initial Non-dominating Solutions (NDS) or the Elite set. Second, generate neighbourhood solutions. Lastly, implement the non-dominating check of the new solutions based on the discussion in Section 5.2.1, and update NDS set. The algorithm is given in Algorithm 5.2.

5.7.1 Initial Solutions

A number of initial solutions are randomly generated after setting the search parameters are selected based on the discussion presented in Section 4.7.2. The combination of the parameters were as follows: $I_{iter} = 150$, $T_0 = 20$, $T_f = 1$, α =0.95, $N_{non-imp.} = 15$. Cauchy

function is used in the annealing process that gives the SA more chances to escape from local minima. The non-dominated check is implemented to specify the initial Pareto set that would be updated based on the neighbourhood generation in the sequence of the families as well as the jobs in each family.

5.7.2 Neighbourhood Generation

Two common methods were used in generating the new solution: Swapping, and Insertion. For a given a sequence(π), the set of families and jobs in each family can be obtained from the current sequence π using the swapping and insertion as follows:

- **Swapping:** let j, k be two positions in the sequence π, which are selected randomly. For example, consider a neighbor of (π). Swapping is obtained by interchanging the jobs in the positions j and k. For instance if π = (5,2,3,4,1,6) and j = 2, k = 4, then a neighbor of (π) will be: (5,4,3,2,1,6).

- **Insertion:** let j and k be two positions in the sequence (π) which are selected randomly. A neighbor of (π) is obtained by inserting the job of position j to the position k pushing the jobs in between these positions backward (forward), including the job of position j, if k is greater (or less) than j. For example if π = (5,2,3,4,1,6) and $j = 2$, $k = 4$, then the neighbor of π will be: (5,3,4,2,1,6). If $j = 4$, k= 2, then the neighbor of π will be: (5,4,2,3,1,6).

For a given sequence of the families, and jobs in each family. A family f is selected randomly for swapping and insertion methods; the job sequence of the selected family is changed using

Algorithm 5.2 The steps of the proposed MOSA algorithm

Require: Initialize the search parameters $\{ T_0, T_F, \alpha = 0.95, I_{iter}, N_{non-imp.}\}$

- Step 1: Generate initial population (pop) solutions randomly, initialize $NDS = \{\}$;
- Step 2: Do the non-dominated check on the population based on the discussion in Section 5.2.1, and update the NDS set
- Step 3: Set Solution X (randomly selected from initial population) to be the current solution;
- Step 4: Set $obj_1 = TFT$; and $obj_2 = MS$
- Step 5: For F=1 to Family size
  Set $T = T_0$, No. of Moves = 1, and counter = 1
  - o Step 5.1: while $T > T_F$ do:
  - o Step 5.2: Generate a new neighbour solution Y from current solution X by minor swapping on job sequence of Fth family F
  - o Step 5.3: Generate a random number RN uniformly between 0 and 1, and calculate:
    $\triangle E_1 = obj_1(Y) - obj_1(X)$; $\triangle E_2 = obj_2(Y) - obj_2(X)$
    IF $\triangle E_1 \leq 0$, and $\triangle E_2 \leq 0$ then X=Y;
    IF $\triangle E_1 > 0$, and $\triangle E_2 \leq 0$;
    IF $\left\{RN < \frac{T}{T^2+(\triangle E_1)^2}, then X = Y\right\}$
    ELSE No. of Moves = No. of Moves + 1;
    ELSE IF $\triangle E_1 \leq 0$, and $\triangle E_2 > 0$;
    IF $\left\{RN < \frac{T}{T^2+(\triangle E_2)^2}, then X = Y\right\}$
  - o Step 5.4: Do non-dominated check with the possibility of in the population and update NDS
  - o Step 5.5: IF (counter = I_{iter}, OR No. of Moves =$N_{non-imp.}$)
    T= αT
    I_{iter}=0;
    No. of Moves =0;
    ELSE counter = counter+1;
- Step 6: Return the NDS as the final solution set (Pareto front)

swapping and insertion. Then, sequence of families is changed to pick another family for swapping and insertion for a number of iterations.

5.7.3 Computational results

The proposed MOSA algorithm is coded using C++ language and run on a PC with an Intel core I7 (2.93 GHz) CPU and 4.0 GB memory. The performance of the proposed MOSA algorithm is tested using test problems proposed by Schaller (Section 3.4). The computational results of Minimum, Average, and Maximum TFT and MS are summarized in Tables 5.5, 5.6, 5.7 for small problems, and results of large test problems are presented in Table 5.8 with small, medium, and large setup times. A sample of generated Pareto fronts of different test problems is presented in Figures 5.4, 5.5, 5.6, and 5.7.

Table 5.5: Results of the proposed MOSA for small setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	66	66.2	67	172	174.5	175	66	1.34	0.00%
	3	3	9	86	86	86	484	484	484	86	1.89	0.00%
	3	3	9	100	103	104	524	527.6	546	100	2.95	0.00%
	3	3	5	75	82.3	86	249	269.9	275	75	2.73	0.00%
	3	3	13	117	117.2	118	872	886.4	890	117	2.44	0.00%
	3	3	12	124	124.5	125	837	839	841	124	2.74	0.00%
	3	3	15	125	125.7	129	1114	1118	1119	122	3.44	2.46%
	3	3	15	125	130.7	133	1147	1165.4	1210	125	4.82	0.00%
	3	4	9	102	104.3	106	580	599	610	101	2.26	0.99%
	3	4	15	153	153.8	154	1386	1401.5	1448	151	4.09	1.32%
	3	4	13	134	135.5	136	1076	1088	1131	130	2.53	3.08%
	3	4	13	125	127.8	130	1047	1080	1102	123	5.7	1.63%
	3	4	9	116	116	116	674	674	674	115	3.37	0.87%
	3	4	7	96	96	96	402	402	402	96	2.59	0.00%
	3	4	14	128	133.1	135	1084	1118	1187	128	4.56	0.00%
Small	3	4	14	120	126.4	130	1108	1125	1152	118	3.28	1.69%
	3	4	15	140	145.2	147	1296	1304.4	1319	140	3.44	0.00%
	3	4	7	80	86.4	88	403	404.9	410	80	1.83	0.00%
	3	4	14	131	131.4	132	1053	1064	1078	129	4.36	1.55%
	3	4	12	111	115.6	121	769	871	904	109	3.28	1.83%
	4	4	8	104	104	104	563	563	563	104	5.16	0.00%
	4	4	14	139	140	141	1120	1126.7	1131	138	5.44	0.72%
	4	4	13	132	138.7	140	976	1004.3	1146	132	4.34	0.00%
	5	5	15	155	155.8	156	1353	1385.3	1482	153	5.99	1.31%
	5	5	15	152	160	166	1268	1331.7	1439	149	6.32	2.01%
	5	6	14	158	159.9	166	1396	1423	1444	154	8.83	2.60%
	6	5	15	170	178	189	1552	1608	1709	164	12.78	3.66%
										Ave :	4.17	0.95%
										max	12.78	3.66 %

Table 5.6: Results of the proposed MOSA for medium setup instances

Setup Time	Problem Size			Makespan			Total Flow Time			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	116	116	116	329	329	329	116	1.32	0.00%
	3	3	9	150	155.2	176	883	1036	1074	149	2.81	0.67%
	3	3	9	152	159.4	164	757	828.5	934	152	2.48	0.00%
	3	3	5	117	117	117	405	405	405	114	1.71	2.63%
	3	3	13	177	180.8	186	1079	1115	1163	177	2.02	0.00%
	3	3	12	200	205.4	208	1375	1381	1406	199	2.52	0.50%
	3	3	15	161	162.9	166	1469	1484	1499	161	2.96	0.00%
	3	3	15	164	179.9	203	1421	1520	1574	164	3.09	0.00%
	3	4	9	150	150.2	151	755	770.8	774	150	2.37	0.00%
	3	4	15	211	217	219	1943	1993	2121	207	3.44	1.93%
	3	4	13	192	199.1	207	1636	1691	1717	192	3.08	0.00%
	3	4	13	187	187.8	191	1574	1616	1626	186	3.19	0.54%
Medium	3	4	9	178	179	182	1118	1123	1125	174	3.82	2.30%
	3	4	7	156	156	156	633	633	633	156	3.06	0.00%
	3	4	14	179	181.4	184	1414	1473	1647	179	4.57	0.00%
	3	4	14	187	196.9	219	1725	1742	1756	186	4.67	0.54%
	3	4	15	172	173.9	175	1629	1631	1635	172	2.62	0.00%
	3	4	7	119	119	119	593	593	593	119	2.42	0.00%
	3	4	14	199	205	212	1697	1837	1865	199	5.59	0.00%
	3	4	12	161	161	161	1335	1335	1335	160	3.49	0.63%
	4	4	8	160	192	204	851	856.4	887	160	3.84	0.00%
	4	4	14	196	196	196	1476	1476	1476	196	3.87	0.00%
	4	4	13	203	212.2	219	1363	1430	1809	199	3.59	2.01%
	5	5	15	233	249.1	274	1936	2064	2245	233	7.28	0.00%
	5	5	15	211	211.3	213	1869	1875	1876	211	6.71	0.00%
	5	6	14	254	275.3	314	2230	2322	2500	254	8.41	2.00%
	6	5	15	254	257.4	269	2376	2539	2646	251	11.29	1.20%
										Ave :	4.03	0.57 %
										max	11.29	2.63 %

5.8 An Improved Multi-objective Particle swarm optimization algorithm

An Improved Multi-objective Particle Swarm Optimization (IMPSO) is implemented in this study to improve the convergence of the Pareto fronts obtained by MPSO. Moreover, TA algorithm is combined with IMPSO algorithm as a local search to improve the obtained non-dominated solution sets. The proposed algorithm is named IMPSO-TA.

5.8.1 An Improved Multi-objective Particle Swarm Optimization (IMPSO)

The performance of classic PSO like all evolutional algorithms depends on the search parameters. Further, it often suffers from problem of premature convergence that might be

Table 5.7: Results of the proposed MOSA for large setup instances

Setup Time	Problem Size			Makespan			Total Flow Time			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	162	162	162	439	439	439	162	1.39	0.00%
	3	3	9	263	268.7	297	1467	1471	1472	263	2.5	0.00%
	3	3	9	257	267.7	273	1336	1451	1683	257	2.39	0.00%
	3	3	5	254	265	311	900	930.1	936	254	1.86	0.00%
	3	3	13	278	285.5	289	2077	2213	2485	278	2.46	0.00%
	3	3	12	251	293.5	323	1847	2122	2369	251	4.21	0.00%
	3	3	15	253	253.2	254	2493	2496	2497	253	3.21	0.00%
	3	3	15	269	274.2	281	2395	2507	2574	266	4.44	1.13%
	3	4	9	237	242.9	244	1318	1342	1480	236	2.18	0.42%
	3	4	15	290	325.6	341	2806	2838	2932	290	4.46	0.00%
	3	4	13	268	272.3	278	2054	2096	2121	266	2.77	0.75%
	3	4	13	263	263	263	1831	1831	1831	261	2.88	0.77%
Large	3	4	9	264	276.4	281	1503	1600	1847	264	2.34	0.00%
	3	4	7	251	251.8	252	1002	1002	1003	251	2.82	0.00%
	3	4	14	284	285.5	286	2105	2109	2128	284	3.19	0.00%
	3	4	14	291	325.5	337	2091	2107	2153	291	3.22	0.00%
	3	4	15	286	286.8	287	2905	2905	2906	286	6.1	0.00%
	3	4	7	202	203.7	204	988	988.2	989	201	4.22	0.50%
	3	4	14	309	316	318	2513	2739	3155	309	5.38	0.00%
	3	4	12	249	250.3	257	1396	1733	1800	246	6.61	1.22%
	4	4	8	284	284	284	1351	1351	1351	284	5.06	0.00%
	4	4	14	324	328.4	344	2855	3192	3296	323	5.57	0.31%
	4	4	13	331	337.1	365	2304	2967	3089	331	5.64	0.00%
	5	5	15	395	412.8	422	3320	3355	3415	392	9.19	0.77%
	5	5	15	372	387.8	409	3246	3504	4270	372	12.74	0.00%
	5	6	14	418	449.2	464	2697	2876	3797	412	14.97	1.46%
	6	5	15	441	498.1	577	4075	4233	4933	436	14.79	1.15%
										Ave :	5.06	0.31%
										max	14.97	1.46%

Table 5.8: Results of the proposed MOSA for Large setup instances

Setup Time	Problem Size			Makespan			Total Flow Time			MakeSpan Lower Bound	CPU Time (sec)	Deviation for Makespan	
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max				
	3	3	17	144	144.3	145	1457	1470.8	1481	144	0.00%	2.99	
	4	4	21	185	191.7	195	2150	2190.1	2301	180	2.78%	3.97	
	5	5	22	282	286.1	289	5039	5079.2	5218	271	4.06%	10.71	
	6	5	19	199	202.2	209	2245	2300.17	2317	192	3.65%	13.65	
Small	5	6	39	309	316.1	334	6801	7069.4	7356	301	2.66%	16.48	
	6	6	39	338	346	351	7526	7573.6	7805	321	5.30%	19.5	
	8	8	37	372	375.9	388	8116	8411.5	8567	349	6.59%	34.25	
	10	10	59	538	546.9	558	18175	18417.6	18819	491	9.57%	95.89	*Ave.* = 4.32%
	3	3	17	208	210	216	2212	2236	2243	207	0.48%	4.05	
	4	4	21	270	279.2	286	3073	3142	3372	269	0.37%	4.93	
	5	5	22	355	362.3	377	6681	6948.8	7081	355	0.00%	11.47	
	6	5	19	287	294.6	312	3005	3309.6	3498	283	1.41%	13.7	
Medium	5	6	39	386	393.4	409	8314	8640.6	9205	378	2.12%	17.51	
	6	6	39	426	436.5	457	9396	9931.5	10302	416	2.40%	20.29	
	8	8	37	484	513.1	554	10068	10691.1	11357	465	4.09%	36.26	
	10	10	59	694	714.3	722	23017	23319.8	23518	647	7.26%	105.97	*Ave.* = 2.27%
	3	3	17	311	315.4	341	3000	3067.4	3081	311	0.00%	4.03	
	4	4	21	367	368.4	369	4657	4661.2	4669	366	0.27%	5.39	
	5	5	22	495	504.1	519	8671	9525.9	9951	492	0.61%	12.86	
	6	5	19	415	454.3	496	4997	5273.1	5627	410	1.22%	13.96	
Large	5	6	39	551	561.3	626	12871	13630.5	14162	539	2.23%	16.44	
	6	6	39	599	607.8	618	13871	14212.3	14746	584	2.57%	21.73	
	8	8	37	723	743.3	764	14387	15452	18313	696	3.88%	38.46	
	10	10	59	1010	1035.6	1113	32195	34848.8	36779	937	7.79%	116.79	*Ave.* = 2.32%
											Ave. =2.97%	*Ave.* = 26.72	

because of being trapped in local optima. The probability of being trapped into the local optima can be reduced by improving the ability of particles to explore the global and local

optimization solutions. The Decline Disturbance Index (DDI) proposed by [1] is applied to enhance the ability of particles to explore the global and local optimization solutions. The DDI is added to improve the updating velocity formula. Therefore the new formula for the velocity will be as follows:

$$v_j^{t+1} = wv_j^t + C_1 r_1^*(P_j^t - x_j(t)) + C_2 r_2^*(G^t - x_j(t)) + lr_3 \tag{5.10}$$

$$x_j^{t+1} = v_j^{t+1} + x_j^t \tag{5.11}$$

Where $j \in \{1, 2, ..., n\}$, $t \in \{1, 2, ..., I_{max}\}$, P_j^t is the best for the jth particle and the i_{th} iteration, G^t is the g-best till the tth iteration, $l = -d_1(x - d_2)$ is a linear decline function controlled by parameters d_1 and d_2 with $xt\triangle(x)$. Both d_1 and d_2 are small parameters that can be set dynamically, t is the iteration index, and $\triangle(x)$ is an interval whose length can be adjusted according to the objective functions. During the evolution process of the algorithm, the DDI declines at a certain rate, and has minimal impact on the evolution of the particles at last, thus, increasing the chance of convergence to an optimum solution [1].

5.8.2 Local Search based on Threshold Acceptance (LS-TA)

The ultimate goal of any search strategy is to find an optimal or a near-optimal solution. Local search or Memetic Algorithms (MAs) are usually implemented due to the simple and successful application of these algorithms to various optimization problems. Ishibuchi et al presented the first MA algorithm named as a multiobjective genetic local search approach. They implemented the local search algorithm for multi-objective flowshop scheduling prob-

lems and they concluded that a high performance is demonstrated by applying the local search to multi-objective flowshop scheduling problems [121]. Knowles et al provided a simple framework that guides for designing MAs for MOPs [122]. A local search method based on TA is implemented to enhance the g-best after generating the initial swarm to improve the intensification. Due to the computational time advantage, TA is implemented to improve the quality of Pareto sets obtained by MPSO by guiding the g-best to the promising regions. Basically, TA escapes from local optima by accepting solutions that are not worse than the current by more than a given threshold [45]. The pseudo code of the LS-TA approach is shown in Algorithm 5.3.

5.9 The proposed IMPSO-TA algorithm

The proposed IMPSO-TA is similar to MPSO (Algorithm 5.1). The velocity and position of each particle are updated according to equations 5.10 and 5.11. Local search algorithm is inserted in each swarm to increase the chance of getting g-best in promising regions, and to enhance the quality of the g-best. TA (Algorithm 5.3) is combined with IMPSO to form a hybrid algorithm named as IMPSO-TA to solve FMCSP with SDSTs to minimize TFT and MS simultaneously. Therefore, the overall steps of IMPSO-TA is shown in Algorithm 5.4.

5.9.1 Computational results

The proposed IMPSO-TA algorithm is coded using C++ language and run on a PC with an Intel core I7 (2.93 GHz) CPU and 4.0 GB memory. The performance of the proposed MOSA algorithm is tested using test problems proposed by Schaller (Section 3.4). Results

Algorithm 5.3 The Pseudo code for Loacl Search Threshold Acceptance algorithm LS-TA

Require: Initialize the search parameters $\{ I_{iter}, N_{non-imp.}, Familysize\}$

- Step 1: Set Solution X to be the current
- Step 2: Set $obj_1 = TFT$; and $obj_2 = MS$
- Step 3: For F=1 to Family size
  Set No. of Moves =1, and counter = 1
- Step 4 while counter $< I_{iter}$
- Step 4: Generate a new neighbour solution Y from current solution X by minor swapping on the job sequence of family F
- Step 5: Calculate: $\triangle E_1 = obj_1(Y) - obj_1(X)$; $\triangle E_2 = obj_2(Y) - obj_2(X)$
- Step 6: Generate a random number RN uniformly between 0 and 1:
  IF $\triangle E_1 \leq 0$, and $\triangle E_2 \leq 0$; then$X = Y$
  Else $\triangle E_1 > 0$, and $\triangle E_2 \leq 0$;
  IF $\left[RN < \frac{T}{T^2+(\triangle E_1)^2}, then X = Y\right]$
  else No. of Moves = No. of Moves + 1;
  Else $\triangle E_1 \leq 0$, and $\triangle E_2 > 0$;
  IF $\left[RN < \frac{T}{T^2+(\triangle E_2)^2}, then X = Y\right]$
  else No. of Moves = No. of Moves + 1;
- Step 7 IF (No. of Moves =$N_{non-imp.}$)
  counter = counter+1;

---

of Minimum, Average, and Maximum TFT and MS are summarized in Tables 5.9, 5.10, 5.11 for small problems, and in Table 5.12 for large problems with small, medium, and large setup times. As shown in Figures 5.4, 5.5, 5.6, and 5.7, the obtained Pareto fronts are compared with Pareto fronts found by MOSA.

Algorithm 5.4 The steps of proposed IMPSO-TA algorithm

Require: Initialize parameters { swarm size n, maximum Iteration I_{max}, C_{1max}, C_{1min}, $C_{2max}, C_{2min}, V_{max}, V_{min}, W_{max}, W_{min}, X_{max}, X_{min}$}

- Step 1: Set iteration $t = 0$
- Step 2: If $t = 0$
  - o Generate initial positions X_i^t and V_i^t initial velocities for $i \in \{1, 2, ..., n\}$ according to equations (5.6, and 5.7)
    Else
  - o Generate a new swarm by updating the velocity V_i^t and position X_i^t of particles according to equations (5.10, and 5.11)
- Step 3: Set the initial Non Dominated Set $NDS = \{\}$ (The Null Set)
- Step 4: Apply the (ROV) on X_i^t to find the sequence of families as well as jobs in families.
- Step 5: Calculate the objective function f_{TFT} (X_i^t) and f_{MS} $(X_i^t)$for each particle $i \in \{1, 2, ..., n\}$
- Step 6: Check whether the particle (X_i^t) qualifies to be included in NDS based on the discussion in Section 5.2.1
- Step 7 Implementing the local search $LS - TA$
  - o Step 7.1 For each particle in the swarm the p-best (P_i^t) is calculated as:

$$P_i^t = \underset{X_i^j}{\operatorname{argmin}} f(X_i^j) \quad for \quad j \in \{1, 2, ..., t\}$$

$$f_i^t = \left(0.5 f_{TFT}(X_i^t) + 0.5 f_{MS}(X_i^t)\right)$$

  - o Step 7.2 For $j = \{1, 2, ..., NDS\}$, $G^t = LS - TA(G^t)$
  - o Step 7.3 Do the non-dominated check (Section 5.2.1), and update the NDS
  - o Step 7.4 The g-best G^t is randomly selected from the NDS set.
- Step 8 Set $t = t + 1$
- Step 9 If $t = I_{max}$, STOP; and set $NDS^{I_{max}}$ as the Pareto front; else, go to step 2.

Table 5.9: Results of the proposed IMPSO-TA for small setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	66	66.7	67	172	173	175	66	8.17	0.00%
	3	3	9	86	86	86	484	484	484	86	10.12	0.00%
	3	3	9	100	101.5	104	524	530	542	100	10.78	0.00%
	3	3	5	75	81.3	86	249	265.7	275	75	8.49	0.00%
	3	3	13	117	117.3	118	880	882.3	887	117	11.68	0.00%
	3	3	12	124	126.8	129	858	866	883	124	11.18	0.00%
	3	3	15	125	125	125	1107	1107	1107	122	10.65	2.46%
	3	3	15	125	130.2	135	1136	1161.7	1205	125	13.74	0.00%
	3	4	9	102	104.75	106	580	595.5	610	101	4.22	0.99%
	3	4	15	153	153.7	154	1376	1395.7	1435	151	13.52	1.32%
	3	4	13	133	134.3	136	1065	1078	1084	130	12.8	2.31%
	3	4	13	126	128.3	130	1044	1068	1085	123	12.67	2.44%
Small	3	4	9	116	116	116	674	674	674	115	11.18	0.87%
	3	4	7	96	96	96	402	402	402	96	10.61	0.00%
	3	4	14	128	131.4	134	1076	1111	1173	128	13.46	0.00%
	3	4	14	120	123.2	128	1093	1103	1115	118	13.66	1.69%
	3	4	15	140	144.5	147	1273	1285.3	1305	140	14.36	0.00%
	3	4	7	80	84.3	88	403	407.3	410	80	8.42	0.00%
	3	4	14	131	131.5	132	1043	1049	1055	129	11.24	1.55%
	3	4	12	111	115	121	761	824.8	889	109	10.33	1.83%
	4	4	8	104	104	104	563	563	563	104	11.31	0.00%
	4	4	14	139	139	139	1105	1105	1105	138	14.29	0.72%
	4	4	13	132	134.7	140	961	1073.7	1130	132	17.77	0.00%
	5	5	15	156	156.3	157	1342	1344.7	1347	153	24.39	1.96%
	5	5	15	152	158.7	166	1237	1293.4	1413	149	27.01	2.01%
	5	6	14	157	166.2	176	1392	1406.2	1417	154	29.95	1.95%
	6	5	15	168	174.4	180	1543	1629.1	1787	164	31.97	2.44%
										Ave :	13.9	0.91%
										max	31.9	2.46%

Table 5.10: Results of the proposed IMPSO-TA for medium setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	116	116	116	329	329	329	116	8.37	0.00%
	3	3	9	150	163	176	882	978	1074	149	10.58	0.70%
	3	3	9	152	158.8	164	755	834	934	152	10.66	0.00%
	3	3	5	117	117	117	405	405	405	114	8.6	2.60%
	3	3	13	177	179.5	182	1073	1113.5	1154	177	11.6	0.00%
	3	3	12	200	202	204	1367	1375.6	1384	199	13.57	0.50%
	3	3	15	161	164.5	169	1461	1468	1476	161	11.3	0.00%
	3	3	15	164	184.8	199	1405	1469.2	1562	164	13.4	0.00%
	3	4	9	150	150.7	151	755	761.3	774	150	11.28	0.00%
	3	4	15	211	215	219	1905	2000	2095	207	13.7	1.90%
	3	4	13	192	199.3	207	1626	1667	1711	192	13.03	0.00%
	3	4	13	187	188	190	1560	1594.7	1612	186	12.84	0.50%
Medium	3	4	9	178	180.3	182	1118	1121	1125	174	11.64	2.30%
	3	4	7	156	156	156	633	633	633	156	10.52	0.00%
	3	4	14	179	179.5	180	1390	1505.5	1621	179	15.59	0.00%
	3	4	14	187	195.3	215	1699	1725	1738	186	15.85	0.50%
	3	4	15	172	173	174	1613	1615.5	1618	172	17.44	0.00%
	3	4	7	119	119	119	593	593	593	119	10.45	0.00%
	3	4	14	199	204.7	212	1697	1797.7	1850	199	13.75	0.00%
	3	4	12	161	161	161	1335	1335.5	1336	160	12.41	0.60%
	4	4	8	160	180	204	851	861.75	887	160	14.19	0.00%
	4	4	14	196	197	198	1455	1457.5	1460	196	17.77	0.00%
	4	4	13	202	208.5	219	1355	1457.9	1760	199	20.43	1.50%
	5	5	15	233	243.4	254	1915	2080.8	2237	233	24.83	0.00%
	5	5	15	211	211	211	1846	1851	1856	211	25.89	0.00%
	5	6	14	254	266	287	2215	2357.8	2489	254	30.03	0.00%
	6	5	15	254	268.3	300	2366	2415.5	2644	251	31.68	1.20%
										Ave :	15.24	0.46 %
										max	31.68	2.63 %

Table 5.11: Results of the proposed IMPSO-TA for large setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			Makespan Lower Bound	CPU Time (sec)	Deviation for Makespan
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max			
	3	3	4	162	162	162	439	439	439	162	8.95	0.00%
	3	3	9	263	280	297	1467	1469	1471	263	10.4	0.00%
	3	3	9	257	265	273	1336	1509	1683	257	10.59	0.00%
	3	3	5	254	269.3	311	900	926.8	936	254	8.59	0.00%
	3	3	13	278	283.5	289	2073	2271.5	2470	278	11.72	0.00%
	3	3	12	251	289.6	323	1845	2130.7	2356	251	13.8	0.00%
	3	3	15	253	253	253	2477	2477	2477	253	11.01	0.00%
	3	3	15	269	275	281	2385	2468	2551	266	12.97	1.13%
	3	4	9	237	241.3	244	1318	1373.3	1480	236	11.46	0.42%
	3	4	15	290	311.7	341	2791	2852.3	2923	290	14.13	0.00%
	3	4	13	268	273	277	2053	2090	2110	266	12.77	0.75%
	3	4	13	263	263	263	1828	1828.5	1829	261	13	0.77%
Large	3	4	9	264	274.3	286	1503	1679.3	1847	264	11.49	0.00%
	3	4	7	251	251.5	252	1002	1002.5	1003	251	10.61	0.00%
	3	4	14	284	285	286	2089	2095	2109	284	15.49	0.00%
	3	4	14	291	322	338	2084	2106	2150	291	15.4	0.00%
	3	4	15	286	286.5	287	2893	2896	2899	286	16.79	0.00%
	3	4	7	202	202.7	204	988	988.7	989	201	10.8	0.50%
	3	4	14	309	314.7	318	2506	2783	3142	309	14	0.00%
	3	4	12	249	253	257	1393	1596	1799	246	12.31	1.22%
	4	4	8	284	284	284	1351	1351	1351	284	14.44	0.00%
	4	4	14	324	332.7	344	2849	3091	3293	323	18.27	0.31%
	4	4	13	331	342.7	365	2284	2816.3	3094	331	20.45	0.00%
	5	5	15	395	404.7	422	3307	3376	3413	392	24.49	0.77%
	5	5	15	372	382.5	409	3237	3545.3	4269	372	26.9	0.00%
	5	6	14	417	434.2	460	2655	3003.3	3774	412	31.05	1.21%
	6	5	15	439	472	515	4011	4391.4	4969	436	32.66	0.69%
										Ave :	15.35	0.29%
										max	32.66	1.22 %

Table 5.12: Results of the proposed IMPSO-TA for Large setup instances

Setup Time	**Problem Size**			**Makespan**			**Total Flow Time**			MakeSpan Lower Bound	CPU Time (sec)	Deviation for Makespan	
	F	M	J	Min.	Ave.	Max.	Min.	Ave.	Max				
	3	3	17	144	144	144	1428	1428.5	1431	144	0.00%	12.93	
	4	4	21	182	189	193	2137	2191.4	2244	180	1.11%	20.33	
	5	5	22	279	280.5	282	4883	4887	4891	271	2.95%	34.47	
	6	5	19	196	202	208	2208	2251	2281	192	2.08%	30.82	
Small	5	6	39	304	313.3	321	6580	6665.7	6731	301	1.00%	40.74	
	6	6	39	331	335.5	340	7315	7495	7597	321	3.12%	48.51	
	8	8	37	363	369	380	7772	7944.3	8030	349	4.01%	76.96	
	10	10	59	517	521.17	524	17236	17309	17508	491	5.30%	153.18	*Ave.* = 2.45%
	3	3	17	208	213.75	217	2199	2208.8	2233	207	0.48%	13.11	
	4	4	21	270	274.83	279	3018	3144.3	3316	269	0.37%	20.58	
	5	5	22	355	372.14	383	6532	6635	6996	355	0.00%	34.76	
	6	5	19	287	302.14	311	2982	3110.7	3471	283	1.41%	31.86	
Medium	5	6	39	378	387.17	397	8053	8340.2	8699	378	0.00%	42.43	
	6	6	39	418	429.45	442	9048	9279.3	9501	416	0.48%	50.12	
	8	8	37	477	508.38	541	9609	10008	11022	465	2.58%	77.56	
	10	10	59	671	680	692	22159	22323	22469	647	3.71%	153.22	*Ave.* = 1.13%
	3	3	17	311	326	341	2979	3021	3063	311	0.00%	13.23	
	4	4	21	367	367	367	4611	4611	4611	366	0.27%	20.48	
	5	5	22	493	500.17	510	8590	9548.5	9784	492	0.20%	35.31	
	6	5	19	414	453.19	493	4953	5243.9	5631	410	0.98%	32.46	
Large	5	6	39	539	547.83	554	12748	12980	13415	539	0.00%	42.84	
	6	6	39	592	600.5	609	13600	14019	14490	584	1.37%	50.55	
	8	8	37	707	734.42	764	13872	14384	18062	696	1.58%	78.32	
	10	10	59	954	956.83	965	31504	31537	31568	937	1.81%	155.43	*Ave.* = 0.78%
											Ave. =1.45%	*Ave.* = 52.92	

5.10 Performance Analysis and Discussion

The ultimate goal of this chapter is to design an efficient algorithm to generate a good approximation of the Pareto front. A rigorous analysis is conducted to evaluate the performance of the proposed MPSO, MOSA, and IMPSO-TA for bi-objective optimization scheduling problem. Addressing the efficiency of an algorithm often includes considering a certain amount of computational time. The average CPU time is calculated for the same number of test problems with different problem sizes and different setup times. Figure 5.1 shows the CPU time for a range of problems. LSU, MSU, and SSU refer to large, medium, and small setup times respectively. For maximum total number of 15 jobs, the fourth bar (LARGE) represents the average CPU time of LSU, MSU, and SSU for the case more than 15 jobs. There is not a significant difference in the CPU time of the proposed MPSO and MOSA for different setup time classes for small size test problems (less than 15 jobs). However, the CPU time is considerably increased for the large test problems. MPSO is the best for large problem in terms of CPU time. According to [7], there is no proper lower bound for TFT for the case of test problems proposed by [40]. Therefore the lower bound of makespan is used to evaluate the proposed algorithms. Relative Percentage Deviation RPD is calculated according to the following equation:

$$RPD = \left(\frac{MS - LB}{LB}\right) \times 100 \tag{5.12}$$

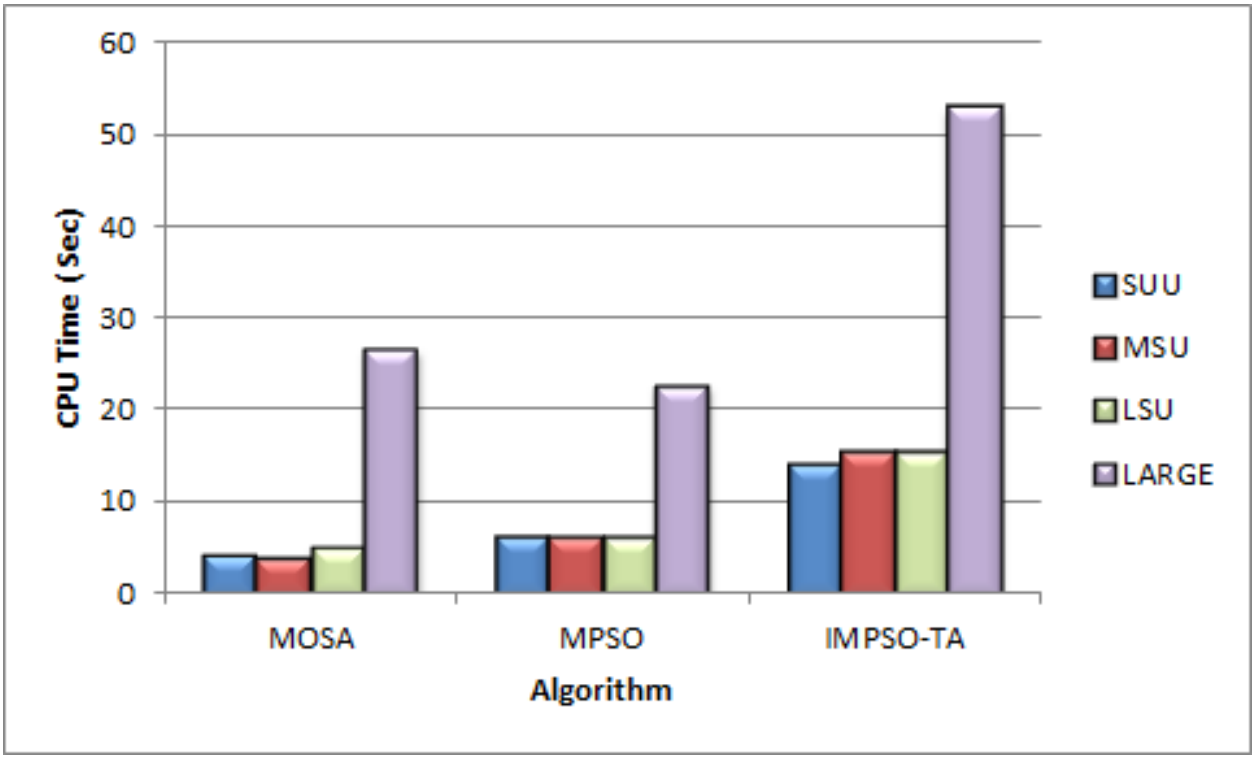

Figure 5.1: The average CPU time for the proposed algorithms

Figure 5.2 compares the RPD for the proposed algorithms; it shows that IMPSO-TA algorithm is slightly better than MOSA and MPSO due to use of the local search and improving the diversity by adding the Decline Disturbance Index (DDI) to update the velocity of the particles. Figure 5.3 compares the RPD for the proposed algorithms for large problems; it shows that IMPSO-TA algorithm outperform MOSA and MPSO due to use of the local search. Further, the performance of the proposed IMPSO-TA is compared with the MOGA proposed by Hendizadeh et al [7] and TLMSA that developed by Lin et al [8]; it turns out that the IMPSO-TA performed the same as MOGA and TLMSA in the small and medium test problems and there is insignificant deviation for large test problems.

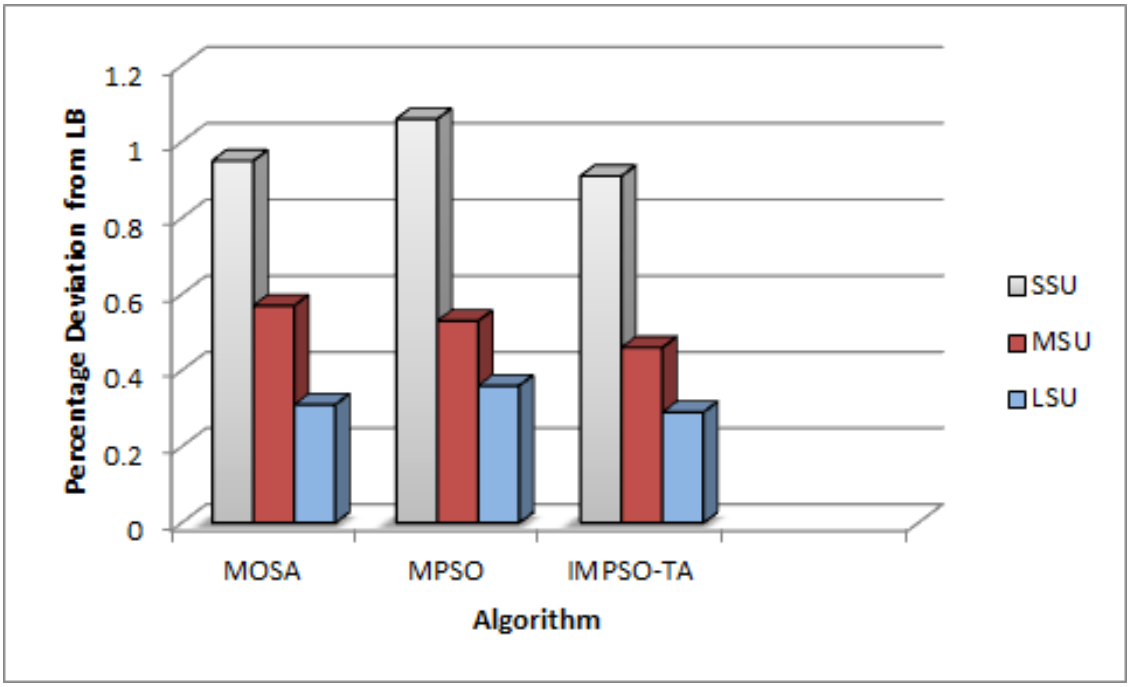

Figure 5.2: The percentage deviation from makespan for small test problems

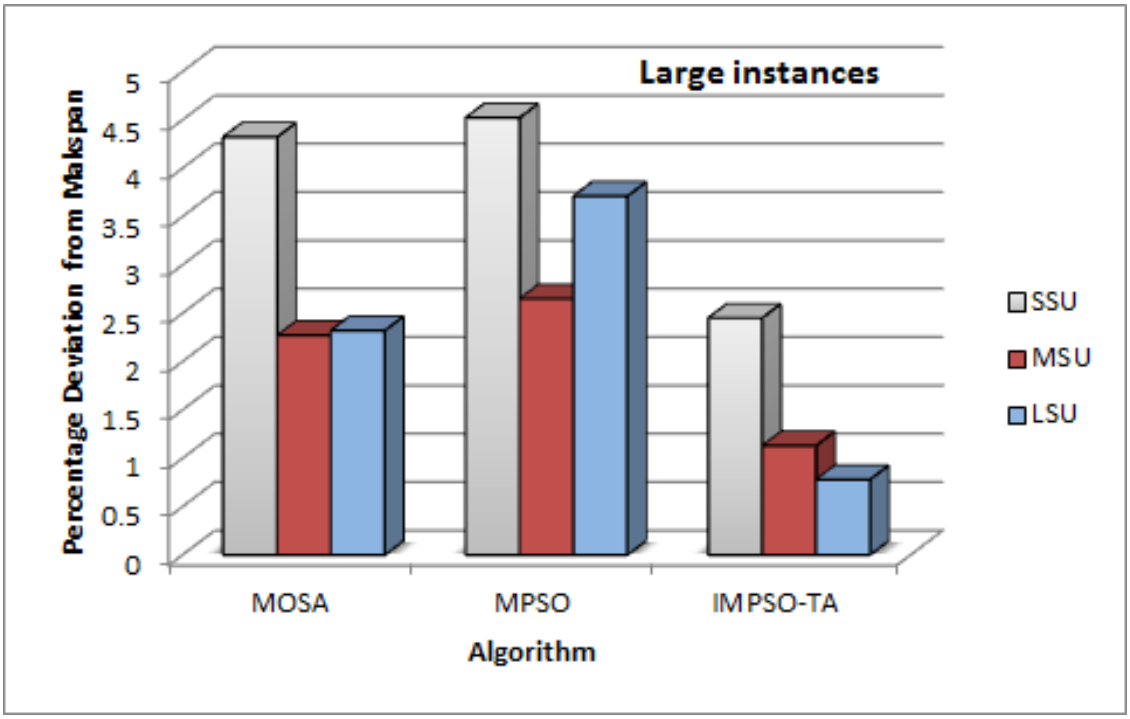

Figure 5.3: The percentage deviation from makespan for large test problems

Moreover, various performance measures have been defined in the literature to compare the quality of generated Pareto fronts. These quality indicator measures will be described in Section 5.10.1. Interested readers may refer to [70, 104, 123–125] for more details about performance metrics for multi-objective optimization.

5.10.1 Performance measures based on Quality Indicators

Unlike the single objective optimization, there are several Quality Indicators (QI) often used to evaluate the quality of NDS. These performance indicators are widely used in practice. In this work, the QIs listed below are calculated using the C++ code presented in Appendix 1.

1. **Number of Non-dominated solutions**

   The first quality measure used in this research is how many non-dominated solutions in a Pareto front. Let NDS_1 is the Pareto set generated by IMPSO-TA algorithm. The size of this set is X_1. Similarly, X_2 is the size of Pareto set obtained by using MOSA algorithm.

2. **Percentage of non-dominated solutions**

   The ratio between the sizes of NDS in each set is calculated (X_1/X_2) to indicate which algorithm generates more non-dominated solutions.

3. **Coverage index** (CM)

   For the two different non-dominated solution sets X and X', consider a mapping $(X, X') \rightarrow [0, 1]$ where $x \in X$, and $x' \in X'$. If a solution a dominates a solution

b , that means $b \prec a$, then the CM is defined by:

$$CM(X, X') = \frac{\mid x' \in X' \mid \exists : x \in X : x' \prec x \mid}{X'} \tag{5.13}$$

If all the solutions in X' were dominated by the solutions in X, then $CM(X, X') = 1$; on the contrary, if all the solutions in X were dominated by the solutions in X' , then $CM(X, X') = 0$.

4. D_{ave} and D_{max}

   The performance measure D_{av} is defined as:

$$D_{av}(X) = \frac{1}{\mid X' \mid} \sum_{x' \in X'} \min_{x \in X} (d_{xx'} \mid x \in X) \tag{5.14}$$

   D_{max} is defined as:

$$D_{max}(X) = \frac{1}{\mid X' \mid} \max_{x' \in X'} \{\min_{x \in X} \{d_{xx'} \mid x \in X\}\} \tag{5.15}$$

   where

$$d_{xx'} = \sqrt{(obj_1(x') - obj_1(x))^2 + (obj_2(x') - obj_2(x))^2} \tag{5.16}$$

5. **Spacing Distance measure**

   Multi-objective evolutionary methods aim to minimize the distance between the obtained solution and the true Pareto front [126]. The Spacing Distance (SD) measures the uniformity spread of solutions over the approximated Pareto front. For

non-dominated solutions set, SD metric is calculated using the following equation:

$$SD(X) = \sqrt{\frac{1}{X}\sum_{1}^{X}(d_i - d')^2} \tag{5.17}$$

where $d' = \sum_{i=1}^{|X|} \frac{d_i}{|X|}$, d_i is the Euclidean distance between solution x_i in X and the nearest solution in the space. X is the number of the non-dominated solution in the Pareto front. The smaller of the SD, the superior of the solution distribution is.

6. **Quality Index measure**

Quality index QI is used to compare the obtained Pareto front with the true Pareto front. Since there is no true Pareto front for the studied problem, the Pareto fronts found by IMPSO-TA and the MPSO are compared using QI. Quality index can be calculated using the following formula:

$$QI = \sum_{x \in X} \left\{ 1 - Min\left(\frac{obj_1^x - \min_{x' \in X'} \left\{ obj_1^{x'}(X') \right\}}{\min_{x' \in X'} \left\{ obj_1^{x'}(X') \right\}}, \frac{obj_2^x - \min_{x' \in X'} \left\{ obj_2^{x'}(X') \right\}}{\min_{x' \in X'} \left\{ obj_2^{x'}(X') \right\}} \right) \right\} / \mid X \mid \tag{5.18}$$

5.10.2 Discussion of the quality measures results

The results of the performance evaluation based on the QI are presented in Tables 5.13, 5.14, and 5.15 for small, medium, and large test problems. From the viewpoint of the number of non-dominated solutions (NDS), in most cases the average number of non-dominated solutions found by IMPSO-TA is greater than the obtained solutions by MOSA. Furthermore, the results of CM metric indicate that solutions found by MOSA are dominated by the one

obtained by IMPSO-TA ($CM = 0$). Therefore, quality of Pareto fronts found by IMPSO-TA is better than Pareto fronts generated by MOSA algorithm. For instance, the divergence between the proposed algorithms is shown in Figure 5.4 for one of the small instances, and Figure 5.5 for one of the large instances. In addition, as shown in Figure 5.6, the variation between the obtained Pareto fronts is increased as the problem size is increased. Yet, the variation is less for small size instances as shown in Figure 5.7. Furthermore, the Pareto fronts generated by IMPSO-TA is better than that of MOSA in terms of average distance $D_{ave}$and as well as in terms of maximum distance (D_{max}). QI metric is introduced to evaluate the performance between proposed algorithms for multi-objective problem. It is obvious that QI computed by IMPSO-TA is smaller than that of MOSA, which states that the overall performance for Pareto optimal from IMPSO is better than the ones that obtained by PSO and MOSA. We can conclude that IMPSO-TA is an effective algorithm to solve the bi-criteria scheduling problem of flowshop manufacturing cell with sequence dependent setup times than MPOS, and MOSA algorithms in terms of the quality of the solutions.

Table 5.13: Results of QI measures of IMPSO-TA VS MOSA for small setup time instances

Setup Time	**Problem Size**			**Quality Measures**							
	F	M	J	X_1	X_2	$RatioX_1/X_2$	CM	SD	D_{ave}	D_{max}	QI
	3	3	4	3	3	1.0	0	2.3	1.1	3.2	1.0
	3	3	9	2	2	1.0	0	0	0	0	1.0
	3	3	9	2	4	0.5	0	7.5	6.1	15.5	0.994
	3	3	5	2	3	0.7	0	9.8	17.5	28.2	1.0
	3	3	13	2	3	0.7	0	3.5	6.4	8.1	1.0
	3	3	12	2	4	0.5	0	9.3	25.4	42	1.0
	3	3	15	2	2	1.0	0	0	8.1	8.1	0.997
	3	3	15	6	6	1.0	0	12.7	4.7	8.9	0.986
	3	4	9	4	4	1.0	0	0.6	1.3	5	0.998
	3	4	15	3	3	1.0	0	34.6	11	13	1.0
	3	4	13	3	4	0.8	0	7.1	6.7	11	0.989
	3	4	13	6	4	1.5	0	8.1	5.1	11	0.993
	3	4	9	2	2	1.0	0	0	0	0	1.0
Small	3	4	7	2	2	1.0	0	0	0	0	1.0
	3	4	14	5	5	1.0	0	25.8	6	9.1	0.987
	3	4	14	7	6	1.2	0	3.1	10.1	17.5	0.981
	3	4	15	3	4	0.8	0	9.2	15.8	23	0.983
	3	4	7	2	3	0.7	0	1.7	1.7	5.1	0.993
	3	4	14	2	2	1.0	0	0	6.1	10	0.996
	3	4	12	5	6	0.8	0	2.6	7.4	8.1	0.979
	4	4	8	2	2	1.0	0	0	0	0	1.0
	4	4	14	4	3	1.3	0	42.1	7.4	10.2	0.989
	4	4	13	2	3	0.7	0	102.2	15.7	16	0.992
	5	5	15	2	3	0.7	0	1.2	8.3	11	1.0
	5	5	15	6	11	0.5	0	28.2	12.7	31	0.972
	5	6	14	3	5	0.6	0	6.9	9.5	13	0.993
	6	5	15	6	10	0.6	0	17.9	23.1	78.1	0.987

Table 5.14: Results of QI measures of IMPSO-TA VS MOSA for medium setup times instances

Setup Time	**Problem Size**			**Quality Measures**							
	F	M	J	X_1	X_2	$RatioX_1/X_2$	CM	SD	D_{ave}	D_{max}	QI
	3	3	4	2	2	2	0	0	0	0	1.0
	3	3	9	2	2	1.0	0	0	0.5	1	0.999
	3	3	9	6	6	1	0	0.52	0.3	2	0.979
	3	3	5	2	2	1	0	0	0	0	1.0
	3	3	13	5	2	2.5	0	0	7.5	9	0.995
	3	3	12	6	5	1.2	0	3.13	4.95	8.94	0.993
	3	3	15	5	4	1.3	0	4.04	4.39	8.54	0.993
	3	3	15	8	5	1.6	0	3.51	10.78	16.49	0.975
	3	4	9	2	3	0.7	0	11.55	0	0	1.0
	3	4	15	3	2	1.5	0	0	32	38	0.983
	3	4	13	5	6	0.8	0	2.45	7.65	10	0.986
	3	4	13	2	3	0.7	0	31.75	14.0119	14.04	0.996
	3	4	9	3	3	1	0	2.89	0	0	0.999
Medium	3	4	7	2	2	1	0	0	0	0	1.0
	3	4	14	5	2	2.5	0	0	25.17	26	0.989
	3	4	14	5	4	1.3	0	25.54	14.69	26.31	0.988
	3	4	15	3	2	1.5	0	0	13.72	16.03	0.993
	3	4	7	2	2	1	0	0	0	0	1.0
	3	4	14	3	3	1	0	86.6	62.28	162.11	0.987
	3	4	12	2	2	1	0	0	0.5	1	1.0
	4	4	8	4	4	1	0	20.63	0	0	0.998
	4	4	14	2	2	1	0	0	18.55	21.09	1.0
	4	4	13	8	10	0.8	0	6.54	16.52	53	0.981
	5	5	15	8	5	1.6	0	32.08	16.29	29	0.969
	5	5	15	2	2	1	0	0	18.12	23.08	0.995
	5	6	14	10	4	2.5	0	23.67	15.54	22.09	0.985
	6	5	15	5	6	0.8	0	107.55	10.76	32.57	0.998

Table 5.15: Results of QI measures of IMPSO-TA VS MOSA for large setup times instances

Setup Time	**Problem Size**			**Quality Measures**							
	F	M	J	X_1	X_2	$RatioX_1/X_2$	CM	SD	D_{ave}	D_{max}	QI
	3	3	4	2	2	1	0	0	0	0	1.0
	3	3	9	2	2	1	0	0	0.5	1	1.0
	3	3	9	4	4	1	0	1.15	0	0	0.998
	3	3	5	3	4	0.8	0	43.23	0	0	0.995
	3	3	13	3	2	1.5	0	0	8.02	12.04	0.998
	3	3	12	10	7	1.4	0	36.95	3.84	7	0.937
	3	3	15	2	2	1	0	0	16.03	16.03	0.998
	3	3	15	3	2	1.5	0	0	15.01	20.03	0.997
	3	4	9	3	3	1	0	91.8	0	0	0.999
	3	4	15	4	3	1.3	0	2.89	14.7	20.1	0.984
	3	4	13	5	3	1.7	0	27.71	2.8	4	0.994
	3	4	13	2	2	1	0	0	2.5	3	1.0
	3	4	9	3	4	0.8	0	11.55	7.37	25.5	0.997
Large	3	4	7	2	2	1	0	0	0	0	1.0
	3	4	14	3	4	0.8	0	9.11	11.81	16	0.996
	3	4	14	4	3	1.3	0	64.09	5.69	7.07	0.997
	3	4	15	2	2	1	0	0	9.04	12	0.998
	3	4	7	2	2	1	0	0	0	0	1.0
	3	4	14	3	3	1	0	146.07	6.67	13	0.99
	3	4	12	2	2	1	0	0	2	3	0.999
	4	4	8	2	2	1	0	0	0	0	1.0
	4	4	14	4	6	0.7	0	30.43	4.87	7.07	0.991
	4	4	13	4	3	1.3	0	459.57	9.39	20	0.996
	5	5	15	4	3	1.3	0	68.7	6.67	13	0.995
	5	5	15	5	4	1.3	0	446.66	13.26	23.02	0.994
	5	6	14	10	8	1.3	0	256.57	15.79	41.19	0.97
	6	5	15	12	11	1.1	0	24.8	34.17	71.18	0.971

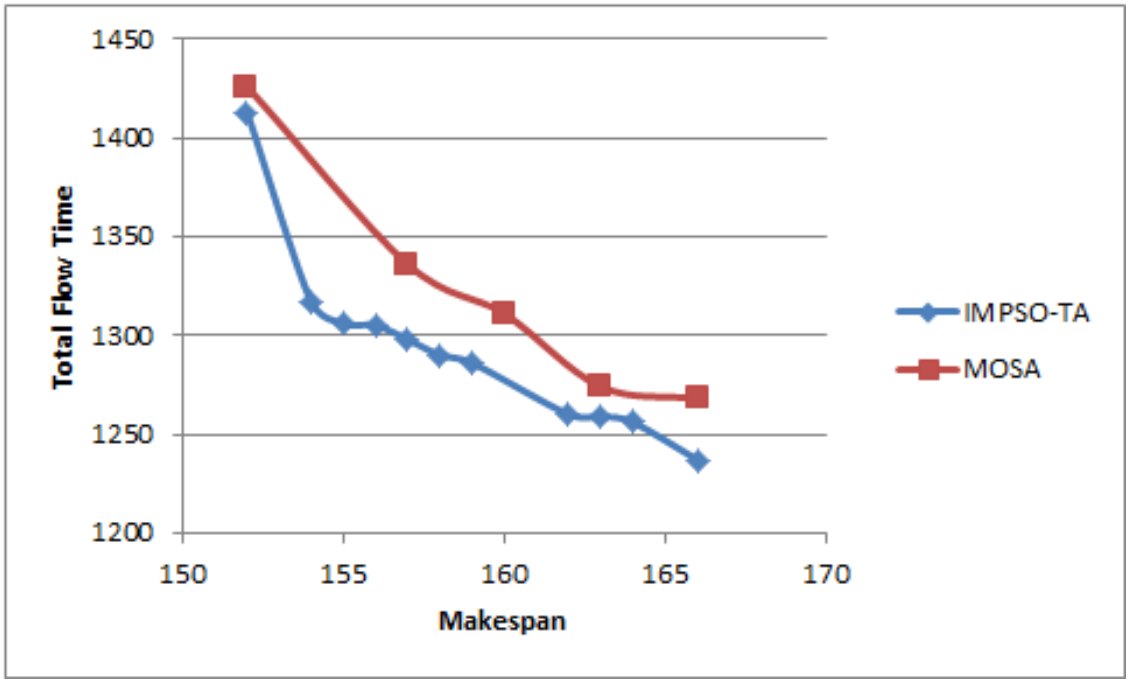

Figure 5.4: Pareto fronts obtained by MOSA and IMPSO-TA (SSU55)

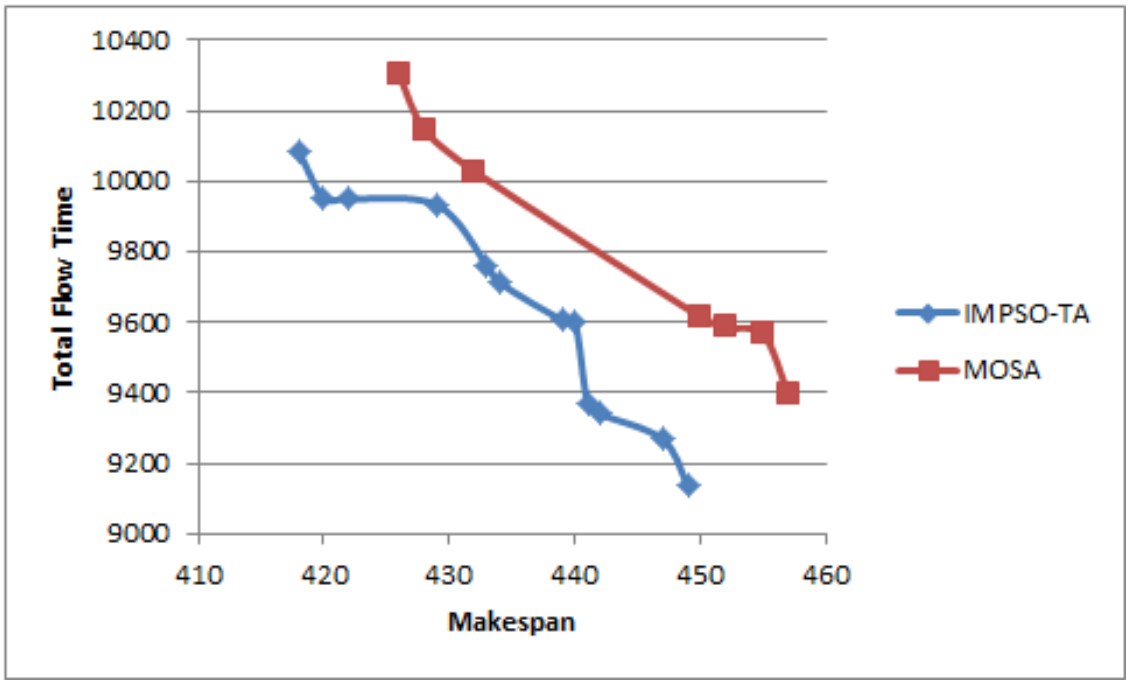

Figure 5.5: Pareto fronts obtained by MOSA and IMPSO-TA (LSU66)

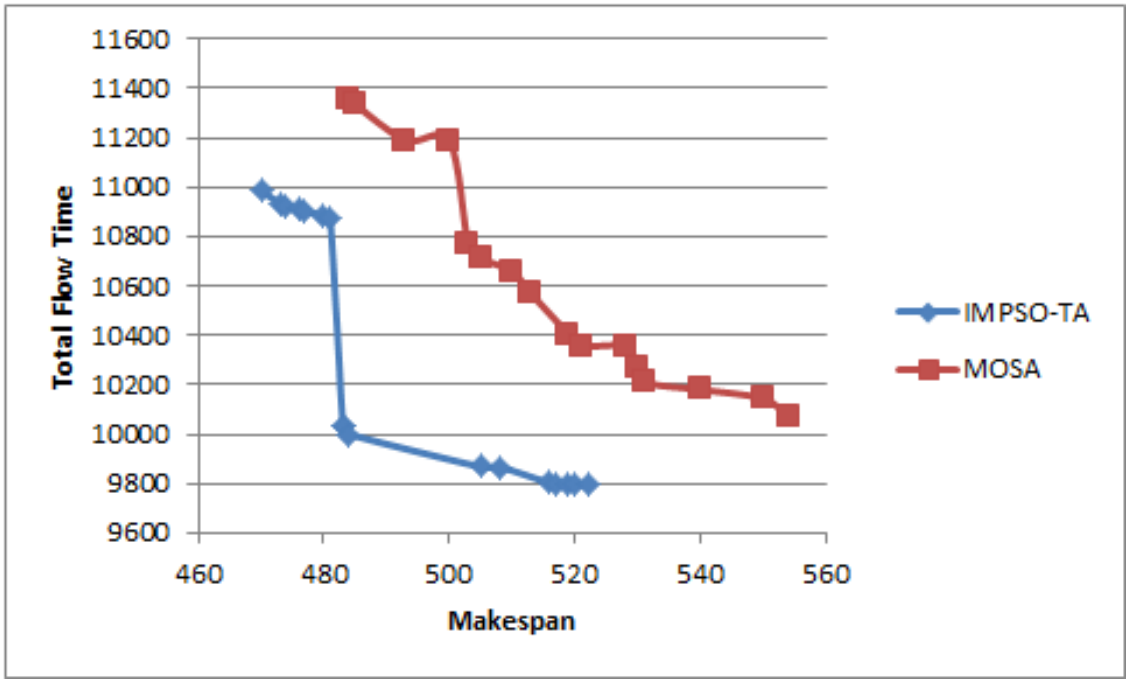

Figure 5.6: Pareto fronts obtained by MOSA and IMPSO-TA (LSU88)

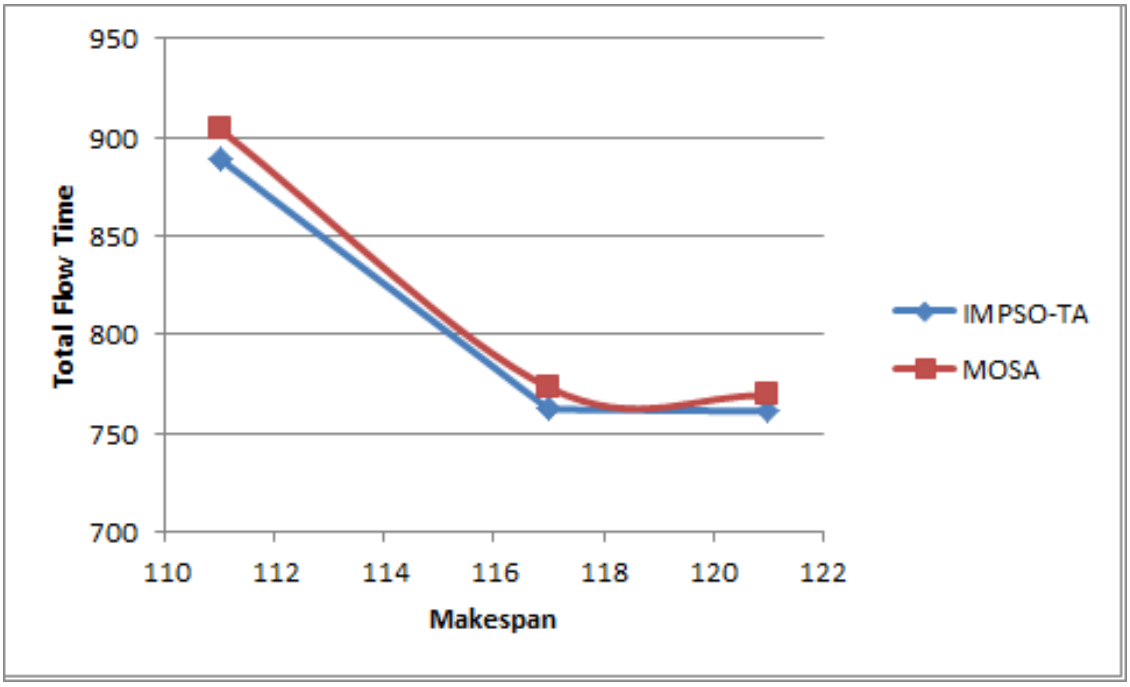

Figure 5.7: Pareto fronts obtained by MOSA and IMPSO-TA (SSU34)

5.11 Summary

In summary, the bi-objective scheduling problem is presented to minimize the total flow time and makespan simultaneously for a FMCSP with SDSTs in this chapter. Two meta-heuristics based on particle swarm optimization (MPSO and IMPSO-TA) and multi-objective simulated annealing (MOSA) are developed. The proposed algorithms are evaluated and tested using the well-known test problems developed by Schaller et al (2000). Furthermore, the performance of the proposed algorithms is evaluated to find the most efficient algorithm to cope with the proposed problem. Quality indicator measures are used to assess the best convergence and the diversity of the Pareto fronts obtained by the proposed algorithms. Results indicate that quality of Pareto fronts generated by IMPSO-TA is better than Pareto fronts found by MPSO and MOSA based on the test problems that are used in this research at the cost of CPU time. Further, the proposed IMPSO-TA performs as best available algorithms in the literature for small and medium test problems with a very minor deviation for best results for large test problems.

Chapter 6

Conclusion

6.1 Research Contributions

6.1.1 Scheduling a flowshop manufacturing cell with sequence-dependent family setup times

- Chapter 4 proposes several metaheuristic algorithms based on GA, SA, and PSO to minimize the TFT for a FMCSP with SDSTs

- A simple and efficient paradigm to identify the best combination of the SA parameters based on the Taguchi method is presented in this thesis. The following points are concluded:

    o Performance of SA algorithm is strongly affected by the probability function of

accepting solutions (cooling schedule). Using Cauchy function instead of the Boltzmann function (that is commonly used) for SA algorithm gives better solutions.

  - Reduction of the probability of accepting solutions (cooling rate) has significant effect on search. The best cooling rate found is the geometric rate.
  - Performance of SA algorithm is dependent on initial and final values of acceptance parameters (temperatures).
  - Number of iterations, and number of non-improving parameters are insignificant parameters compared with other SA parameters.

- Hybrid algorithms based on SA and PSO are proposed to balance between the intensification and diversification to improve the quality of the proposed algorithms.
  - Implementing SA algorithm as a local search combined with PSO leads to improve the quality of obtained solutions.

- The performance of proposed algorithms is evaluated based on Relative Percentage Deviation (RPD) from the lower bound developed by Salmasi et al (2010), and compared with best available algorithms ACO proposed by Salmasi [14], and GSA proposed by Naderi et al [20]. Results indicate that PSO-SA algorithm matches the best available algorithms for small and medium test problems. A very small deviation from the best algorithms is observed for large instances due to the high possibility to trap in local optima.

- A novel approach to evaluate the performance of metaheuristics based on robustness is presented in this study. Results conclude that PSO-SA is better than PSO even though all proposed algorithms have the same performance at the small-size problems (with less than five families). The hybrid metaheuristic PSO-SA algorithm is noticeably better than SA and GA. Moreover, according to the robustness of metaheuristics, PSO-SA solves tested problems with less variation from lower bounds than the other proposed algorithms. Further, PSO-SA has best performance at different levels of test problems. Therefore, PSO-SA is more robust than SA, GA, and PSO for solving FMCSP with SDSTs to minimize TFT based on the problems examined in this study.

6.1.2 Solving the Bi-criteria Scheduling Problem of Flowshop Manufacturing cell with Sequence Dependent Setup Times

- Chapter 5 proposes several metaheuristic algorithms based on MPSO, MOSA, and IMPSO-TA to minimize the total flow time and the makespan simultaneously.

- Approximated Pareto fronts are generated by proposed algorithms. Performance of these algorithms is evaluated based on the best available algorithms and using Quality Indicators (QI) measures.

    - An improved PSO is implemented to enhance the diversity and to improve searching ability of MPSO

    - A local search algorithm based on Threshold acceptance (TA) is combined with MPSO to improve the quality of the Pareto set

- Results show that IMPSO-TA, MPSO, and MOSA perform at the same level in solving the small and medium test problems. However, IMPSO-TA is superior to MPSO and MOSA in solving the large problems

6.2 Recommendations for Future Research

The suggestions for future research are grouped into two categories:

- Describing different research problems related to the proposed problem to cope with the real-world and industrial applications

    - o Relaxing one of the assumptions considered in this thesis:

      One of the proposed research directions can be extended by relaxing one or more of the used assumptions in this study. For example, relaxing the assumption of the permutation flowshop, or assuming different manufacturing environments such as job shop scheduling problem.

    - o Studying different performance measures:

      In this thesis, the completion time measures (TFT, and MS) were studied. Yet, there are other fruitful research directions to study other performance measures such as due date performance measures. Furthermore, rather than considering the minimization of TFT and MS as a bi-criteria optimization problem, multiple measures (more than two objectives) could be an interesting research topic.

- Developing different techniques to solve the proposed problems in this study

- Design metaheuristic algorithms

  As a result of the limited work on the proposed problem, there is room for designing more algorithms based on a pure metaheuristic algorithm or combination of different metaheuristics. For instance, Ant Colony Optimization (ACO) could be combined with a single solution based algorithm such as Variable Neighborhood Search (VNS).

- Develop an efficient lower bound

  One of the most interesting research directions in scheduling optimization problems is developing an efficient lower bound that could be used in the evaluation of any proposed algorithm. There is a room for applying different approaches to develop efficient lower bound for the minimization of the TFT

Bibliography

[1] F. Zhao, J. Tang, and J. Wang, "An improved particle swarm optimization with decline disturbance index (DDPSO) for multi-objective job-shop scheduling problem," *Comput. Oper. Res.*, vol. 45, pp. 38–50, may 2014.

[2] M. Zandieh, B. Dorri, and a. R. Khamseh, "Robust metaheuristics for group scheduling with sequence-dependent setup times in hybrid flexible flow shops," *Int. J. Adv. Manuf. Technol.*, vol. 43, pp. 767–778, sep 2008.

[3] B. E. Rumbold, J. a. Smith, J. Hurst, A. Charlesworth, and A. Clarke, "Improving productive efficiency in hospitals: findings from a review of the international evidence," *Heal. Econ. Policy Law*, vol. 10, pp. 21–43, aug 2014.

[4] S. Heragu, *Group technology and cellular manufacturing A state-of-art synthesis of research and practice*. Kluwer Academic Pub, 1995.

[5] M. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer, 2012.

[6] S. Lin, K. Ying, and Z. Lee, "Metaheuristics for scheduling a non-permutation flowline manufacturing cell with sequence dependent family setup times," *Comput. Oper. Res.*,

vol. 36, pp. 1110–1121, apr 2009.

[7] H. Hendizadeh, T. Elmekkawy, and G. Wang, "Bi-criteria scheduling of a flowshop manufacturing cell with sequence dependent setup times," *Eur. J. Ind. Eng.*, vol. 1, no. 4, pp. 1751–5254, 2007.

[8] S. W. Lin and K. C. Ying, "Scheduling a bi-criteria flowshop manufacturing cell with sequence-dependent family setup times," *Eur. J. Ind. Eng.*, vol. 6, no. 4, pp. 474–495, 2012.

[9] H. G. Campbell, R. a. Dudek, and M. L. Smith, "A Heuristic Algorithm for the n Job, m Machine Sequencing Problem," *Manage. Sci.*, vol. 16, pp. B–630–B–637, jun 1970.

[10] M. Nawaz, E. Enscore Jr, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[11] J. Schaller, "A comparison of heuristics for family and job scheduling in a flow-line manufacturing cell," *Int. J. Prod. Res.*, vol. 38, no. April 1999, pp. 287–308, 2000.

[12] G. Celano, A. Costa, and S. Fichera, "Constrained scheduling of the inspection activities on semiconductor wafers grouped in families with sequence-dependent set-up times," *Int. J. Adv. Manuf. Technol.*, vol. 46, pp. 695–705, jun 2010.

[13] C. a. Gelogullari and R. Logendran, "Group-scheduling problems in electronics manufacturing," *J. Sched.*, vol. 13, pp. 177–202, dec 2010.

[14] N. Salmasi, R. Logendran, and M. R. Skandari, "Total flow time minimization in a flowshop sequence-dependent group scheduling problem," *Comput. Oper. Res.*, vol. 37, pp. 199–212, jan 2010.

[15] S. Hamed Hendizadeh, H. Faramarzi, S. Mansouri, J. N. Gupta, and T. Y ElMekkawy, "Meta-heuristics for scheduling a flowline manufacturing cell with sequence dependent family setup times," *International Journal of Production Economics*, vol. 111(2), pp. 593–605, 2008.

[16] H. M. Hsu, Y. Hsiung, Y. Z. Chen, and M. C. Wu, "A GA methodology for the scheduling of yarn-dyed textile production," *Expert Syst. Appl.*, vol. 36, no. 10, pp. 12095–12103, 2009.

[17] W. L. Pearn, S. H. Chung, and C. M. Lai, "Scheduling integrated circuit assembly operations on die bonder," *IEEE Trans. Electron. Packag. Manuf.*, vol. 30, no. 2, pp. 97–105, 2007.

[18] M. Bevilacqua, F. Ciarapica, I. De Sanctis, G. Mazzuto, and C. Paciarotti, "A Changeover Time Reduction through an integration of lean practices: a case study from pharmaceutical sector," *Assem. Autom.*, vol. 35, pp. 22–34, feb 2015.

[19] A. Panwar, R. Jain, and A. Rathore, "Lean implementation in Indian process industries: some empirical evidence," *J. Manuf. Technol. Manag.*, vol. 25, no. 1, pp. 131–160, 2015.

[20] B. Naderi and N. Salmasi, "Permutation flowshops in group scheduling with sequence-dependent setup times," *Eur. J. Ind. Eng.*, vol. 6, no. 2, pp. 177–198, 2012.

[21] I. H. Osman and G. Laporte, "Metaheuristics: A bibliography," *Ann. Oper. Res.*, vol. 63, pp. 516–623, 1996.

[22] C. Shiyas and V. Madhusudanan Pillai, "Cellular manufacturing system design using grouping efficacy-based genetic algorithm," *Int. J. Prod. Res.*, pp. 1–14, jan 2014.

[23] S. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Nav. Res. Logist. Q.*, vol. 1, no. 1, pp. 61–68, 1954.

[24] A. Allahverdi, J. N. D. Gupta, and T. Aldowaisan, "A review of scheduling research involving setup considerations," *Omega*, vol. 27, pp. 219–239, apr 1999.

[25] A. Allahverdi and H. M. Soroush, "The significance of reducing setup times / setup costs," *Eur. J. Oper. Res.*, vol. 187, pp. 978–984, 2008.

[26] J. Gupta, "Flowshop schedules with sequence dependent setup times," *J. Oper. Res. Soc. Japan*, vol. 29, no. 3, pp. 206–219, 1986.

[27] M. Garey, D. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, 1976.

[28] J. Schaller, J. N. D. Gupta, and A. J. Vakharia, "Scheduling a flowline manufacturing cell with sequence dependent family setup times," *Eur. J. Oper. Res.*, vol. 125, pp. 324–339, sep 2000.

[29] P. Franca, J. Gupta, A. Mendes, P. Moscato, and K. Veltink, "Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups," *Comput. Ind. Eng.*, vol. 48(3), pp. 491–506, 2005.

[30] S.-W. Lin, J. N. Gupta, K.-C. Ying, and Z.-J. Lee, "Using simulated annealing to schedule a flowshop manufacturing cell with sequence-dependent family setup times," *Int. J. Prod. Res.*, vol. 47, pp. 3205–3217, jun 2009.

[31] S.-W. Lin, K.-C. Ying, C.-C. Lu, and J. N. Gupta, "Applying multi-start simulated annealing to schedule a flowline manufacturing cell with sequence dependent family setup times," *Int. J. Prod. Econ.*, vol. 130, pp. 246–254, apr 2011.

[32] K. Ying, J. Gupta, S. Lin, and Z. Lee, "Permutation and non-permutation schedules for the flowline manufacturing cell with sequence dependent family setups," *Int. J. ...*, no. July 2012, pp. 2169–2184, 2010.

[33] R. Logendran, P. Deszoeke, and F. Barnard, "Sequence-dependent group scheduling problems in flexible flow shops," *Int. J. Prod. Econ.*, vol. 102, pp. 66–86, jul 2006.

[34] N. Salmasi, R. Logendran, and M. R. Skandari, "Makespan minimization of a flowshop sequence-dependent group scheduling problem," *Int. J. Adv. Manuf. Technol.*, feb 2011.

[35] N. Salmasi, D. Davarnia, and R. Logendran, "A tight lower bound for makespan minimization sequence dependent flowshop group scheduling problems," *2009 Int. Conf. Comput. Ind. Eng.*, pp. 86–89, jul 2009.

[36] R. Bouabda, B. Jarboui, and A. Rebai, "A nested iterated local search algorithm for scheduling a flowline manufacturing cell with sequence dependent family setup times," *(LOGISTIQUA), 2011 4th*, pp. 526–531, 2011.

[37] J. Neufeld, J. Gupta, and U. Buscher, "Minimising makespan in flowshop group scheduling with sequence-dependent family set-up times using inserted idle times," *Int. J. Prod. Res.*, pp. 1–16, sep 2014.

[38] A.-m. Ibrahem, T. Elmekkawy, and Q. Peng, "Robust Metaheuristics for Scheduling Cellular Flowshop with Family Sequence-Dependent Setup Times," *Procedia CIRP*, vol. 17, pp. 428–433, 2014.

[39] Y. Li, X. Li, and J. N. Gupta, "Solving the multi-objective flowline manufacturing cell scheduling problem by hybrid harmony search," *Expert Syst. Appl.*, sep 2014.

[40] J. Schaller, "A new lower bound for the flow shop group scheduling problem," *Comput. Ind. Eng.*, vol. 41, pp. 151–161, nov 2001.

[41] J. Shedden, *An evaluation of a modified simulated annealing algorithm for various formulations*. PhD thesis, Arizona state University, 1990.

[42] J. Schaller, "An improved branch and bound procedure for scheduling a flow line manufacturing cell," *Int. J. Prod. Res.*, vol. 43, pp. 4697–4720, nov 2005.

[43] G. Raidl, "A unified view on hybrid metaheuristics," *Proc. HM 2006 3rd Int. Work. Hybrid Metaheuristics, Lect. Notes Comput. Sci. Springer-Verlag, Berlin, Ger.*, vol. 4030, pp. 1–12, 2006.

[44] C. Blum, A. Roli, and M. Sampels, *Hybrid Metaheuristics*, vol. 114 of *Studies in Computational Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[45] E.-G. Talbi, *Metaheuristics: from design to implementation*. JohnWiley , Sons, Inc., Hoboken, New Jersey, 2009.

[46] E.-G. Talbi, *Hybrid Metaheuristics*, vol. 434 of *Studies in Computational Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[47] E. Talbi, "A taxonomy of hybrid metaheuristics," *J. heuristics*, pp. 541–564, 2002.

[48] K. Haley and C. Sparks, "A comparative study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in three scheduling problems," *Eur. J. Oper. Res.*, vol. 83, pp. 330–346, jun 1995.

[49] D. Sung Lee, V. S. Vassiliadis, and J. M. Park, "A novel threshold accepting metaheuristic for the job-shop scheduling problem," *Comput. Oper. Res.*, vol. 31, pp. 2199–2213, nov 2004.

[50] E. Aarts, "A computational study of local search algorithms for job shop scheduling," *ORSA J. Comput.*, vol. 6, no. 2, pp. 118–125, 1994.

[51] E. Aarts and J. Lenstra, *Local search in combinatorial optimization*. John Wiley & Sons, Inc., 2003.

[52] G. Dueck and T. Scheuer, "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing," *J. Comput. Phys.*, 1990.

[53] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simmulated annealing," *Science*, pp. 671–680, 1983.

[54] A. Khosla, S. Kumar, and K. Aggarwal, "Identification of strategy parameters for particle swarm optimizer through Taguchi method," *J. Zhejiang Univ. ...*, vol. 12, pp. 1989–1994, 2006.

[55] G. Taguchi, "Quality engineering (Taguchi methods) for the development of electronic circuit technology," *IEEE Trans. Reliab.*, vol. 44, pp. 225–229, jun 1995.

[56] G. Taguchi, S. Chowdhury, and Y. Wu, *Taguchi's quality engineering handbook*. John Wiley & Sons, Inc., 2005.

[57] B. Naderi, "An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness," *Expert Syst. Appl.*, vol. 36, pp. 9625–9633, aug 2009.

[58] M. Phadke, *Quality engineering using robust design*. Prentice-Hall, 1989.

[59] B.-W. Cheng and C.-L. Chang, "A study on flowshop scheduling problem combining Taguchi experimental design and genetic algorithm," *Expert Syst. Appl.*, vol. 32, pp. 415–421, feb 2007.

[60] L.-Y. Tseng and Y.-T. Lin, "A hybrid genetic local search algorithm for the permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 198, pp. 84–92, oct 2009.

[61] L.-Y. Tseng and Y.-T. Lin, "A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem," *Int. J. Prod. Econ.*, vol. 127, pp. 121–128, sep 2010.

[62] A. R. Yildiz, "Hybrid Taguchi-differential evolution algorithm for optimization of multi-pass turning operations," *Appl. Soft Comput.*, vol. 13, pp. 1433–1439, mar 2013.

[63] C. Yang, J. Chou, and C. Chang, "Hybrid Taguchi-based genetic algorithm for flowshop scheduling problem," *Int. J. Innov. Comput. Inf. Control*, vol. 9, no. 3, pp. 1045–1063, 2013.

[64] H. Seidgar, M. Abedi, S. Tadayonirad, and H. Fazlollahtabar, "A hybrid particle swarm optimisation for scheduling just-in-time single machine with preemption, machine idle time and unequal release times," *Int. J. Prod. Res.*, pp. 1–24, oct 2014.

[65] Y. Jin and B. Sendhoff, "Trade-off between performance and robustness: an evolutionary multiobjective approach," *Evol. Multi-Criterion Optim.*, vol. 2632, pp. 237–251, 2003.

[66] D. Montgomery, *Design and analysis of experiments*. John Wiley&Sons. Inc., 2008.

[67] T. Bartz-Beielstein, *Experimental research in evolutionary computation: the new experimentalism*. Springer, 2006.

[68] H.-G. Beyer and B. Sendhoff, "Robust optimization A comprehensive survey," *Comput. Methods Appl. Mech. Eng.*, vol. 196, pp. 3190–3218, jul 2007.

[69] H. Samarghandi, T. Y. Elmekkawy, and A.-m. M. Ibrahem, "Studying the effect of different combinations of timetabling with sequencing algorithms to solve the no-wait job shop scheduling problem," *Int. J. Prod. Res.*, vol. 51-16, pp. 4942–4965, 2013.

[70] E. Zitzler and L. Thiele, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 117–132, 2003.

[71] N. Salmasi and R. Logendran, "A Lower Bounding Approach for Sequence-Dependent Group Scheduling Problems," in *IIE Annu. Conf. Proc.*, pp. 1–7, 2004.

[72] M. Varmazyar and N. Salmasi, "Minimizing the Number of Tardy Jobs in Flow Shop Sequence Dependent Setup Times Scheduling Problem," *Appl. Mech. Mater.*, 2012.

[73] T. Keshavarz and N. Salmasi, "Makespan minimisation in flexible flowshop sequence-dependent group scheduling problem," *Int. J. Prod. Res.*, pp. 1–12, aug 2013.

[74] G. Zäpfel, R. Braune, and M. Bögl, "Metaheuristic Search Concepts," *Production*, pp. 121–143, 2010.

[75] R. Bouabda, B. Jarboui, M. Eddaly, and A. Rebai, "A branch and bound enhanced genetic algorithm for scheduling a flowline manufacturing cell with sequence dependent family setup times," *Comput. Oper. Res.*, vol. 38, no. 1, pp. 387–393, 2011.

[76] M. Eddaly, B. Jarboui, R. Bouabda, and A. Rebai, "Hybrid estimation of distribution algorithm for permutation flowshop scheduling problem with sequence dependent family setup times," in *Comput. Ind. Eng. 2009. CIE 2009. Int. Conf.*, 2009 International

Conference on Computers and Industrial Engineering, CIE 2009, (Troyes, France), pp. 217–220, IEEE, 2009.

[77] Y. Zhang, X. Li, and Q. Wang, "Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization," *Eur. J. Oper. Res.*, vol. 196, pp. 869–876, aug 2009.

[78] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proc. ICNN'95 - Int. Conf. Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[79] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intell.*, vol. 1, pp. 33–57, aug 2007.

[80] B. Liu, L. Wang, and Y.-H. Jin, "An effective PSO-based memetic algorithm for flow shop scheduling.," *IEEE Trans. Syst. Man. Cybern. B. Cybern.*, vol. 37, pp. 18–27, mar 2007.

[81] Z. Lian, X. Gu, and B. Jiao, "A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan," *Chaos, Solitons & Fractals*, vol. 35, pp. 851–861, mar 2008.

[82] C.-T. Tseng and C.-J. Liao, "A particle swarm optimization algorithm for hybrid flow-shop scheduling with multiprocessor tasks," *Int. J. Prod. Res.*, vol. 46, pp. 4655–4670, sep 2008.

[83] I.-H. Kuo, S.-J. Horng, T.-W. Kao, T.-L. Lin, C.-L. Lee, T. Terano, and Y. Pan, "An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model," *Expert Syst. Appl.*, vol. 36, pp. 7027–7032, apr 2009.

[84] K. Rameshkumar, R. Suresh, and K. Mohanasundaram, "Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan," *Adv. Nat. Comput.*, pp. 572–581, 2005.

[85] T.-L. Lin, S.-J. Horng, T.-W. Kao, Y.-H. Chen, R.-S. Run, R.-J. Chen, J.-L. Lai, and I.-H. Kuo, "An efficient job-shop scheduling algorithm based on particle swarm optimization," *Expert Syst. Appl.*, vol. 37, pp. 2629–2636, mar 2010.

[86] H. Liu, L. Gao, and Q. Pan, "A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem," *Expert Syst. Appl.*, vol. 38, pp. 4348–4360, apr 2011.

[87] S. Gohari and N. Salmasi, "Flexible flowline scheduling problem with constraints for the beginning and terminating time of processing of jobs at stages," *Int. J. Comput. Integr. Manuf.*, pp. 1–14, 2014.

[88] B. Liu, L. Wang, and Y.-H. Jin, "An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers," *Comput. Oper. Res.*, vol. 35, pp. 2791–2806, sep 2008.

[89] S. Ledesma, G. Aviña, and R. Sanchez, "Practical considerations for simulated annealing implementation," *Simulated Annealing*, no. February, pp. 401–420, 2008.

[90] R. W. Eglese, "Simulated annealing: a tool for operational research," *Eur. J. Oper. Res.*, vol. 46, pp. 271–281, 1990.

[91] R. Rutenbar, "Simulated annealing algorithms: An overview," *Circuits Devices Mag. IEEE*, no. x, 1989.

[92] F. Busetti, "Simulated annealing overview," *World Wide Web URL www. geocities. com/ DOI10.1.1.66.5018*, 2003.

[93] A. J. Vakharia and Y.-L. Chang, "A simulated annealing approach to scheduling a manufacturing cell," *Nav. Res. Logist.*, vol. 37, pp. 559–577, aug 1990.

[94] J. Sridhar and C. Rajendran, "Scheduling in a cellular manufacturing system: a simulated annealing approach," *Int. J. Prod. Res.*, vol. 31, pp. 2927–2945, dec 1993.

[95] S. A. Mansouri, S. H. Hendizadeh, and N. Salmasi, "Bicriteria scheduling of a two-machine flowshop with sequence-dependent setup times," *Int. J. Adv. Manuf. Technol.*, vol. 40, pp. 1216–1226, mar 2009.

[96] C. Coello, G. Lamont, and D. V. Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.

[97] E. Zitzler, *Evolutionary algorithms for multiobjective optimization: Methods and applications*. PhD thesis, Swiss Federal Institute of Technology Zurich, 1999.

[98] K. Deb, *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*. London: Springer London, 2011.

[99] J. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proceedings of an Interna- tional Conference on Genetic Algorithms and Their Applications*, pp. 93–100, 1985.

[100] F. Kursawe, "A variant of evolution strategies for vector optimization," *Parallel Probl. Solving from Nat.*, pp. 193–197, 1991.

[101] C. M. Fonseca and P. J. Fleming, "Unknown title," in *Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California*, pp. 416–426, 1983.

[102] J. Horn, N. Nafpliotis, and D. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization," *Evol. Comput. 1994. IEEE , Proceeding first IEEE Conf.*, pp. 82–87, 1994.

[103] N. Srinivas and K. Deb, "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, 1994.

[104] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithmsA comparative case study," *Parallel Probl. Solving from NaturePPSN V*, pp. 292–301, 1998.

[105] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: empirical results.," *Evol. Comput.*, vol. 8, pp. 173–195, jan 2000.

[106] K. Sindhya, *Hybrid Evolutionary Multi-Objective Optimization with Enhanced Convergence and Diversity Karthik Sindhya Hybrid Evolutionary Multi-Objective Optimiza-*

tion with Enhanced Convergence and Diversity. PhD thesis, University of Jyväskylä, Finland, 2011.

[107] X. Wang and L. Tang, "Scheduling a single machine with multiple job processing ability to minimize makespan," *J. Oper. Res. Soc.*, vol. 62, pp. 1555–1565, jul 2011.

[108] Q.-K. Pan, L. Wang, and Q. B, "A novel multi-objective particle swarm optimization algorithm for no-wait flow shop scheduling problems," *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.*, vol. 222, pp. 519–539, apr 2008.

[109] S. Chandrasekaran, S. G. Ponnambalam, R. K. Suresh, and N. Vijayakumar, "multi-objective particle swarm optimization for scheduling in flowshop to minimize makespan, total flow time and competion time varianvce," *IEEE Congr. Evol. Comput. (CEC 2007)*, pp. 4012–4018, 2007.

[110] W. Leong and G. Yen, "PSO-based multiobjective optimization with dynamic population size and adaptive local archives," *Syst. Man, Cybern. Part B . . .*, vol. 38, no. 5, pp. 1270–1293, 2008.

[111] J. Durillo, J. García-Nieto, and A. Nebro, "Multi-objective particle swarm optimizers: An experimental comparison," *Evol. Optim. , Springer*, pp. 495–509, 2009.

[112] L. Tang and X. Wang, "An Improved Particle Swarm Optimization Algorithm for the Hybrid Flowshop Scheduling to Minimize Total Weighted Completion Time in Process Industry," *IEEE Transaction on Control System Technology*, vol. 18, no. 6, pp. 1303–1314, 2010.

[113] X. Wang and L. Tang, "A PSO-Based Hybrid Multi-Objective Algorithm for Multi-Objective Optimization Problems," *Adv. Swarm Intell.*, pp. 26–33, 2011.

[114] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," *Eur. J. Oper. Res.*, vol. 177, pp. 1930–1947, mar 2007.

[115] A. Elhossini, S. Areibi, and R. Dony, "Strength Pareto particle swarm optimization and hybrid EA-PSO for multi-objective optimization.," *Evolutionary computation*, vol. 18, pp. 127–56, jan 2010.

[116] D. Simon, *Evolutionary optimization algorithms*. John Wiley & Sons, Inc., 2013.

[117] S. A. Mansouri, "Coordination of set-ups between two stages of a supply chain using multi-objective genetic algorithms," *Int. J. Prod. Res.*, vol. 43, pp. 3163–3180, aug 2005.

[118] B. Suman and P. Kumar, "A survey of simulated annealing as a tool for single and multiobjective optimization," *J. Oper. Res. Soc.*, vol. 57, pp. 1143–1160, oct 2005.

[119] P. Czyzzak and A. Jaszkiewicz, "Pareto simulated annealinga metaheuristic technique for multiple-objective combinatorial optimization," *J. Multi-criteria Decis. Anal.*, vol. 7, no. February 1997, pp. 34–47, 1998.

[120] T. Varadharajan and C. Rajendran, "A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs," *Eur. J. Oper. Res.*, vol. 167, pp. 772–795, dec 2005.

[121] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Trans. Syst. Man Cybern. Part C (Applications Rev.*, vol. 28, no. 3, pp. 392–403, 1998.

[122] J. Knowles and D. Corne, "Memetic algorithms for multiobjective optimization: issues, methods and prospects," *Recent Adv. memetic algorithms*, 2005.

[123] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, 1999.

[124] E. Zitzler, M. Laumanns, and L. Thiele, "Why quality assessment of multiobjective optimizers is difficult," *Proc. Genet. Evol. Comput. Conf.*, pp. 666–674, 2002.

[125] E. Zitzler, J. Knowles, and L. Thiele, "Quality assessment of pareto set approximations," *Multiobjective Optim.*, pp. 373–404, 2008.

[126] K. Tan, C. Goh, Y. Yang, and T. Lee, "Evolving better population distribution and exploration in evolutionary multi-objective optimization," *Eur. J. Oper. Res.*, vol. 171, pp. 463–495, jun 2006.

Appendix A

Quality Indicators code

```
#include "stdafx.h"
#include<stdio.h>
#include <iostream>
#include<vector>
#include<math.h>
#include<fstream>
#include<string>
#include<sstream>
#include "QualityMeasures.h"

using namespace std;
vector<int> TFT_PSO, MS_PSO;
vector<int> TFT_PSO_SA, MS_PSO_SA;
void Read_PSO_Data(string filename);
void Read_PSO_SA_Data(string filename);
```

```
void Read_MOGA_Data(string filename);


bool dominated_PSO(float TFT, float MS);
float Number_NDS_Ratio ();
float calculate_coverage();
float D_average();
float D_Max();
float D_Spacing();
float QI_measure();
float find_min(vector<float> , short sizeofarray);
int find_min(vector<int> , short sizeofarray); // integer version
float find_max(vector<float> , short sizeofarray);


int _tmain(int argc, _TCHAR* argv[])
{
        ofstream outputfile;


        outputfile.open("Quality_Measures.csv",ofstream::app);


//inputfiles


        Read_PSO_Data("NDS_SSu1010_MOSA8.csv");
        Read_PSO_SA_Data("NDS_SSu1010_IMPSO8.csv");


cout<<endl;
        cout<<"LSU_56,"<<"GT15////Large_"<<endl;
```

```
        cout<<endl;
        // Seven Quality Measures:....

        float No_NDSs_and_Ratio_C= Number_NDS_Ratio ();
//1. How many NDSs generated by algorithm, and the Ratio between them
        float coverage = calculate_coverage();
//3. coveargе index
        float D_ave= D_average();
//4. D max
        float Dmax= D_Max();
//5. D average
        float DistanceSpacing = D_Spacing();
//6. Spacing
        float QualityIndicator =QI_measure();
//7.  Quality Index

        //Store output in a csv file
        outputfile<<"8- Large,"<<"Large "<<endl;
        outputfile<<endl;
        outputfile<<"NDS_PSO,"<<TFT_PSO.size()<<endl;
        outputfile<<"NPSO_SA ,"<<TFT_PSO_SA.size()<<endl;
        outputfile<<"R_NDSs,"<<No_NDSs_and_Ratio_C<<endl;
        outputfile<<"C_Index, "<<coverage<<endl;
        outputfile<<"D_spacing, "<<DistanceSpacing<<endl;
        outputfile<<"D ave., "<<D_ave<<endl;
        outputfile<<"D_Max,  "<<Dmax<<endl;
```

```
        outputfile<<"QI measure ,  "<<QI_measure()<<endl;
        outputfile<<" \n Next Pro"<<endl;
        outputfile<<endl;
        outputfile.close();

        return 0;
}

//The functions used to calculate Quality Measures
//1, and 2: This gives the number of each NDS set and the Ratio between them

float Number_NDS_Ratio ()
{
        cout<<" NDS_PSO  ...."<<TFT_PSO.size()<<endl;
        cout<< "NDS_PSO_SA ...."<<TFT_PSO_SA.size()<<endl;
        float C_ratio = ( (float)TFT_PSO.size() / (float)TFT_PSO_SA.size());
        cout<<"Ratio NDS1_NDS2  "<<C_ratio<<endl;
        return C_ratio;

}

// 3- converges

float calculate_coverage()
{
        short count = 0;
```

```
        for(short i=0; i<TFT_PSO_SA.size();i++)
        {
                if(dominated_PSO(TFT_PSO_SA[i], MS_PSO_SA[i]) == true)
                        count++;
        }
        float coverage = (float)(count / TFT_PSO_SA.size());
        cout<<"Coverage Index, ="<<coverage<<endl;
        return coverage;
}

        //4 D average

float D_average()
{

        float d_av = 0.0;
        float val1, val2, temp1, temp2;
        vector<float> distances, distance_maxes;
        for(short i = 0;i<TFT_PSO_SA.size();i++)
        {
                distances.clear();
                for(short j=0;j<TFT_PSO.size();j++)
                {
                        val1 = pow((TFT_PSO_SA[i] - TFT_PSO[j]),2.0);
                        val2 = pow((MS_PSO_SA[i] - MS_PSO[j]),2.0);
```

```
                        distances.push_back(sqrt((float)val1 + (float)val2));
                }
                d_av += find_min(distances,distances.size());
                distance_maxes.push_back(find_min(distances,distances.size()));
        }
        d_av = d_av/(float)TFT_PSO_SA.size();
//float d_max = find_max(distance_maxes, distance_maxes.size());
        cout<<"D_average_is_"<<d_av<< endl;
        //cout<<" D max is "<<d_max<<endl;

        return d_av;
}

        // 5 D maximum .....

float D_Max()
{
        float d_av = 0.0;
        float val1, val2, temp1, temp2;
        vector<float> distances, distance_maxes;
        for(short i = 0;i<TFT_PSO_SA.size();i++)
        {
                distances.clear();
                for(short j=0;j<TFT_PSO.size();j++)
                {
                        val1 = pow((TFT_PSO_SA[i] - TFT_PSO[j]),2.0);
```

```
                        val2 = pow((MS_PSO_SA[i] - MS_PSO[j]),2.0);
                        distances.push_back(sqrt((float)val1 + (float)val2));
                }
                d_av += find_min(distances,distances.size());
                distance_maxes.push_back(find_min(distances,distances.size()));
        }
        //d_av = d_av/(float)TFT_PSO_SA.size();
        float d_max = find_max(distance_maxes, distance_maxes.size());
        //cout<<"D average is "<<d_av<< endl;
        cout<<" D_max is "<<d_max<<endl;
        return d_max;

}
        // 6 Spacing measure

float D_Spacing()
{
        vector<float> temp_vec, distance_spaces;
        float val1, val2, temp1, temp2;
        float spacing = 0.0;
        for(short i=0;i<TFT_PSO_SA.size();i++)
        {
                for(short j=0;j<TFT_PSO_SA.size();j++)
                {
                        if(j==i)
                                continue;
```

```
                    val1 = abs(TFT_PSO_SA[i] - TFT_PSO_SA[j]) +
                    abs(MS_PSO_SA[i] - MS_PSO_SA[j]);
                    temp_vec.push_back(val1);
            }
        distance_spaces.push_back(find_min(temp_vec,temp_vec.size()));
            temp_vec.clear();
        }

        float d_bar = 0.0;
        for(short i=0;i<distance_spaces.size();i++)
        {
            d_bar += distance_spaces[i];
        }
        d_bar = (float)d_bar/(float)distance_spaces.size();

        for(short i=0;i<TFT_PSO_SA.size();i++)
        {
            spacing += pow(distance_spaces[i]-d_bar,2);
        }
        spacing = spacing / (TFT_PSO_SA.size()-1);
        spacing = sqrt(spacing);

        cout<<"Spacing = "<<spacing<<endl;

        return spacing;
}
```

```
        //7, the QI_measure
float QI_measure()
{
        int TFT_min = find_min(TFT_PSO_SA,TFT_PSO_SA.size());
        int MS_min = find_min(MS_PSO_SA,MS_PSO_SA.size());
        float QI_measure = 0.0;
        float temp1, temp2;
        for(short i=0;i<TFT_PSO.size();i++)
        {
                temp1 = (float)(TFT_PSO[i]-TFT_min)/((float)TFT_min);
                temp2 = (float)(MS_PSO[i]-MS_min)/((float)MS_min);
                QI_measure += (1-min(temp1, temp2));
        }
        QI_measure = QI_measure / (float)TFT_PSO.size();

        cout<<"QI_measure = "<<QI_measure<<endl;
    outputfile<<"QI_measure,  "<<QI_measure<<endl;
        return QI_measure;
        }

// function for Reading inputs (NDS sets)

void Read_PSO_Data(string filename)
{
        string buffer, temp;
```

```
        ifstream in;
        in.open(filename);
        int m;
        short comma_index;
        while(!in.eof())
        {
                getline(in,buffer);
                comma_index = buffer.find_first_of(",");
                temp = buffer.substr(0,comma_index);
                m = strtod(temp.data(), NULL);
                if(m != 0)
                        TFT_PSO.push_back(m);
                temp = buffer.substr(comma_index+1, buffer.size());
                m = strtod(temp.data(), NULL);
                if(m != 0)
                        MS_PSO.push_back(m);
        }
        in.close();

}
void Read_PSO_SA_Data(string filename)
{
        string buffer, temp;
        ifstream in;
        in.open(filename);
        int m;
```

```
        short comma_index;
        while(!in.eof())
        {
                getline(in,buffer);
                comma_index = buffer.find_first_of(",");
                temp = buffer.substr(0,comma_index);
                m = strtod(temp.data(), NULL);
                if(m != 0)
                        TFT_PSO_SA.push_back(m);
                temp = buffer.substr(comma_index+1, buffer.size());
                m = strtod(temp.data(), NULL);
                if(m != 0)
                        MS_PSO_SA.push_back(m);
        }
        in.close();

}

void Read_MOGA_Data(string filename)
{
        string buffer, temp;
        ifstream in;
        in.open(filename);
        int m;
        short comma_index;
        while(!in.eof())
```

```
        {
                getline(in,buffer);
                comma_index = buffer.find_first_of(",");
                temp = buffer.substr(0,comma_index);
                m = strtod(temp.data(), NULL);
                if(m != 0)
                        TFT_MOGA.push_back(m);
                temp = buffer.substr(comma_index+1, buffer.size());
                m = strtod(temp.data(), NULL);
                if(m != 0)
                        MS_MOGA.push_back(m);
                // to insert MS into another vector
        }
        in.close();
}


// Domination check ......
bool dominated_PSO(float TFT, float MS)
{
        short length = TFT_PSO.size();
        for(short i=0;i<length;i++)
        {
                if((TFT_PSO[i] < TFT) && (MS_PSO[i] < MS))
                        return true;
        }
        return false;
```

```
}


// Finding the minimum function
float find_min(vector<float> array1, short sizeofarray)
{
        float min=100000.0,index=0;
        for (int i=0;i<sizeofarray;i++)
        {
                if (array1[i]<min)
                {
                        min=array1[i];
                        index = i;
                }
        }
        return min;
}


// finding min for intergers
int find_min(vector<int> array1, short sizeofarray)
{
        int min=100000,index=0;
        for (int i=0;i<sizeofarray;i++)
        {
                if (array1[i]<min)
                {
                        min=array1[i];
```

```
                        index = i;
                }
        }
        return min;
}

float find_max(vector<float> array1, short sizeofarray)
{
        float max=-100000.0,index=0;
        for (int i=0;i<sizeofarray;i++)
        {
                if (array1[i]>max)
                {
                        max=array1[i];
                        index = i;
                }
        }
        return max;
}
```