INTELLIGENT MANUFACTURING FOR PRODUCTION PLANNING BASED UPON HIERARCHICALLY COUPLED CONSTRAINED AND MULTIMODAL OPTIMIZATION

A Dissertation
Presented to
The Academic Faculty

by

Manik Rajora

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
George W. Woodruff School of Mechanical Engineering

Georgia Institute of Technology
May 2018

INTELLIGENT MANUFACTURING FOR PRODUCTION PLANNING BASED UPON HIERARCHICALLY COUPLED CONSTRAINED AND MULTIMODAL OPTIMIZATION

Approved by:

Dr. Steven Y. Liang Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Nagi Gebraeel
School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Shreyes N. Melkote
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Jie Zhang
College of Mechanical Engineering
Donghua Unversity

Dr. Roger Jiao
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Wei Xu
Intelligent Manufacturing Unit
Shanghai Electric Academia Sinica

Date Approved: November 27, 2017

To my grandparents Omveer Singh, Onkari Devi, Rajesh Kumari, and Dr. Chokhey Singh.

ACKNOWLEDGEMENTS

It is impossible for me to thank all the people that have influenced, inspired, and motivated me throughout this PhD process but I will try my best to do so.

First and foremost, I would like to thank my advisor, Dr. Steven Y. Liang for providing me the opportunity to work with and learn from him and for his encouragement and guidance throughout my time as a Ph.D. student. I would also like to sincerely thank my committee members, Dr. Shreyes Melkote, Dr. Roger Jiao, Dr. Nagi Gebraeel, Dr. Wei Xu, and Dr. Jie Zhang for reviewing my dissertation and providing me with their valuable insights and comments. Special thanks go to Dr. Wei Xu, for all the valuable insight I have gained while working on various projects with him.

Next, I would like to thank my colleagues, Dr. Yamin Shao, Alexander Shih, Zhipeng Pan, Yanfei Liu, and Dr. Zishan Ding. I would like to especially thank Pan Zou for the support she has provided me and allowing me to discuss with her my many ideas (the good, the bad, and the extremely bad) that made this dissertation possible. I have learnt a lot from her from all the projects we have worked. Also, thanks for introducing me to coffee, without which, my last year of PhD would not have been possible.

I would like to thank my friends, Abhishek and Azad for all the life lessons I have learnt from them drinking bottomless coffee at Mel's Diner. I also want to thank Mitra for always reminding me how close I was to the finish line even when it felt like it was nowhere in sight. Special thanks to Anand, Prashant, Sumit, and Sudhir for all the support you guys have provided me. Luke Yates, I thank you for introducing me to some of the finer cuisines

in Atlanta and also for putting up with the smell of my awfully cooked food for the past four and a half years.

My family has been one of my biggest source of inspiration. You guys believed in my abilities more than I ever could and without your constant and unconditional love and support I would not have made it through this rigorous PhD process. I would like to thank my family members, both in India and US, including Cheenu, Mitali, Yakshi, Nishtha, Nikki, as well as all my aunts and uncles. Next up are my parents, Yogendra and Vineeta Rajora and my brother Vaibhav. To my parents, thank you for providing Vaibhav and I can have all the opportunities to succeed in this world. Vaibhav, I appreciate you being there for me whenever I needed.

Lastly, I want to thank Shanghai Electric Academia Sinica (SEAS) and Metals Industry Research and Development Center (MIRDC) for kindly providing me with the financial support as well as experimental data during my PhD.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

a_n	n^{th} independent variable
$b_{m_1}^{(1)}$	m_1 independent variable belonging to the 1st level
C_{a_1}	Constraint on the independent variable a_1
C_{b_1}	Constraint on the independent variable b_1
$x(a_i)$	Frequency of occurrence of variable a_i
l_{a_i}	Location of variable a_i in the chromosome
$l_{b_1^{(1)}}$	Location of the variable $b_1^{(1)}$
$G_{j_1}^{(0)}$	j_1 0th level gene
$w_{j_1,i}^{(0)}$	Weight value of the independent variable a_i for gene $G_{j_1}^{(0)}$
$w_{j_2,j_1}^{(1)}$	Weight value of the 1st level dependent variable $b_{j_1}^{(1)}$ for gene $G_{j_1}^{(1)}$
w_x	Crossover level weight-value
$w_x(r_o)$	Crossover weight value for the r_o^{th} level
x_p	Random integer between 1 and m_{r_o}
w_m	Mutation level weight-value
$w_m(r_o)$	Mutation weight value for r_o^{th} level
$CM^{(0)}$	Mutation candidates for the 0th level
CM	Mutation candidates for all levels

$CG^{(r_o)}_{b^{(r_o+1)}_{m(r_o+1)}}(c)$	A vector containing gene groups belonging to the r_o level that constraint the $b^{(r_o+1)}_{m_{(r_o+1)}}$ dependent variable and have the length c
P_{rm}	Mutation probability (between 0 and 1)
P	Random integer which is between 1 and $m_{(r_o+1)}$
Q	Random integer which is between 1 and the number of genes in c
R	Random integer which is between 2 and the number of genes in $CG^{(r_o)}_{b^{(r_o+1)}_{m(P)}}(Q)$
n	Number of genes in $CG^{(r_o)}_{b^{(r_o+1)}_{m(P)}}(Q)$ -R
$\eta\beta_{output}$	Number of biases of the output layer
TF_{output}	Transfer function of the output layer
α_i	Number of neurons in the ith hidden layer
$nw_{1,input}$	Number of weighted connections from the input layer to the 1st hidden layer
$w_{1,input}$	Weighted connections from the input layer to the 1st hidden layer
β_1	Bias values for the 1st hidden layer
PA_1	1st parent chosen for crossover
G_{PA_1}	Number of levels for parent PA_1
LG_{PA_1}	Length of genes for PA_1 on crossover level
RC	Random number between 1 and the smaller of LG_{PA_1} or LG_{PA_2}
Fr	Froude number
ks	Bed roughness height
h_1	Upstream hydraulic jump

N	Spindle Speed
V	Welding Speed
F_z	Plunge force
EFI	Emperical force index
UTS	Ultimate tensile strength
S	Solution set in which all unique optimal solutions are stored

SUMMARY

Hierarchically coupled constrained optimization problems (HCCOPs) are commonly encountered in the manufacturing industries, however, they haven't been categorized as such. Due to a lack of clear definition to identify these problems, numerous techniques have been developed for the optimization of HCCOPs but these techniques are not universally applicable to all HCCOPs and are unable to cope with large scale problems. Furthermore, current techniques used for the optimization of these HCOOPs only provide a single optimal solution upon execution. Though the single optimal solution may theoretically satisfy the objective function, it might not be applicable in real life scenario.

This research will first focus on establishing an abstract definition and identifying the common principles amongst different HCCOPs. Next, based on the established definition and common principles, a new optimization technique, based on evolutionary computation, will be developed. The proposed algorithm will be developed in a way such that only feasible solutions are generated during the iterations of the algorithm thereby reducing its computational complexity. To validate the proposed algorithm, it will be utilized to optimize HCCOPs commonly encountered in the manufacturing industry such as assembly job-shop scheduling problem (AJSSP) and the simultaneous optimization of Neural Network (NN) structure and weight values. The research will also focus on developing a technique for the multimodal optimization (MMO) of HCCOPs i.e. obtain multiple solutions with the same objective value. To validate the proposed MMO approach, it will first be utilized for the MMO of benchmark job-shop scheduling problem (JSSP) and permutation flow-shop scheduling problem (PFSSP) followed by the MMO of AJSSP.

This research would aid in the identification and MMO of HCCOP. The proposed algorithm could be easily applied to any HCCOP while requiring minimal changes.

CHAPTER 1. INTRODUCTION

1.1 Overview and Motivation

Constrained optimization problems (COPs), in the form of linear or nonlinear equalities or inequalities, are commonly encountered in the manufacturing industry and theoretical application, due to the limit and interaction between related impact factors and resources. The presence of non-linear constraints makes these real-world problems quite challenging to solve, due to their multimodal and complex search space and disjoint feasible regions which render the search for an optimal solution difficult and inefficient. As such, researchers have proposed many different constraint handling techniques for the optimization problems.

The different categories of constraint handling methods have shown promising results in solving some of the COPs, but there is no specific research of methodology being done on solving one special COP called HCCOP. HCCOP are a special type of COP which consists of both independent and dependent variables and the constraints on the i^{th} level dependent variables (where $i = 1,2,3\ldots,n$) are a function of the (i-1) level dependent variables and the constraints on 1^{st} level dependent variables are a function of the independent variables. Though HCCOPs are commonly encountered, both in manufacturing industries as well as theoretical applications, there is no previous work done in identifying HCCOPs as such. Furthermore, the algorithms available for solving COPs require immense modifications in order to apply them to solve HCCOPs. Also, each type of HCCOP require a unique modification. Therefore, there is still an opportunity to identify

the common principles amongst different HCCOPs and develop and algorithm that is capable of solving the different HCCOP without requiring major modifications.

Lastly, current techniques used to optimize HCCOPs such as AJSSP only provide a single optimal solution upon execution. Though this single optimal solution may satisfy the original objective function, it may not be applicable in real-life scenarios. Therefore, there is a need to develop a MMO algorithm in order to obtain multiple optimal solutions for the HCCOPs.

1.2 Research Goals and Objectives

Despite the extensive research in COPs, there is still an opportunity for classifying different optimization problems encountered in the manufacturing industry as HCCOPs developing a versatile algorithm capable of solving these problems. Furthermore, it is of great importance, in both theory as well as practical application, to develop an algorithm for the MMO of the scheduling problem. Therefore, the objective of the current research will be as follows:

1. Develop definitions and identify the common principles to classify COPs as HCCOPs.
2. Develop a versatile algorithm to optimize HCCOPs.
3. Utilize the developed definition and common principles to identify HCCOPs in manufacturing industry as well as theoretical applications.
4. Utilize the developed algorithm to optimize the classified HCCOPs.
5. Develop a MMO algorithm to obtain multiple global optima for HCCOPs.
6. Validate the proposed MMO by its application to JSSP, PFSSP, and AJSSP.

1.3 Research Approach

To develop an algorithm for optimizing HCCOPs, the common principles amongst different HCCOP must first be identified. A flowchart of the methodology is shown in Figure 1.

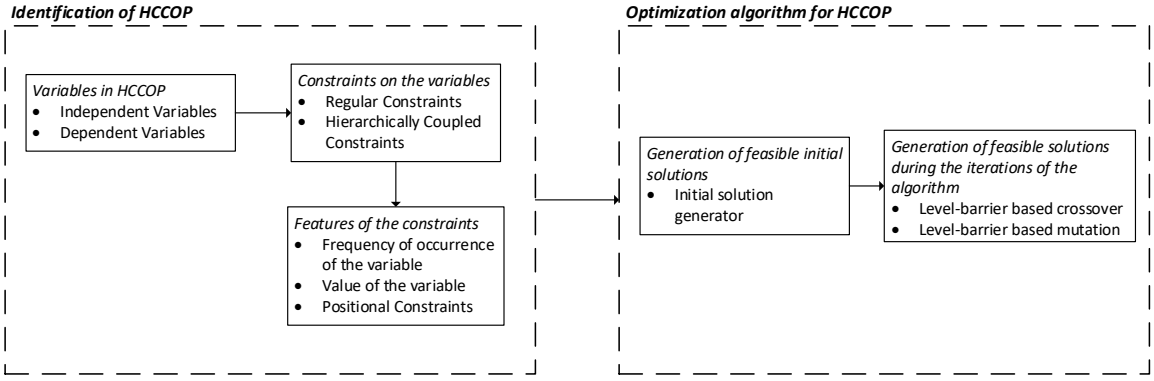

Figure 1. Research approach for optimization of HCCOP

As shown in the flowchart, the different types of variables that make up HCCOPs are first identified. Next, the constraints on these variables are identified along with their hierarchically coupled nature. Lastly, the features of these constraints are established and an optimization problem is formed. To solve the optimization problem, an algorithm based on evolutionary computation is proposed. To reduce computation complexity of the proposed algorithm, an initial solution generator, a level-barrier based crossover operator, and a level-barrier based mutation operator are developed. The new operators are developed in a way such that only feasible solutions are generated during its iterations. The new algorithm is also developed in a manner such that very minimal changes are required when it is used to optimize different HCCOP.

To validate the developed definitions and common principles, they are first utilized to classify the optimization of assembly job-shop scheduling problem (AJSSP) and the simultaneous optimization of Neural Network (NN) structure and weights as HCCOPs i.e. the variables along with the constraints and the features of these constraints are identified. Next, the proposed algorithm is used to optimize different case studies of the two problems. First, the proposed initial solution generator is used to create feasible initial solutions for the two problems. Next, it is demonstrated how the proposed level-barrier based crossover and mutation operators can be used to create feasible solutions during the rest of the iterations of the algorithm. The results obtained by using the proposed algorithm are also compared with the results published in other literature.

In the second part of the research, a MMO technique for HCCOP is developed. To enable MMO, the algorithm developed for optimization of HCCOPs is paired with *k-means clustering* algorithm [1]. In the proposed MMO algorithm, the solutions of every generation are clustered together, based on a feature matrix. New solutions are generated by utilizing the operators of the proposed algorithm within each cluster. By doing so, it can be ensured that the algorithm will converge to solutions having the same objective value but different features. As the research available in the optimization of AJSSP is not as extensive as in the optimization of job-shop scheduling problem (JSSP) and flow-shop scheduling problems (FSSP), no benchmark AJSSP are available. The lack of benchmark AJSSP means that the global optima of the AJSSP used in this research are unknown. Since the quality of solutions obtained during MMO is extremely important and multiple global optima are desired during the MMO of different problems, the proposed MMO algorithm

is first used for the MMO of benchmark JSSP and permutation PFSSP, as their global optima is known, followed by the multimodal optimization of AJSSP.

1.4 Overview of Thesis

Figure 2 presents the technical roadmap of this dissertation, including motivation & significance, problem formulation, technical approach, methodology & solution, and validation & application.

CHAPTER 1 discusses motivation and significance of this research topic along with a holistic view of research goals. CHAPTER 2 provides background information and a review of the present and past literature in the area of COPs and MMO.

In CHAPTER 3, an abstract definition and the common principles used to identify some COPS as HCCOP are presented in. Next, an algorithm, based on evolutionary computation, to optimize the HCCOPs is also developed.

In Chapters 4 and 5, the definitions and common principles are used to classify the optimization of AJSSP and simultaneous optimization of NN structure and weights values of HCCOPs. Next, the proposed algorithm is utilized to optimize these problems and the obtained results are compared with those published in literature.

In CHAPTER 6, the MMO technique is first developed. The performance of the proposed MMO is first validated by its application for the MMO of JSSP and PFSSP followed by its application for AJSSP.

CHAPTER 7 concludes the dissertation with discussions of research limitation and future works.

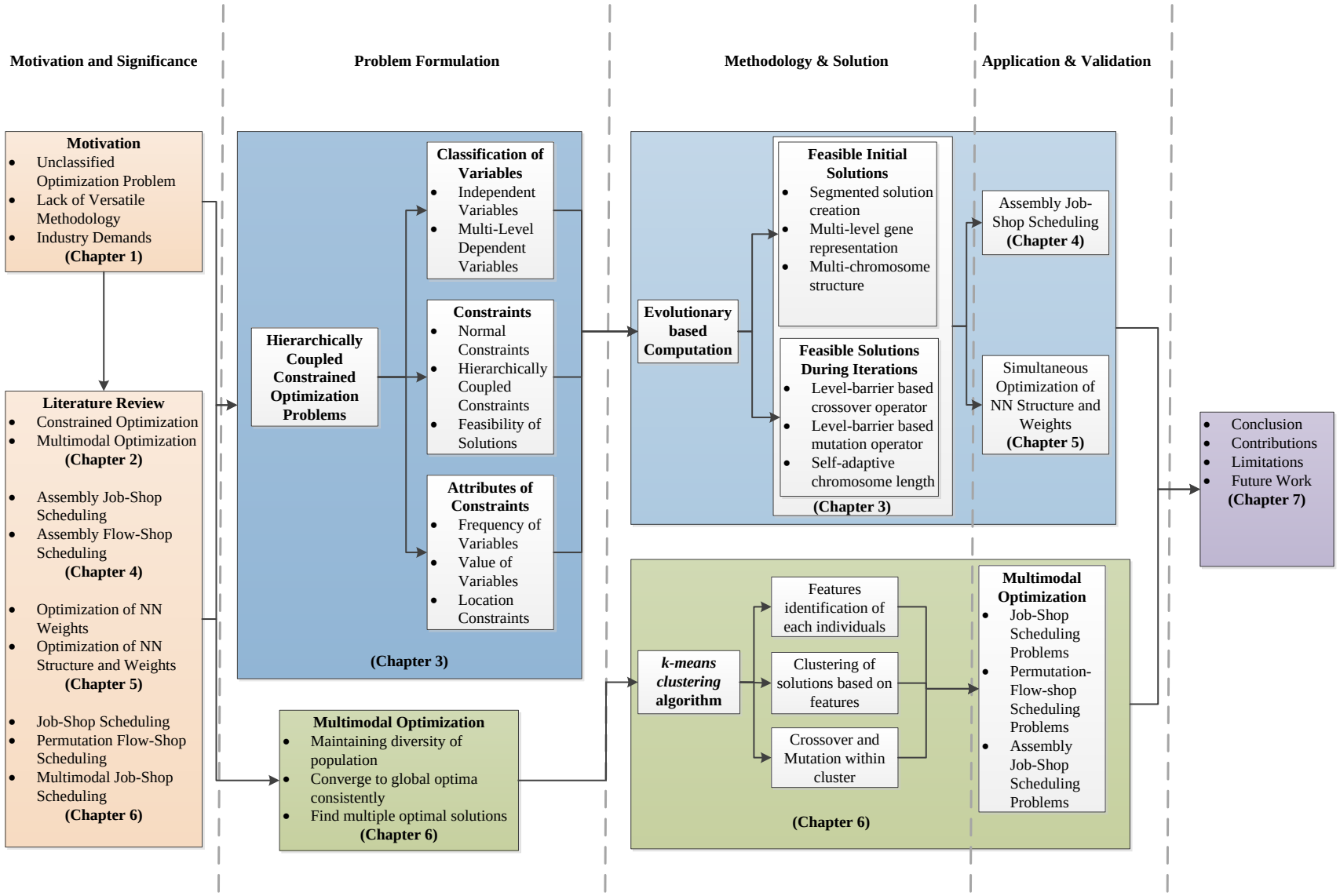

Figure 2. Organization of the dissertation

CHAPTER 2. LITERATURE REVIEW

The literature review is categorized into two parts: 1) research on COPs and 2) research on the different MMO techniques. The literature review on COPs discusses the categories of COPs and the algorithms proposed to deal with the different types of COPs. The literature review on MMO includes discussion about the concept behind MMO and the different techniques utilized for MMO. After the review, a summary of potential avenues for the current research is presented. Further literature review about the different algorithms used to optimize AJSSP, simultaneously optimize NN weight and structure, and for the MMO of JSSP is provided in Sections 4.1, 5.1, and 6.2

2.1 Literature Review on Constrained Optimization Problems

As mentioned earlier, COPs are omnipresent both in real-world problems and theoretical applications. Researchers have proposed various methods to solve these COPs. As discussed by Coello [2], constraint handling methods can be classified into five different categories: (1) Penalty functions, (2) Special representations and operators, (3) Repair algorithms, (4) Separation of objectives and constraints, and (5) Hybrid methods.

Penalty functions [3,4,5] are commonly used to handle COPs by transforming them into unconstrained problems by adding or subtracting a certain value to or from the objective function based on the amount of constraint violation present in the solution. Over the years, different types of penalty functions have been proposed such as the death penalty [6], static penalty [7], dynamic penalty [8] etc. Implementation of the penalty functions is

straightforward but they include drawbacks such as the sensitivity of the solutions to the penalty parameters and the lack of a guideline to determine the value of the user defined penalty parameters.

Researchers have developed special representation schemes for problems where generic representation scheme might not be appropriate. Due to the change of representation, special operators are also designed. Koziel and Michalewicz [9] and Monson et al. [10] used decoders, a special operator that gives instructions on how to build a feasible solution, to transform constrained optimization problems into unconstrained ones via homomorphous mappings. Genetic Algorithm for Numerical Optimization for Constrained Problems (GENOCOP), proposed by Michalewicz and Janikow [11], reduces the search space through the elimination of equalities constraints with an equal number of problem variables. Though a competitive constraint-handling technique, its drawbacks include high computational cost (over 1.4 million fitness function evaluations required) and their inability to cope with non-linear constraints, or disjoint feasible regions.

As the name suggests, the purpose of the repair algorithms is to 'repair' the infeasible solutions either for evaluation or to replace the original individual in the population as shown in the works of Chootinan *et al.* [12] and Pal *et al.* [13]. GENOCOP III, an extension of the original GENOCOP, proposed by Michalewicz and Nazhiyath [14], is also based on the same principle. Alducin *et al.* [15] used a repair method for DE to solve dynamic constrained problems. Fan *et al.* [16] proposed a repair operator, inspired by opposition-based learning, that employed reverse correction strategy to fix solutions that violated the box-constraints. The main drawback of the repair algorithm includes high computational

cost, the dependence on the availability of feasible solutions as reference to repair infeasible ones, and the need for modification for different applications.

In separation of constraints and objectives, the constraints and objectives are handled separately unlike penalty functions, where the value of the objective function and constraint are assigned to a single fitness value. Methods such as coevolution [17], use of multi-objective optimization [2], and behavioral memory [18] are some examples of techniques that employ separation of constraints and objectives. Deb [19] separated the objective and constraints by proposing three feasibility rules that are very popular and effective for handling constrained optimization. These feasibility-based rules have been used extensively with other hybrid evolutionary algorithms to solve linear and non-linear COPs problems as shown by Ma and Simon [20], Chen *et al.* [21], Zhou *et al.* [22], and Mohamed and Sabry [23]. Since the feasibility rule emphasizes feasible solutions to infeasible ones, their usage can lead to premature convergence or the need for high number of user inputs.

Literature review shows that there is a lack of categorization of COPs with HCCOPs as such. Furthermore, the algorithms available for the optimization of COPs currently used to solve HCCOPs are not universal and require immense modifications to solve the HCCOPs. Therefore, there is room to develop a methodology to identify and optimize the HCCOPs.

2.2 Literature review on Multimodal Optimization

Generally speaking, there are three different types of optimization problems [24]. The first type is referred to as classical optimization problem and the objective is to solve a single objective function which has a single global optimum. The second type of

optimization problems are called multi-objective optimization problems. As the name suggests, these problems have multiple objectives and the second type of optimization problem, rather than having a single objective function, the problem has multiple objectives functions and the solution to these problems is called Pareto-set. The third type of optimization problems, called MMO problems, again have a single objective function, but instead of having a single global optimum, the function has several global and local optima. The optimization objective for the third type of optimization problems is to find as many of these global and local optima as possible. Figure 3 shows an example of a MMO function with 5 global optima in (a) and a single global optimum and 4 local optima in (b).

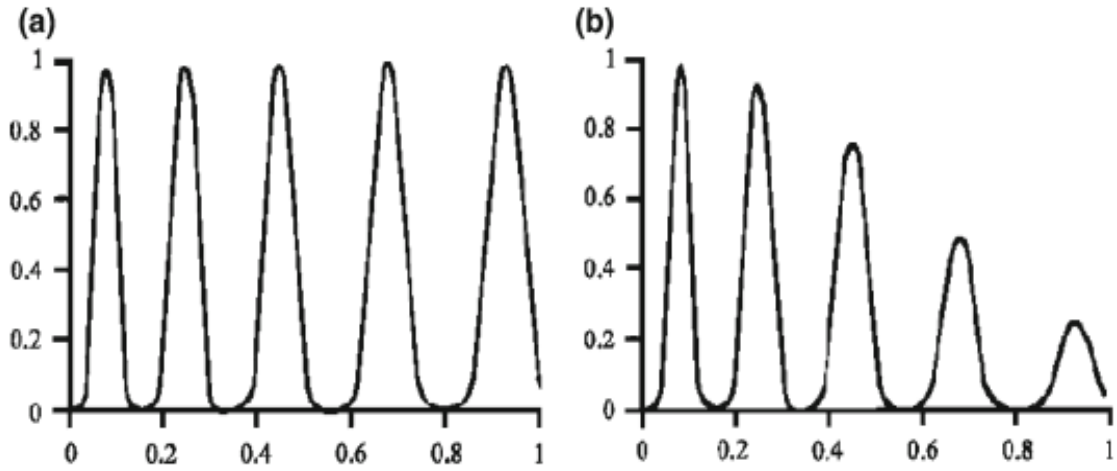

Figure 3. Example of MMO functions.

Unlike classical optimization problems, the goal of MMO problems is to obtain multiple optimal solutions in a single execution of the algorithm by using niching methods. Distance between individuals is used to separate the individuals into different niches. Fitness sharing [25] and crowding [26] method are examples of classical niching techniques used for MMO. In fitness sharing, the diversity of the population is maintained

by degrading an individual's fitness based on the presence of other similar neighboring individuals. The degradation of the fitness takes place during the selection process and the degradation discourages similar individuals to occupy the same niche [27]. In crowding method, designed to ensure population diversity and prevent premature, a single individual is compared to a small random sample taken from the current population, and the most identical individual in the chosen sample is replaced. These niching techniques have been used as a foundation to create other niching techniques such as RTS [28], clearing [29], multinational GA [30], and speciation [31].

More recently, other meta-heuristics have been modified to induce niching behavior with Particle Swarm Optimization (PSO) and differential evolution (DE) being the most commonly used algorithms. In PSO, *memory* is to induce niching behavior. This is accomplished by splitting the swarm into two categories: 1. explorer-swarm which consists of current particles, and (2) memory-sawm which consist of best-known positions of the individuals of the swarm. Furthermore, classical niching techniques have also been used in conjunction with PSO for MMO [32,33,34,35].

In DE [36], scaled differences between randomly sampled pairs of individuals is utilized to determine how individual's vectors are modified to produce offspring. To incorporate niching into DE, probabilistic parent selection scheme based on fitness and proximity information has been proposed [37]. Parent-centric mutation strategies combined with crowding [38] and speciation [39] are also DE niching variants that have shown promising results in MMO. Clustering techniques have also been utilized for niching for MMO. *K-means clustering,* dynamic niche sharing [40], dynamic niche

clustering [41], and species conserving GA [42] are some algorithms belonging to this category.

As it can be seen, there is a plethora of work available on MMO, however, these techniques have only been tested on benchmark mathematical problems. Furthermore, the aim of these techniques is to find both global and local optima which increases their computational complexity. Also, it is possible for local optima to have much worse objective value than global optima which is undesirable in scheduling problems. Compared to the MMO of benchmark problems very little [24,93] to no work is available in the MMO of scheduling problems and HCCOPs.

2.3 Summary

Based on literature reviews relating to COPs,

In the area of MMO, niching techniques, clustering techniques, meta-heuristic algorithms with niching techniques have been developed. However, they have only been implemented on benchmark mathematical problems and are utilized for finding both global and local optima.

To address these issues, this dissertation consists of the following tasks:

1. Identification of common principles amongst different HCCOPs.
2. Development of an algorithm to optimize the HCCOPs.
3. Validation of the proposed algorithm by its application to different HCCOPs.
4. Development and validation of an algorithm for MMO of scheduling problems and HCCOPs.

CHAPTER 3. IDENTIFICATION AND MATHEMATICAL MODELING OF HIERARCHICALLY COUPLED CONSTRAINT OPTIMIZATION PROBLEMS

In this chapter, the characteristics of HCCOP, specifically, its variables, the different categories of constraints on these variables, and the features of these constraints are first introduced. Next, to solve the HCCOP, a new algorithm is developed, based on evolutionary computation. An adaptive initial solution generator is proposed which capable of coping with the different constraints of HCCOP and create feasible initial solutions. Next, to create feasible solutions for every iteration of algorithm, level-barrier based crossover and mutation operators are proposed.

3.1 Description of Hierarchically Coupled Constraint Optimization Problems

The adjustable parameters in the HCCO can be divided into two categories, independent variables and dependent variables, as follows:

Independent variables: $a_1, a_2, \ldots, a_n$

The dependent variables can further be split into different levels of dependent variables as follows:

The 1st level dependent variables: $b_1^{(1)}, b_2^{(1)}, ..., b_{m_1}^{(1)}$

The 2nd level dependent variables: $b_1^{(2)}, b_2^{(2)}, ..., b_{m_2}^{(2)}$

The k^{th} level dependent variables: $b_1^{(k)}, b_2^{(k)}, ..., b_{m_k}^{(k)}$

As the name suggests, HCCOPs have hierarchically coupled constraints that differentiate them from regular COPs. Constraints in the HCCO can be split into two categories; 1. basic constraints and 2. Hierarchically coupled constraints (HCCs). Basic constraints are the constraints on the independent variables. These constraints are not functions of other independent or dependent variables. These constraints can be written as:

C_{a_i} ; where the independent variable $\boldsymbol{a_1}$ is subjected to C_{a_i} and i = 1, 2, 3,…, n

HCCs are the constraints that exist on the dependent variables. These constraints are a function of independent and lower level dependent variables. The constraints on the 1st level dependent variables are a function of the independent variables and can be written as:

$C_{b_{j_1}^{(1)}}$; where $j_1 = 1, 2, 3..., m_1$. $C_{b_{j_1}^{(1)}}$ is the constraint on the dependent variable $b_{j_1}^{(1)}$ and is a function of $a_1, a_2, ..., a_n$.

The constraints on the 2nd level dependent variables are a function of the 1st level dependent variables and can be written as:

$C_{b_{j_2}^{(2)}}$; where $j_2 = 1, 2, 3, ..., m_2$. $C_{b_{j_2}^{(2)}}$ is the constraint on the dependent variable $b_{j_2}^{(2)}$ and is a function of $b_1^{(1)}, b_2^{(1)}, \ldots, b_{m_1}^{(1)}$.

Similarly, the constraints on the k^{th} level dependent variables are a function of the (k-1) level dependent variables and can be written as:

$C_{b_{j_k}^{(k)}}$; $j_k = 1, 2, 3, ..., m_k$. $C_{b_{j_k}^{(k)}}$ is the constraint on the dependent variable $b_{j_k}^{(k)}$ and is a function of $b_1^{(k-1)}, b_2^{(k-1)}, \ldots, b_{j_{(k-1)}}^{(k-1)}$.

The objective then becomes:

$$\min f(a_1, a_2, \ldots, a_n, b_1^{(1)}, b_2^{(1)}, \ldots, b_{m_1}^{(1)}, b_1^{(2)}, b_2^{(2)}, \ldots, b_{m_2}^{(2)}, \cdots b_1^{(k)}, b_2^{(k)}, \ldots, b_{m_k}^{(k)}) \quad (1)$$

$$\text{subject to } C_{a_i} \text{ for } i = 1, 2, 3..., n \quad (2)$$

$$C_{b_{j_1}^{(1)}} \text{ for } j_1 = 1, 2, 3..., m_1$$

$$\vdots$$

$$C_{b_{j_k}^{(k)}} \text{ for } j_k = 1, 2, 3..., m_k$$

The constraints on the independent and dependent variables mentioned above can have the following features:

(1) The frequency of occurrence of the variable

Let the frequency of occurrence of the variable a_i be $x(a_i)$. Then,

$$x(a_i) = t_0,\ t_0 \in N^* \text{ OR } t_1 \le x(a_i) \le t_2,\ t_1 \in N^*,\ t_2 \in N^*,\ x(a_i) \in N^* \quad (3)$$

(2) The value of the variable

$$v_1 \le a_i \le v_2, v_1, v_2 \in R \tag{4}$$

(3) Position constraints i.e. if the final solution has sequence constraints then, constraints may be present on the location of the variables. For example, if the location of $b_1^{(1)}$ is constrained by $a_1, a_2, \ldots, a_n$, then this constraint can be represented as:

$$\min\{l_{b_1^{(1)}}\} > \max\{l_{a_i} \mid i = 1, \ \cdots, \ n\} + 1 \tag{5}$$

Where $l_{b_j^{(1)}}$ represents the location of variable $b_1^{(1)}$ in the solution while represents the location of the independent variables.

3.2 Modeling of the solution

To accommodate for a variety of HCCOPs, solutions in the proposed algorithm will be represented using a multi-chromosome structure (if applicable) i.e. each aspect of the solution will be located in a different chromosome. From henceforth in the paper, each solution will be referred to as an individual (or chromosome) and each individual will be made up of multiple sub-chromosomes. Each sub-chromosome can be represented using different methods such as permutation or binary encoding and the information can be encoded by changing the following: (1) The location of the variables, (2) The frequency of the variables, and (3) The value of the variables. As mentioned earlier, to cope with the HCC's and only search the feasible solution space, a new initial solution generator, level

barrier based crossover, and level barrier based mutation operators are proposed in this paper. Figure 4 shows a brief flow chart of the proposed algorithm to solve HCCOPs.

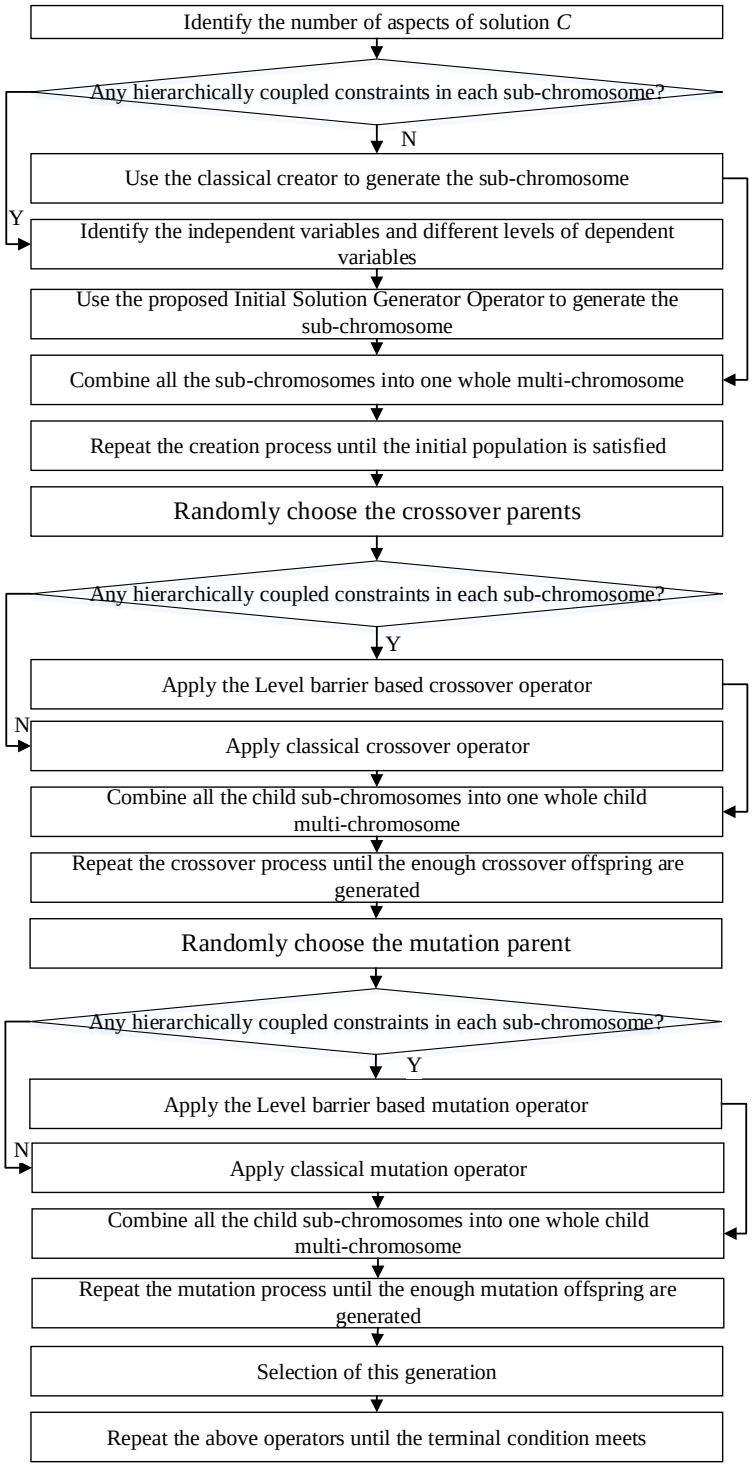

Figure 4. A brief flowchart of the proposed algorithm

The special operators mentioned in Figure 4 that enable the proposed algorithm to function on the HCCOPs are described in further details in the following sections.

3.2.1 Initial Solution Generator Operator

To create an initial population, all the sub-chromosomes of each chromosome must first be created. The generation of a solution that satisfies all HCCs is accomplished in multiple steps. First, a partial solution is created that satisfies the lowest level of the HCCs. Next, multiple partial solutions are appended together to satisfy the next level of HCCs. These steps are repeated until a complete solution is created that satisfies all the HCCs. Unlike the proposed initial solution generator operator, the creation operator used in classical GA generates a complete solution in a single step. This indicates that a solution created using the classical creation operator might not satisfy all the HCCs and therefore, might be infeasible. Since the proposed initial solution generator only creates feasible solutions, it significantly reduces the computation time required to find an optimal solution as the algorithm must only search the feasible solution space for an optimal solution. The overall flow chart of the initial solution generator is shown in Figure 5.

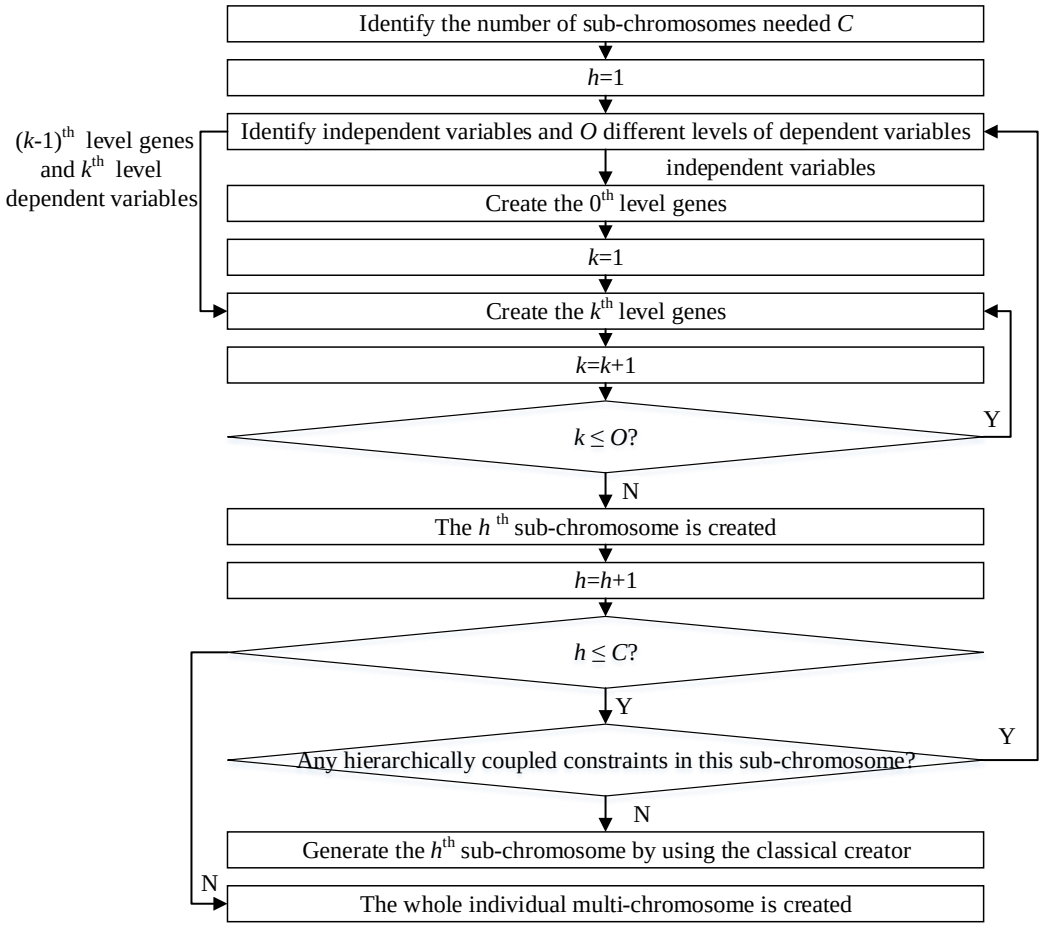

Figure 5. Flowchart of the initial solution generator operator

The detailed steps of the initial solution generator are outlined below:

Step 1.a: Create the 0^{th} level genes

For every 1^{st} level dependent variable $b_{j_1}^{(1)}$, create a 0^{th} level gene $G_{j_1}^{(0)}$ which comprises of the independent variables that constrain the 1^{st} level dependent variable $b_{j_1}^{(1)}$, shown in Figure 6.

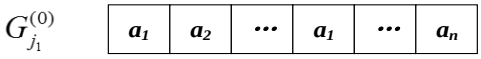

Figure 6. The 0th level gene

The length of this gene is given by:

$$length\ (G_{j_1}^{(0)}) = \sum_{i=1}^{n} w_{j_1,i}^{(0)} \cdot x(a_i) \tag{6}$$

Where:

$$w_{j_1,i}^{(0)} = \begin{cases} 1 \text{ If } a_i \text{ constraints the } 1^{st} \text{ level dependent variable } b_{j_1}^{(1)} \\ 0 \text{ Otherwise} \end{cases} \tag{7}$$

$$w^{(0)} = \begin{bmatrix} w_{1,1}^{(0)} & \cdots & w_{1,i}^{(0)} & \cdots & w_{1,n}^{(0)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j_1,1}^{(0)} & \cdots & w_{j_1,i}^{(0)} & \cdots & w_{j_1,n}^{(0)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m_1,1}^{(0)} & \cdots & w_{m_1,i}^{(0)} & \cdots & w_{m_1,n}^{(0)} \end{bmatrix} \tag{8}$$

Step 1.b: For every 1st level dependent variable $b_{j_1}^{(1)}$, create the 1st level gene $G_{j_1}^{(1)}$,shown in Figure 7, by placing the 1st level dependent variable after the 0th level genes.

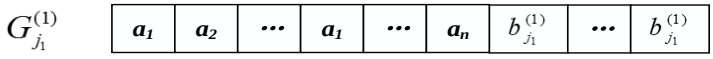

Figure 7. The 1st level gene

The length of the 1st level gene is given by:

$$length\ (G_{j_1}^{(1)}) = length\ (G_{j_1}^{(0)}) + x\ (b_{j_1}^{(1)}) \tag{9}$$

Step 2: For every 2nd level dependent variable $b_{j_2}^{(2)}$, create the 2nd level gene $G_{j_2}^{(2)}$, shown in Fig. 5, by combining all the 1st level genes $G_{j_1}^{(1)}$ that constraint the 2nd level dependent variable $b_{j_2}^{(2)}$ and placing the 2nd level dependent variable $b_{j_2}^{(2)}$ at the end.

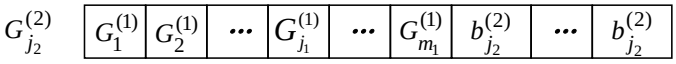

Figure 8. The 2nd level gene

The length of the 2nd level gene is given by:

$$length(G_{j_2}^{(2)}) = \sum_{j_1=1}^{m_1} w_{j_2,j_1}^{(1)} \cdot length(G_{j_1}^{(1)}) + x(b_{j_2}^{(2)}) \tag{10}$$

$$w_{j_2,j_1}^{(1)} = \begin{cases} 1 \text{ If } G_{j_1}^{(1)} \text{ constraints the } 2^{nd} \text{ level gene } b_{j_2}^{(2)} \\ 0 \text{ Otherwise} \end{cases} \tag{11}$$

$$w^{(k-1)} = \begin{bmatrix} w_{1,1}^{(k-1)} & \cdots & w_{1,j_{(k-1)}}^{(k-1)} & \cdots & w_{1,m_{(k-1)}}^{(k-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j_k,1}^{(k-1)} & \cdots & w_{j_k,j_{(k-1)}}^{(k-1)} & \cdots & w_{j_k,m_{(k-1)}}^{(k-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m_k,1}^{(k-1)} & \cdots & w_{m_k,j_{(k-1)}}^{(k-1)} & \cdots & w_{m_k,m_{(k-1)}}^{(k-1)} \end{bmatrix} \tag{12}$$

Step 2 can be repeated until the k^{th} level gene, shown in Fig. 6, is created. To create the k^{th} level gene, the (k-1) level genes are combined and the k^{th} level dependent variable is placed at the end.

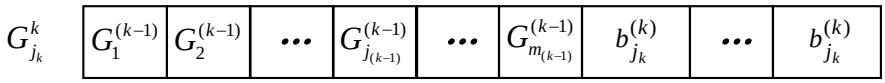

Figure 9. The k^{th} level gene

The length of the k^{th} level gene is given by:

$$length(G_{j_k}^{(k)}) = \sum_{j_{(k-1)}=1}^{m_{(k-1)}} w_{j_k, j_{(k-1)}}^{(k-1)} \cdot length(G_{j_{(k-1)}}^{(k-1)}) + x(b_{j_k}^{(k)}) \tag{13}$$

$$w_{j_k, j_{(k-1)}}^{(k-1)} = \begin{cases} 1 \text{ If } G_{j_{(k-1)}}^{(k-1)} \text{ is a constraining } (k-1)^{th} \text{ level gene of } b_{j_k}^{(k)} \\ 0 \text{ Otherwise} \end{cases} \tag{14}$$

$$w^{(k-1)} = \begin{bmatrix} w_{1,1}^{(k-1)} & \cdots & w_{1,j_{(k-1)}}^{(k-1)} & \cdots & w_{1,m_{(k-1)}}^{(k-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j_k,1}^{(k-1)} & \cdots & w_{j_k,j_{(k-1)}}^{(k-1)} & \cdots & w_{j_k,m_{(k-1)}}^{(k-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m_k,1}^{(k-1)} & \cdots & w_{m_k,j_{(k-1)}}^{(k-1)} & \cdots & w_{m_k,m_{(k-1)}}^{(k-1)} \end{bmatrix} \tag{15}$$

As it can be noticed, when $k = 2$, Equation (13) is exactly the same with Equation (10). When utilizing this method for non-constrained problems, only thie 0^{th} level gene will be used.

3.2.2 Level-barrier based crossover operator

The purpose of the level barrier based crossover operator is to generate feasible solutions for the next generation while maintaining the diversity of the population. Unlike the classical crossover operator, where the genes are switched based single-point, two points, or uniformly between the parents, crossover in the level barrier based crossover operator is performed on each level. First, a gene belonging to the kth level is selected from

the two parents. Next, a (*k*-1) level gene that constraints the kth level gene is selected and its position in the two parents is switched. By switching the entire position of the (*k*-1) level gene, the HCCs of k^{th} as well as the (*k*+1) are still satisfied. This would be difficult to achieve with the classical crossover operator as genes are selected randomly regardless of the level they belong to. The steps of the level barrier based crossover are shown below.

First, for each level in the individual, define a Crossover Level Weight value $w_x = [w_x(2), w_x(3), \cdots, w_x(r_o), \cdots, w_x(k-1)]$, where

$$w_x(r_o) = \begin{cases} 1 \text{ If crossover operator will be utilized in the } r_o{}^{th} \text{ level} \\ 0 \text{ Otherwise} \end{cases} \tag{16}$$

Next, a pair of individuals from the current generation are selected (based on the roulette selection) for crossover and the following steps are taken.

FOR all positive integer number $r_o \le k$

IF $w_x(r_o) = 1$ THEN

x_p ← A random integer number between 1 and m_{r_o}

$[G^{(r_o-1)}_{j_{(r_o-1)}}]$ ← Randomly selected gene which constraints the $b^{(r_o)}_{x_p}$ variable.

Switch the positions of the $[G^{(r_o-1)}_{j_{(r_o-1)}}]$ gene in the $r_o{}^{th}$ level of each parent

ELSE

Keep the parents' original genes as the genes of the $r_o{}^{th}$ level in the offspring

ENDIF

ENDFOR

The change of corresponding sub-chromosome before and after the procedure is shown in Figure 10.

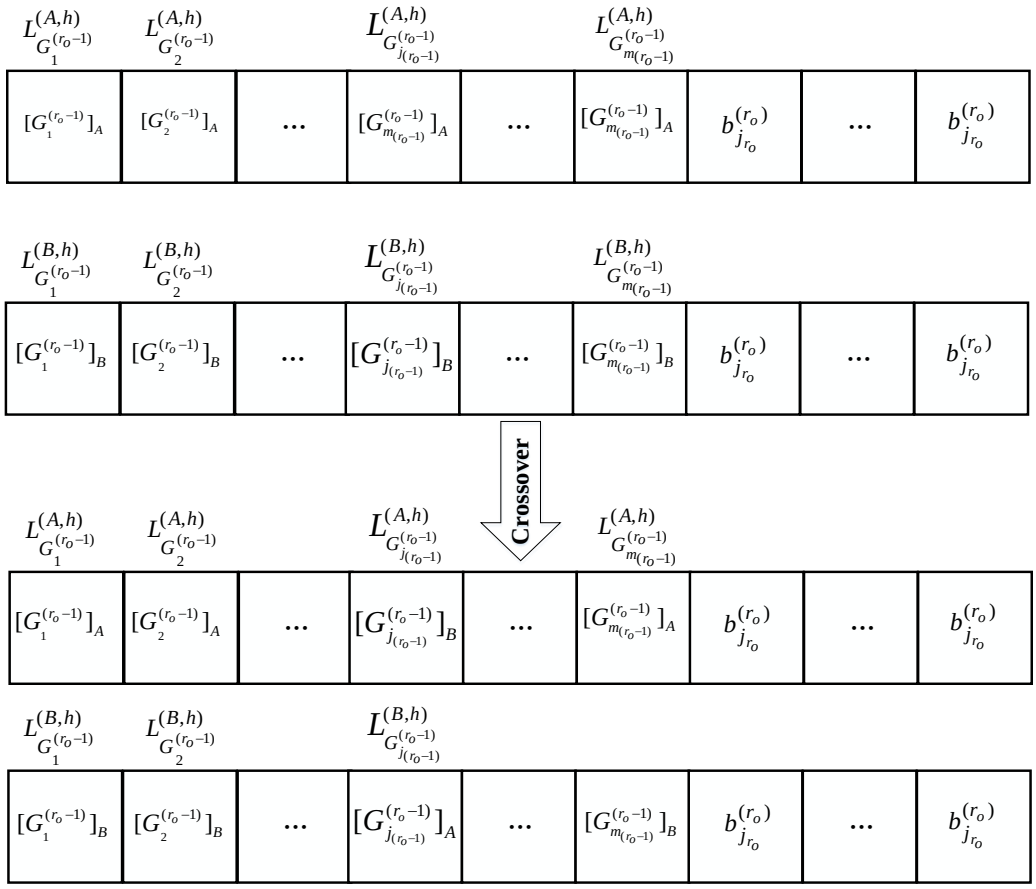

Figure 10. Illustration of crossover on (r_o-1) level gene of the h^{th} sub-chromosome constraining the variable $b_{j_o}^{(r_o)}$

In Figure 10, Individuals A and B represent solutions (current generation) of the optimization problem, Individuals A' and B' represent solutions (next generation) of the optimization problem, $[G_1^{(r_o-1)}]_A$ represents the 1st gene of the (r_o-1) level of individual *A*, and $L_{G_1^{(r_o-1)}}^{(A,h)}$ represents the location of $G_1^{(r_o-1)}$ in the h^{th} sub-chromosome of Individual A.

3.2.3 Level-barrier based mutation operator

The purpose of the level barrier based mutation operator is to increase the diversity of population by making small random changes to the parents. Unlike classical mutation operator, where the value of a randomly chosen gene or the location of two or more randomly chosen genes is altered, in the level barrier based mutation operator, the positions of multiple (*k*-1) level genes that have the same length and constrain the same k^{th} dependent variable is switched. Since multiple (*k*-1) level genes that constraint the same k^{th} level gene are repositioned, the HCCs of the k^{th} as well as the (*k*+1) are still satisfied. This task would be difficult to accomplish with classical mutation operator as the value or position of randomly genes are altered, regardless of the levels.

First, define a mutation rate, P_m and for each level in the individuals, define a mutation level weight-value, w_m where $w_m = [w_m(1), w_m(2), \cdots, w_m(r_o), \cdots, w_m(k-1)]$ and $w_m(r_o)$ is given be:

$$w_m(r_o) = \begin{cases} 1 \text{ If mutation operator will be utilized in the } r_o{}^{th} \text{ level} \\ 0 \text{ Otherwise} \end{cases} \tag{17}$$

To pick the genes whose positions will be switched, mutation candidates from each level must be picked. The mutation candidates of each level stored in the matrix CM , where $CM = \{CM^{(0)}, CM^{(1)}, \cdots, CM^{(r_o)}, \cdots, CM^{(k-1)}\}$ and

$$CM^{(r_o)} = \begin{bmatrix} CG^{(r_o)}_{b_1^{(r_o+1)}}(1) & CG^{(r_o)}_{b_1^{(r_o+1)}}(2) & \cdots & CG^{(r_o)}_{b_1^{(r_o+1)}}(c) \\ CG^{(r_o)}_{b_2^{(r_o+1)}}(1) & CG^{(r_o)}_{b_2^{(r_o+1)}}(2) & & CG^{(r_o)}_{b_2^{(r_o+1)}}(c) \\ & \ddots & & \\ CG^{(r_o)}_{b_{m_{(r_o+1)}}^{(r_o+1)}}(1) & CG^{(r_o)}_{b_{m_{(r_o+1)}}^{(r_o+1)}}(2) & & CG^{(r_o)}_{b_{m_{(r_o+1)}}^{(r_o+1)}}(c) \end{bmatrix} \quad (18)$$

$CG^{(r_o)}_{b_{m_{(r_o+1)}}^{(r_o+1)}}(c)$: A vector containing gene groups belonging to the r_o level that constraint the $b_{m_{(r_o+1)}}^{(r_o+1)}$ dependent variable and have the length c.

1, 2, …, c: Length of genes.

This method ensures that multiple genes from each level have the probability of being mutated which would lead to an increase in the diversity of the population.

Once these variables have been defined, the following steps can be taken for the level barrier based mutation:

$P_{rm} \leftarrow$ A random number which is between 0 and 1.

IF $P_{rm} \le P_m$ THEN

FOR all positive integer number $r_o \le k-1$ DO

IF $w_m(r_o) = 1$ THEN

$P \leftarrow$ A random integer which is between 1 and $m_{(r_o+1)}$.

$Q \leftarrow$ A random integer which is between 1 and the number of genes in *c*.

$R \leftarrow$ A random integer which is between 2 and the number of genes in $CG^{(r_o)}_{b_{m_{(P)}}^{(r_o+1)}}(Q)$

$n \leftarrow$ the number of genes in $CG^{(r_o)}_{b^{(r_o+1)}_{m(P)}}(Q)$ -R

FOR all positive integer number $i \leq (n+1)$ DO

Let the i^{th} gene of the gene group $CG^{(r_o)}_{b^{(r_o+1)}_{m(P)}}(Q)$ takes the original position of $(R+i-1)^{th}$ gene.

ENDFOR

FOR all positive integer number $n+2 \leq j \leq R+n$ DO

Let the j^{th} gene of the gene group $CG^{(r_o)}_{b^{(r_o+1)}_{m(P)}}(Q)$ takes the original position of $(j-n-1)^{th}$ gene.

ENDFOR

ELSE

keep the parent's gene as the r_o^{th} level offspring

ENDIF

ENDFOR

ENDIF

The change of corresponding sub-chromosome before and after the procedure is shown in Figure 11.

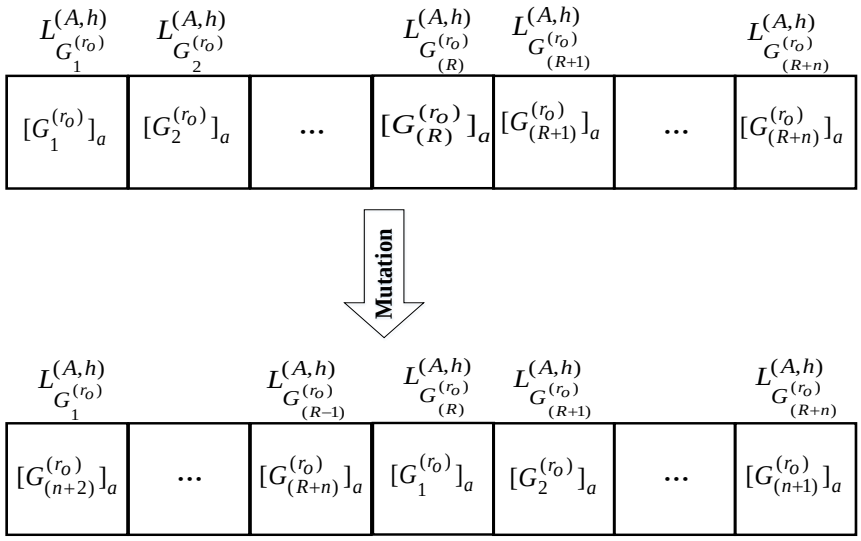

Figure 11. The illustration of mutation on r^{th}_o level candidate of the h^{th} sub-chromosome

3.3 Summary

This chapter presented an abstract definition and common principles used to identify HCCOPs as well as an algorithm proposed for optimizing these problems. Section 3.1, introduces the dependent and independent variables encountered in HCCOPs, the two categories of constraints on these variables, and the features of these constraints.

Section 3.2 presents an algorithm to solve the HCCOPs. The proposed algorithm has four main features: 1. A multi-chromosome structure 2. An initial solution generator, 3. A level-barrier based crossover operator and 4. A level-barrier based mutation operator A multi-chromosome structure is utilized in the proposed algorithm to represent different

aspects of the solution (when applicable) and to ensure feasibility of the solutions generated during the iterations of the algorithm. The initial solution generator creates feasible initial solutions for the HCCOPs by creating multiple partial solutions that satisfy lower level of hierarchical constraints and then combining these partial solutions together to satisfy the next higher level of hierarchical constraints. In the level-barrier based crossover operator, feasibility of the solutions is ensured by switching the positions of a gene, which constraints the same dependent variable, in two individuals. In the level-barrier based mutation operator, the location of multiple genes, which have the same length and constraint the same dependent variable, is switched.

In the following chapter, it is first demonstrated how AJSSP can be classified as HCCOP. Next, the proposed algorithm is utilized to optimize various AJSSP and its performance is compared to that of other algorithms used to solve AJSSP. A complexity analysis of the proposed algorithm is also performed.

CHAPTER 4. OPTIMIZATION OF ASSEMBLY JOB-SHOP SCHEDULING PROBLEM

In this chapter, the algorithm developed for HCCOPs in CHAPTER 3 is utilized for the optimization of AJSSP. It is first demonstrated how AJSSP can be classified as HCC problems using the definitions developed in the CHAPTER 3. Next, the initial solution generator, the level-barrier based crossover and mutation operators are utilized to demonstrate the creation of a feasible initial solution and feasible solutions during the iterations of the algorithm. To validate the performance of the proposed algorithm, it is used to optimize multiple AJSSP and its performance is compared against the performance of other algorithms utilized to optimize AJSSP. Lastly, a complexity analysis of the proposed algorithm is performed.

4.1 Introduction to AJSSP

AJSSP, an extension of the classical job-shop scheduling problems, has attracted researchers' attention since it is commonly encountered in the production industry. In AJSSP, *n* ideal jobs, J_1, J_2, …, J_n, of varying processing times are assigned to *m* resources (usually machines). The objective is to allocate the available resources to the jobs to maximize (or minimize) performance indicators such as makespan, tardiness etc. However, unlike classical JSSP, AJSSP processes end-items with bill of materials (BOM). The BOM generates complex precedence structure in the form of assembly tress that exist on multiple levels. End-items are produce through the fabrication of succeeding and assembly operations. Each of these succeeding operations and their components may have their own

specific BOM and routing. Higher level items cannot be processed until all the lower level items have been processed and assembled. Due to the complex precedence structures of operations issues related to production synchronization, coordination and pacing rise naturally in AJSSP and are much more impacting for scheduling than those evidenced in classical JSSP [43].

Fattahi *et al.* [44] used a hierarchical branch and bound algorithm for minimizing the makespan of hybrid flow shop scheduling problems with assembly steps. The authors evaluated the performance of their algorithm under different bottleneck conditions and observed that the best results were obtained when the assembly stage was the bottleneck. Komaki and Kayvanfar [45] proposed several dispatching rules, lower bounds (LB) modified and applied Grey Wolf Optimizer for minimizing the makespan of two-stage asssembly flow shop scheduling problem with release time. The dispatching rules were based on release times, job processing times, and assembly times while the lower bound was based on workload of each stage of the assembly flow shop. The authors modified GWO and incorporated a local search algorithm to improve the performance of GWO. The experimental results showed that the proposed LB, dispatching rules and GWO could solve the assembly flow shop problem effectively. Yan, Wan and Xiong [46] proposed a hybrid variable neighborhood search – electromagnetism-like mechanism (VNS-ESM) algorithm optimize the weighted sum of maximum makespan, earliness and lateness in two-stage assembly flow shop scheduling problem. When applied to a variety of two-stage assembly flow shop problems, the authors observed that their proposed algorithm could outperform electromagnetism algorithm (EM) and variable-neighborhood search algorithm (VNS) while having the same running time. Komaki, Teymourian and Kayvanfar [47] used a

hybrid artificial immune algorithm (HAIS) to minimize the makespan of a two-stage assembly hybrid flow shop scheduling problem. The authors first proposed a LB for the two-stage assembly hybrid flow shop scheduling problem and then developed a heuristic to solve the problem. When compared to existing methods, the proposed algorithm had better performance. Wong and Ngan [48] used hybrid Genetic Algorithm (GA) and hybrid particle swarm optimization (PSO) to minimize the makespan for an AJSSP. When applied to a variety of problems, the HGA outperformed the HPSO. Nataranjan *et al.* [49] proposed and evaluated multiple priority dispatching rules for multi-level AJSSP. In their study, they changed the weight of the jobs and the utilization percentage of machines to minimize the flowtime and the tardiness. Paul *et al.* [50] combined dispatching rules such as First Come First Serviced (FCFS), Shortest Processing Time (SPT) etc. and Order Review/Release policies such as Interval Release (IR), Maximum Load (MXL) etc. to optimize three objectives i.e. mean flow time, mean tardiness, and machine utilization for AJSSP. Chan *et al.* [51] proposed a GA-based approach with lot sharing (LS) technique to solve AJSSP. Compared to a previous algorithm they had developed to solve AJSSP, their new algorithm had better performance for single objective optimization problems. Guo *et al.* [52] used combination of mathematical model and GA with a new chromosome representation, a heuristic initialization process and modified crossover and mutation operators for solving AJSSP. Thaigarajan and Rajendran [53] developed dispatching rules by incorporating the cost of holding and tardiness of jobs to solve dynamic AJSSP. Wang *et al.* [54] proposed a new repair operator based GA to repair the infeasible solutions when solving AJSSP. Liao, Lee, and Lee [55] used mixed integer programming (MIP) to optimize the makespan of a two-stage assembly scheduling problem with batch setup times. The authors first

proposed an effective heuristic and to validate its performance, they applied it to a real-life scheduling problem. The results of the case study showed that it performed better than the scheduling system used in the shop floor. Seidgar *et al.* [56] used imperialist competitive algorithm (ICA) to minimize a weighted sum of makespan and mean completion time for a two-stage assembly flow shop with parallel machines in the first stage. The authors also used Artificial Neural Networks (ANNs) to calibrate the parameters of the proposed ICA. To validate their proposed algorithm, the authors compared its performance to that of a could theory-based simulated annealing algorithm (CSA) and observed that the proposed ICA had better performance. Seidgar *et al.* [57] proposed a simulated imperialist competitive algorithm to minimize the makespan in a two-stage assembly flow shop with machine breakdowns and preventive maintenance. The authors calibrated the parameters of the algorithm using ANNs and when applied to a case study, the proposed algorithm outperformed GA while having the same computational time. Dileeplal and Narayanan [58] used Tabu Search (TS) and GA for the multi-objective optimization of multiple stage assembly job shop scheduling problems with multiple products.

To validate the proposed methodology, it is utilized to solve two different case studies of AJSSP optimization. In the 1st case study, an AJSSP with reentrant steps and parallel machines is considered and the results obtained from the proposed methodology are compared to the results obtained when standard GA (SGA) is used to solve the same problem. In the 2nd case study, the proposed algorithms performance is measured against the performance algorithms used by Dileeplal and Narayanan. In this case study, the 28 test problems introduced by Dileeplal and are was used to compare the two works. The

problems in the 1st case study are also used to demonstrate the HCCs in AJSSP as well as modelling of the solution using the algorithm developed in CHAPTER 3.

4.2 Case Study 1

As mentioned above, in this case study an AJSSP with reentrant steps and parallel machine was considered. A description of the problems is given below:

1. On the shop floor, there are machines groups available. Each machine group has 3 parallel machines.
2. Each job had multiple parts being produced. As such, a job can be split into smaller batches and each batch can use a separate machine belonging to the same machine group
3. No pre-emption of job operation is permitted
4. The set-up times are not considered in these problems.
5. Machines never break down.
6. The machine group used for each operation of a job is known.
7. A job can revisit a machine group for multiple operations.
8. Operation constraints and assembly constraints must always be followed.

The objective of the proposed algorithm and SGA is to find the optimal sequence of operations and the number of machines to use for each operation of a job that would minimize the makespan. To compare the two algorithms for simulated problems are used with different number of jobs, levels, and machine groups to simulate varying degrees of difficulty. The information about these problems is presented in Table 1 and the structure of the jobs or BOM is shown in Figure 12.

Table 1. Information of the AJSSP used in case study 1

Problem type	# of levels	# of Jobs	# of Operations per Job	# of machine groups
A	2	7	3	2
B	2	11	3	2
C	3	23	3	3
D	4	35	3	3

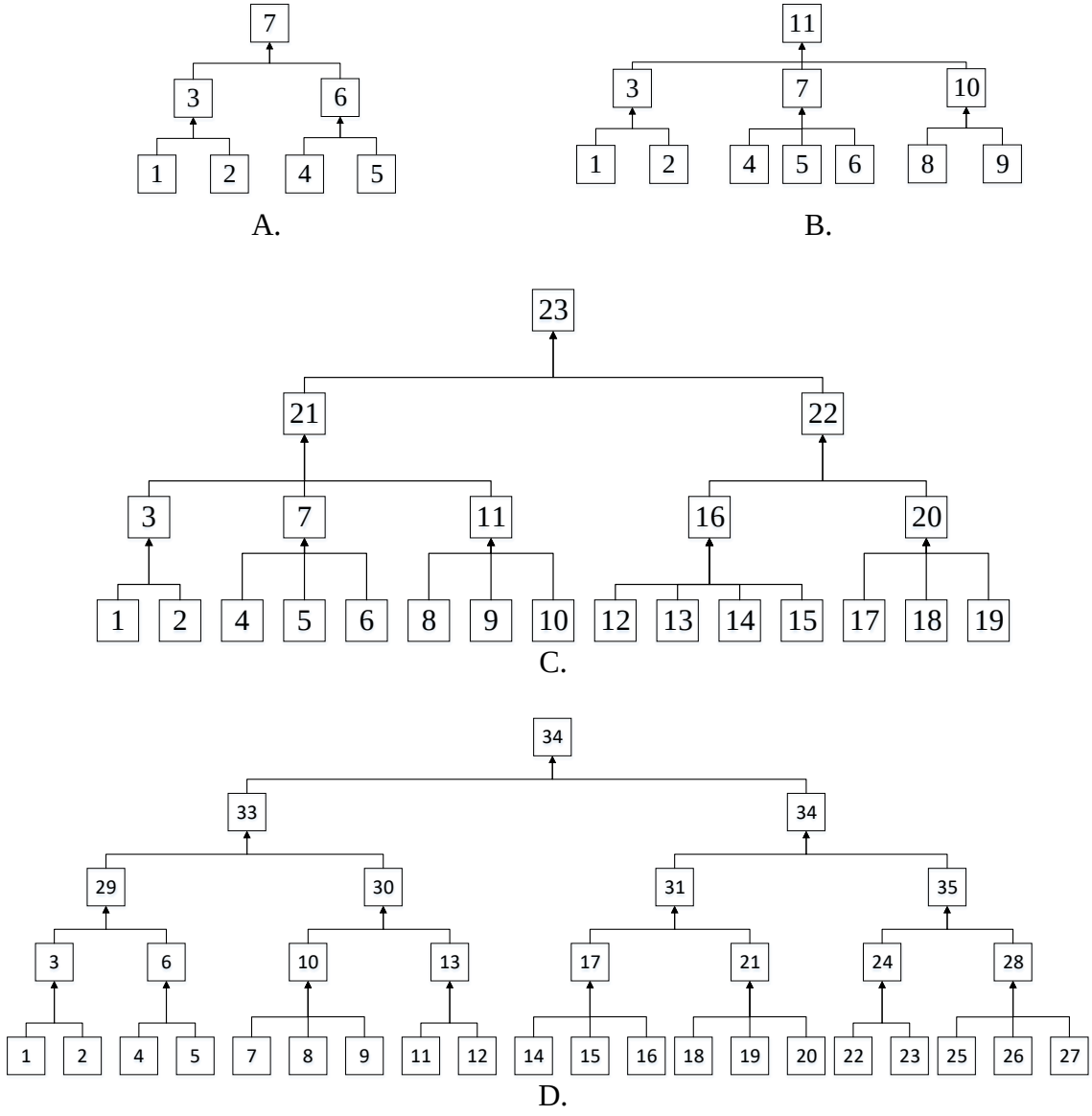

Figure 12. BOM of the four different simulated testing problems

4.2.1 Modelling of Solution

For the 1st case study, each feasible solution needs two parts, the sequence of steps (1st sub-chromosome) and the number of parallel machines that each step can utilize (2nd sub-chromosome). These two parts are comparatively independent; therefore, every individual is encoded as a double-chromosome. The two sub-chromosomes are of equal

length number and the i^{th} location of sub-chromosome two represents the number of parallel machines used by the step in the i^{th} location of sub-chromosome one.

To eliminate the generation of infeasible solutions, permutation encoding is utilized in the 1st sub-chromosome. The frequency of occurrence of each job in the 1st sub-chromosome, which corresponds to the number of operations of each job consist of regular constraints. The location of each job in the 1st chromosome, which corresponds to the sequence of jobs, can be regarded as the HCCs. Each component of the 2nd sub-chromosome is constrained by a lower and upper bound, which corresponds to the minimum and maximum number of parallel machines the operation in the corresponding position in the 1st sub-chromosome is allowed to utilize during its production. Integer encoding is utilized for the 2nd sub-chromosome. Equation (19)-(22) show the constraints for problem B.

$$\min\{l_3^{(A,1)}\} \geq \max\{l_1,\ l_2\}+1 \tag{19}$$

$$\min\{l_7^{(A,1)}\} \geq \max\{l_4,\ l_5,\ l_6\}+1 \tag{20}$$

$$\min\{l_{10}^{(A,1)}\} \geq \max\{l_8,\ l_9\}+1 \tag{21}$$

$$\min\{l_{11}^{(A,1)}\} \geq \max\{l_3\ ,l_7,\ l_{10}\}+1 \tag{22}$$

Per Equation (19), the first occurrence of 3 in the 1st sub-chromosome of individual A must be after the last occurrence of 1 and 2 in i.e. the processing of job 3 can only be started after the completion of jobs 1 and 2. Similarly, according to Equation (20), the

processing of job 7 can only be started after the completion of jobs 4, 5, and 6. According to equation (21), the processing of job 10 can only started after the completion of jobs 8 and 9. Lastly, according to Equation (22), the processing of job 11 can only be started after the completion of jobs 3, 7, and 10. By combining Equations (19)-(22) it can also be said that the starting time of job 11 is not only constrained by jobs 3, 7, and 10 but also by jobs 1, 2, 4, 5, 6, 8, and 9. Therefore, according to the definitions developed in section 2, Equation (22) represents a HCC.

4.2.1.1 Initial solution set generation

As discussed in CHAPTER 3, to create a solution, the 0^{th} level genes are first created then the 1^{st} level then the 2^{nd} and so on. For problem B, the dependent variables constraint the 3^{rd} job are jobs 1 and 2. Therefore a 0^{th} level gene of length six is created which consists 1's and 2's. Next the operations of job 3 are appended in front of this 0^{th} level gene to create a feasible 1^{st} level gene. Similar steps are taken for jobs 7, and 10. To create the 1^{st} sub-chromosome, the three 1^{st} level genes are combined and the operations of job 11 are appended at end. Figure 13 shows an example of the 3 first level genes and the completed 1^{st} sub-chromosome.

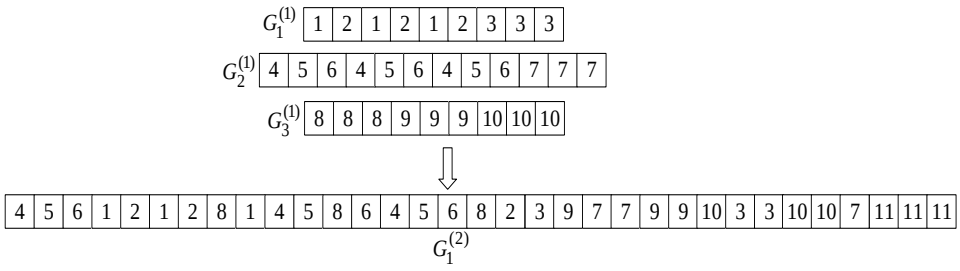

Figure 13. An example of the 1st sub-chromosome for problem type B

For the 2nd sub-chromosome, the only constraints are the limitations on the value of the variables, therefore, it can be created by generating a random array of integers with values ranging between 1 and 3. The combination of 1st and the 2nd sub-chromosome creates a complete solution as shown in Figure 14.

1st sub-chromosome

4	5	6	1	2	1	2	8	1	4	5	8	6	4	5	6	8	2	3	9	7	7	9	9	10	3	3	10	10	7	11	11	11

2nd sub-chromosome

1	2	1	3	2	1	2	3	1	3	2	2	1	2	2	3	1	2	3	1	3	3	1	1	1	3	3	2	2	1	3	3	3

Figure 14. A complete, feasible solution for problem type B

The above procedure can be followed to make the entire initial solution set.

4.2.1.2 Level-barrier based crossover operator

To ensure the feasibility of the solution after crossover, the level barrier based crossover operator is used for the 1st sub-chromosome. As outlined earlier, in level barrier based crossover operator, the location of two identical genes from two individuals (individuals A and B), are switched. In Figure 15, the gene [4,5,6,7] is selected, and its position is switched in the two individuals. For the 2nd sub-chromosome, standard crossover can be used as there are no HCCs.

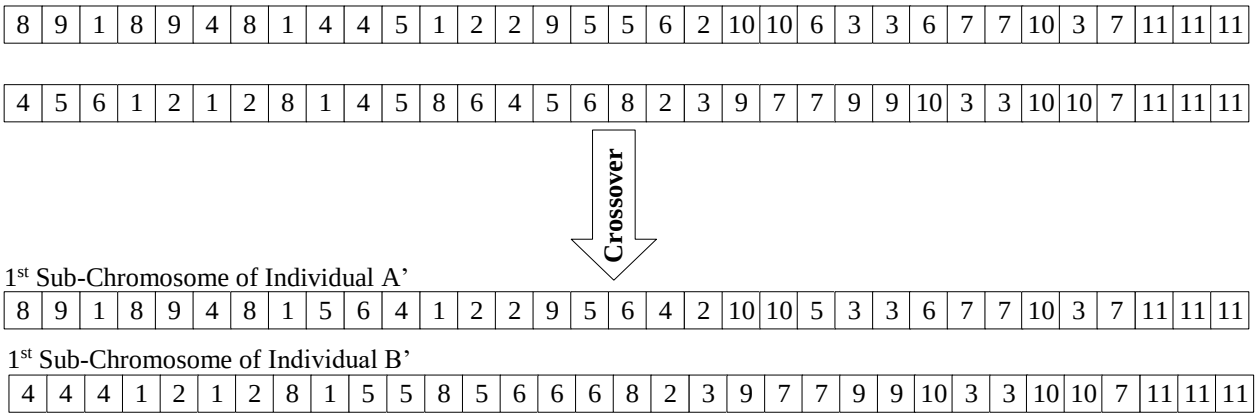

Figure 15. An example of the 1st sub-chromosome created after the level-barrier based crossover operator

4.2.1.3 Level-barrier based mutation operator

As mentioned in CHAPTER 3, in the level barrier based mutation operator, the location of multiple genes of the same length of the (*k*-1) level that constraint the same k^{th} level dependent variables are switched to ensure the HCCs. Figure 16 shows an example of the level barrier based mutation operator. In the example, the 1st level is chosen for mutation and as there are only two genes in the 1st level that have the same length i.e. [1,2,3] and [8,9,10] their locations are switched. The level barrier based mutation operator only needs to be utilized for the 1st sub-chromosome due to the absence of HCCs from the 2nd sub-chromosome.

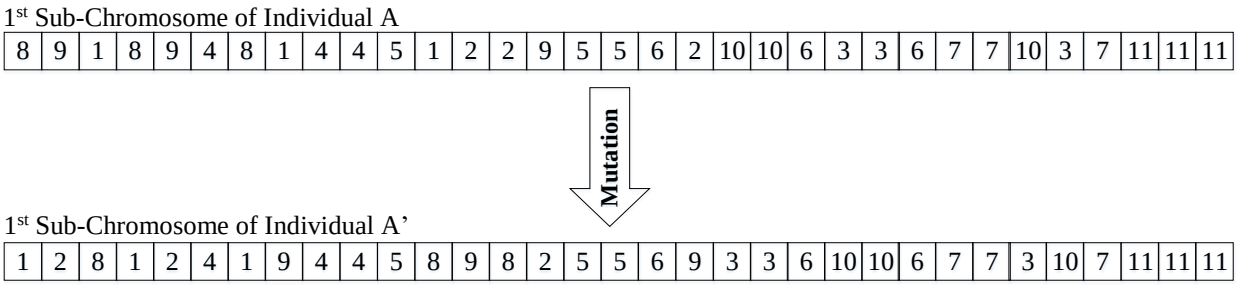

Figure 16. Example of level-barrier based mutation operator

As SGA by itself is unable to cope with HCCs and operation constraints, linear constraints were used to ensure the feasibility of solution after creation, crossover, and mutation. The linear constraints are given by:

$$Ax \leq B \tag{23}$$

Where A nxm matrix of 0's, 1's, and -1's, x is a mx1 vector, representing the sequence of steps, and B is a mx1 vector of -1's, where m is the number of total operations for all the jobs and n is the number of HCC and other constraints. For example, if the jobs [1,2,3] are considered, SGA has to ensure that the in any possible solution x, *the* 1st operation of job 1 has to be performed before the 2nd of job 1 and the 2nd operation of job 1 is performed before the 3rd operation of job 1. At the same time, the 3rd operation of job 1 and 2 have to be performed before the 1st operation of job 3. For this example, the matrix A and vector B would take the following form:

$$A=\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \end{bmatrix} \qquad B=\begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

So, the first row of the matrix *A* and vector *B* indicates that the difference between the sequence of the 1st operation of job 1 and the 2nd operation of job 1 has to be less than or equal to -1. Therefore, it is noticeable that for a large scale AJSSP the matrix *A* will be extremely large and finding unique solutions that satisfy the constraints would be extremely hard.

4.2.2 Result and Comparison

During the comparison, the number of generations and population size is varied until the SGA was able to find a solution. The settings are used for the proposed algorithm. The different settings of the number of generations and population sizes are shown in Table 2. The results obtained for the four different problem types are shown in Table 3.. For SGA and the proposed method, the crossover rate is 0.95 and the mutation probability is 0.1.

Table 2. Parameter settings used for SGA and the proposed method

Parameter Settings	Number of Generations	Population Size
1	50	50
2	100	500
3	500	500

Table 3. Comparison of SGA and the proposed method for the four different types of AJSSP

Problem	Setting	Makespan (Generation in which smallest makespan was first found)	
		Proposed Algorithm	SGA
A	1	54 (6)	No Solution Found
	2	53 (14)	53 (63)
B	1	85 (7)	No Solution Found
	2	82 (40)	82 (74)
C	1	180 (2)	No Solution Found
	2	167 (25)	No Solution Found
	3	167 (24)	171 (248)
D	1	290 (28)	No Solution Found
	2	279 (40)	No Solution Found
	3	275 (26)	289 (278)

In problem type A, when the number of generations and population is low (50 each), SGA is unable to obtain a sequence of operations that satisfied all the HCCs and operation constraints while the proposed algorithm is able to find a solution with a makespan of 54 units. When the number of generations is increased to 100 and population size to 500, SGA in able to obtain a makespan of 54 units (found in the 63rd generation) while the proposed method is able to find a makespan of 52 units (found in the 14th generation). For problem type B, when the number of generations and population are low, SGA is again unable to find an optimal sequence of operations while the proposed algorithm is able to find a makespan of 85 units. When the number of generations and population are increased to 100 and 500 respectively, both the SGA and the proposed method give a makespan of 82 units (found in the 74th and 42nd generation respectively).

In problem type C, SGA is unable to find a solution when the number of generations and population are 50 each and also when the number of generations and population are 100 and 500 respectively. However, when the same settings are used for the proposed algorithm, it gives a makespan of 180 and 167 units respectively. When the number of generations is further increased to 500, SGA gives a makespan of 171 units (found in the 248^{th} generation) while the proposed method gives a makespan of 167 units (found in the 24^{th} generation). Lastly, for problem type D, SGA is again unable to find a solution for the first 2 settings but found a solution that gave a makespan of 289 units (found in the 478^{th} generation). The proposed method on the other hand, gives a makespan of 290, 279, and 275 units for the 3 settings (found in the 24^{th}, 28^{th}, and 26^{th} generation respectively). These results indicate that as the complexity of the problem increases, SGA is unable to find a solution that satisfies all the constraints with low number of generations and population, however, the proposed method is always able to find a solution, as the HCCs integrated into the method proposed. Although SGA is able to find a solution after increase the number of generations and population size, these solutions are found in the later generations, unlike the proposed method.

The convergence speed (in terms of number of generations) is also observed at a specific population size, crossover rate, mutation probability, and generations for each problem. The parameter settings used for each problem are given in Table 4 while Figure 17 shows the convergence plot for each problem.

Table 4. Parameter settings used to generate the convergence plots for the proposed algorithm

Problem	Generations	Population Size	Crossover Rate	Mutation Probability
A	100	50	0.95	0.10
B		50		
C		100		
D		100		

Higher population size is used for Problems C and D due to the higher complexity of these problems. As it can be seen from Figure 17, for Problem A, the proposed algorithm converges (very little to no improvement in the mean fitness between generations) to a makespan of 54 time units after 15 generations, for Problem B, it converges to a makespan of 84 time units after 20 generations, for Problem C, if converges to a makespan of 170 time units after 30 generations, and for Problem D, it converges to a makespan of 285 time units after 45 generations. These results show that regardless of the size of the problem, the proposed algorithm is able to converge rapidly to the best solutions found (given in Table 3).

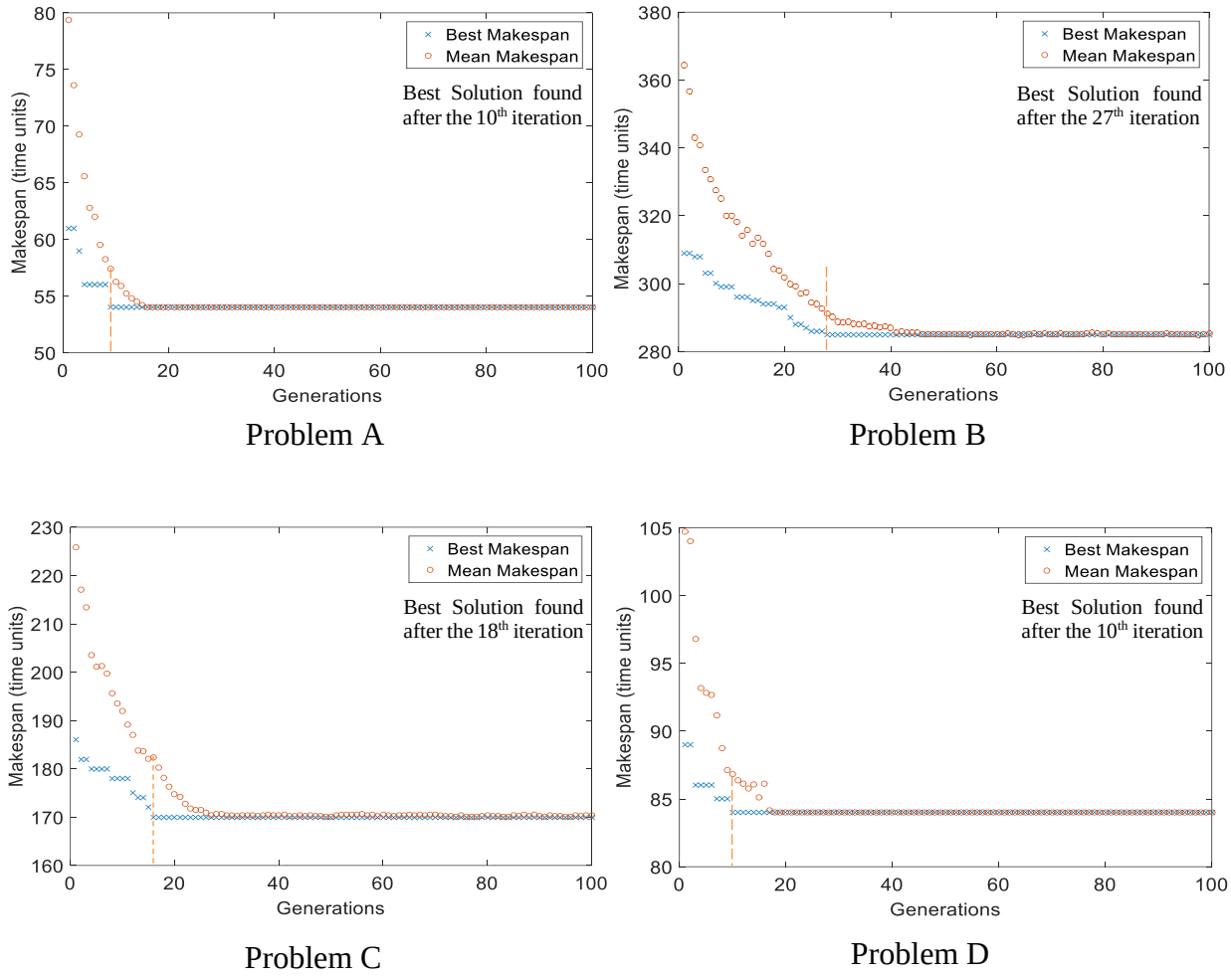

Figure 17. Converge plots of Problems A, B, C, and D. The best solution was found after 10 generation for problems A and D, 18 generations for problem C, and 27 generations for Problem B

Next, for Problem D, different settings of crossover and mutation rate are used, while keeping the generations and population size constant, to study their effects on the makespan for the problem. Problem D is chosen as it was the most complex of the four problems. The settings used are given in Table 5, while the corresponding plots are shown in Figure 18.

.

Table 5. The different crossover and mutation rates used to study their effects on the makespan for Problem D

Parameter Settings	Generations	Population Size	Crossover Rate	Mutation Probability
I			0.75	0.10
II	100	100	0.85	0.20
III			0.95	0.30

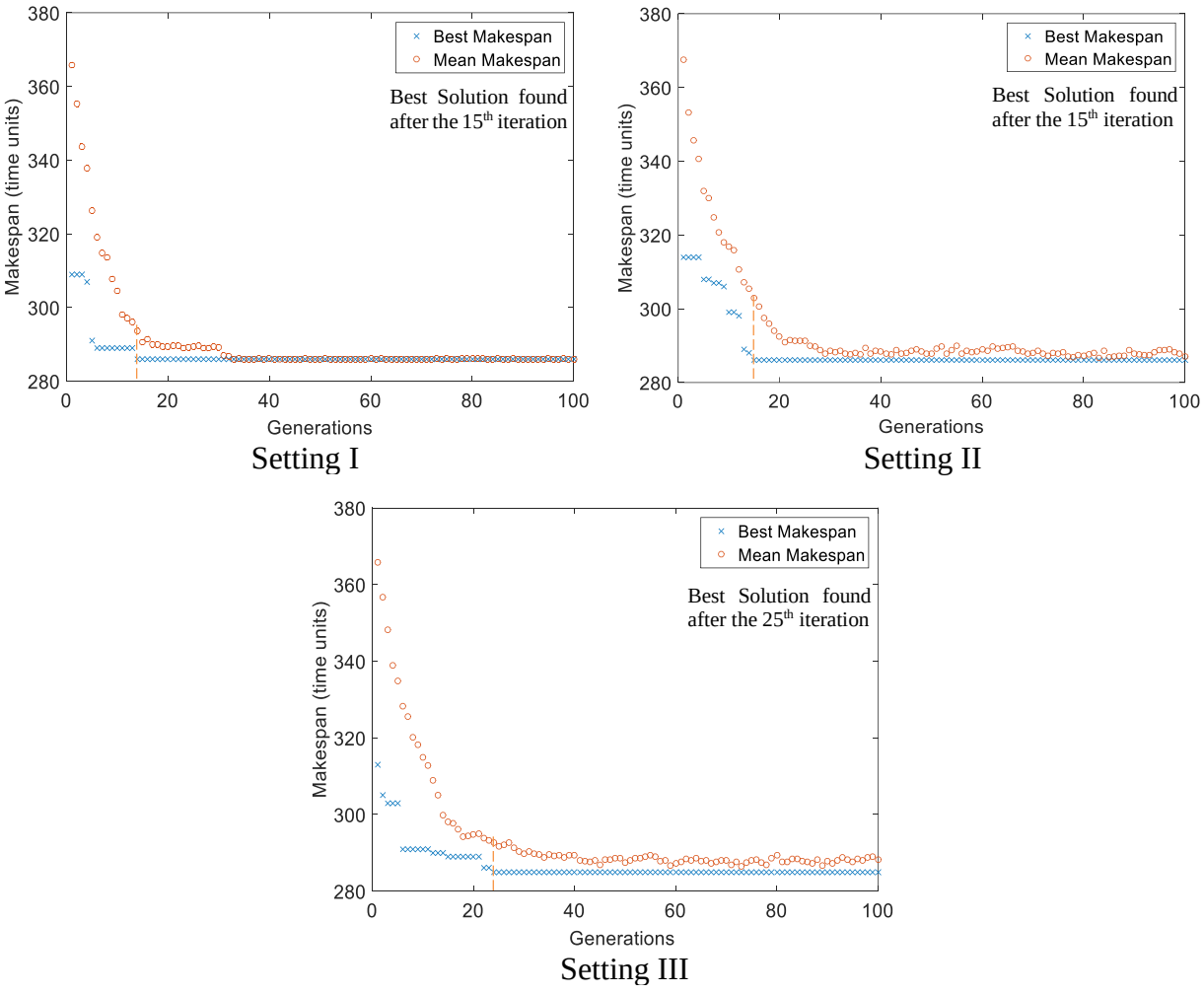

Figure 18. Converge plots obtained using parameter settings I, II, and III for Problem D

As it can be seen from Figure 18, when a relatively low mutation probability was used (parameter settings I), the proposed algorithm found a makespan of 286 time units

after 15 generations. When the mutation probability was increased to 0.20 (parameter settings II), there was a larger difference between the mean fitness value and the best fitness value compared to the parameter settings I, but the proposed algorithm was able to find a best solution of 286 times units after 15 generations, similar to the makespan obtained using parameter settings I. When the mutation probability was further increased, as in parameter settings III, the difference between the mean and best fitness value increased but the proposed algorithm was still able to find a makespan of 285 time units though this makespan was achieved after 25 generations. It can be observed from these results that regardless of the parameter settings used; the proposed algorithm was always able to converge to an optimized makespan that is close to best makespan obtained (275 time units), which demonstrates the robustness of the proposed algorithm.

To further evaluate the performance of the proposed algorithm, 27 additional AJSSP are generated with 9 problems having 2 assembly levels, 9 problems having 3 assembly levels, and 9 problems having 4 assembly levels. The proposed algorithm was used to optimize each problem 30 times and the average percent deviation (APD) was recorded for each problem, given by Equation (24). The objective of these simulations was to verify the stability of the proposed algorithm i.e. observe whether the algorithm could converge to the "best solution" when applied to the same problem multiple times. The "best makespan" mentioned in Equation (24) refers to the best makespan obtained by the proposed algorithm. The settings of the 27 test problems are given in Table 6. For the 27 test problems, an iteration limit of 100, population size of 100, crossover rate of 0.95, and mutation probability of 0.1 were used. Table 7 shows the results obtained for the problems having 2 levels of assembly, Table 8 shows the results obtained for the problems having 3

levels of assembly, and Table 9 shows the results obtained for the problems having 4 levels of assembly.

$$APD = \frac{100}{t}\sum_{i=1}^{t}\left|\frac{makespan_t - best\ makespan}{best\ makespan}\right| \quad (24)$$

Table 6. Additional settings used for the 27 test problems

Range of assembly levels	[2, 4]
Range of jobs required to create an assembly	[1,4]
Range of number of operations in each job	[1,4]
Range of processing time for each operation (time units)	[5,20]

Table 7. Results obtained for the AJSSP with 2 levels of assembly. The highlighted results show that an APD of 0% was achieved

Problem #	Total Number of Jobs	Operations per Job	Best Makespan Found (time units)	Average Makespan Found (time units)	Average Percentage Deviation (%)
1	9	3	181.00	182.50	0.81
2	**10**	**2**	**111.00**	**111.00**	**0.00**
3	**12**	**1**	**150.00**	**150.00**	**0.00**
4	12	1	**173.00**	**173.00**	**0.00**
5	13	3	198.00	198.30	0.13
6	**15**	**2**	**206.00**	**206.00**	**0.00**
7	17	4	278.00	286.50	3.05
8	18	3	250.00	252.60	1.03
9	18	3	280.00	282.40	0.87

Table 8. Results obtained for the AJSSP with 3 levels of assembly. The highlighted results show that an APD of 0% was achieved

Problem #	Total Number of Jobs	Operations per Job	Best Makespan Found (time units)	Average Makespan Found (time units)	Average Percentage Deviation (%)
1	24	3	367.00	367.37	0.10
2	**26**	**1**	**350.00**	**350.00**	**0.00**
3	29	4	465.00	472.77	1.67
4	**34**	**2**	**487.00**	**487.00**	**0.00**
5	40	2	392.00	392.53	0.14
6	40	4	580.00	588.57	1.48
7	**41**	**1**	**550.00**	**550.00**	**0.00**
8	**43**	**2**	**587.00**	**587.00**	**0.00**
9	**45**	**1**	**528.00**	**528.00**	**0.00**

Table 9. Results obtained for the AJSSP with 4 levels of assembly. The highlighted results show that an APD of 0% was achieved

Problem #	Total Number of Jobs	Operations per Job	Best Makespan Found (time units)	Average Makespan Found (time units)	Average Percentage Deviation (%)
1	53	3	717.00	717.33	0.05
2	**63**	**2**	**864.00**	**864.00**	**0.00**
3	**63**	**3**	**832.00**	**832.00**	**0.00**
4	**67**	**1**	**840.00**	**840.00**	**0.00**
5	**69**	**1**	**911.00**	**911.00**	**0.00**
6	71	3	929.00	929.58	0.06
7	**72**	**2**	**911.00**	**911.00**	**0.00**
8	**80**	**1**	**1057.00**	**1057.00**	**0.00**
9	**80**	**3**	**908.00**	**908.00**	**0.00**

As it can be observed from Table 7, for problems with 2 levels of assembly, the algorithm had an APD of 0% for 4 out of the 9 problems and had an APD of less than 4% for the remained of the 5 problems. For the problems with 3 levels of assembly (Table 8), the algorithm achieved an APD of 0% for 5 out of the 9 problems and had a APD of less than 2% for the remaining 4 problems. Finally, for problems with 4 levels of assembly (Table 9), the algorithm had an APD of 0% for 7 out of the 9 problems and an APD of less than 1% for the remaining 2 problems. These results show that regardless of the size of the problem, the algorithm was able to achieve a very low APD, thus verifying its stability.

4.3 Case Study 2 of AJSSP

In the 2nd case study, the performance of the proposed algorithm was measured against the performance the GA and Tabu Search (TS) algorithms used by Dileeplal and Narayanan for the optimization of AJSSP. Dileeplal and Narayanan created 28 test AJSSP problems to evaluate the performance of their algorithm. In each of the test problem, multiple products were being created on a shop-floor and each product was created through an assembly process. Each product on the shop floor had either a flat (single level of assembly with 3-12 components per assembly), tall (2-6 level assemblies with 2-4 components per assembly), or complex structure (2-3 level of assembly with 2-4 components per assembly). The authors evaluated the performance of their algorithm based on three performance indicators: 1. Makespan, 2. Tardiness, and 3. Weighted sum of makespan and Tardiness. In this paper, the proposed algorithm was used to optimize the makespan of the 28 AJSSP mentioned in Dileeplal and Narayanan's works and the results obtained using the proposed algorithm were compared to those obtained by Dileeplal and

Narayanan. Table 10 compares the results obtained using the two different methods. In Table 10, the % improvement is obtained through Equation (25).

$$\% \text{ improvement} = \frac{\text{change in makespan}}{\text{makespan obtained using proposed algorithm}} \quad (25)$$

Where,

$$\text{change in makespan} = \text{makespan obtained using Deleeplals' algorithm} - \text{makespan obtained using proposed algorithm} \quad (26)$$

Table 10. Comparison of the performance of the proposed algorithm to that of Dileeplal and Narayanan's algorithm proposed algorithm

Problem #	Makespan obtained using the proposed algorithm (time units)	Best makespan obtained by Dileeplal and Narayanan (time units)	% improvement
1	78	77	-1.28
2	**88**	**90**	**2.27**
3	**93**	**100**	**7.52**
4	**110**	**110**	**0.00**
5	**135**	**135**	**0.00**
6	**125**	**125**	**0.00**
7	**113**	**118**	**4.42**
8	**128**	**129**	**0.78**
9	**134**	**134**	**0.00**
10	**131**	**154**	**17.50**
11	**162**	**171**	**5.50**
12	174	165	-5.17
13	232	228	-1.74
14	226	221	-2.21
15	233	228	-2.14
16	**237**	**241**	**1.68**
17	**90**	**91**	**1.11**

Table 10 continued.

Problem #	Makepsan obtained using the proposed algorithm (time units)	Best makespan obtained by Dileeplal and Narayanan (time units)	% improvement
18	**98**	**104**	**6.12**
19	**86**	**89**	**3.48**
20	**97**	**97**	**0.00**
21	**107**	**108**	**0.93**
22	**121**	**121**	**0.00**
23	146	139	-4.79
24	**171**	**171**	**0.00**
25	**177**	**177**	**0.00**
26	**199**	**199**	**0.00**
27	205	202	-1.46
28	**215**	**215**	**0.00**

As it can be observed from Table 10, the proposed algorithm was able to find an equal or better makespan than that reported by Dileeplal and Narayanan on 21 out of the 28 problems. The proposed algorithm was able to improve upon the makespan found by Daleeplal and Narayanan by as high as 17%. Even though it was unable to find the best makespan for 7 test problems, the proposed algorithm was able to find a solution within 5% of the solution obtained by Dileeplal and Narayanan. These results indicate that the proposed algorithm can cope with AJSSP with multiple products being produced.

4.4 Complexity Analysis of the Algorithm

The proposed algorithm is an evolutionary computation method that is applied to several different AJSSP problems of varying size. Therefore, it is necessary that the algorithm perform efficiently in real time to consistently function without needing significant amount of computing resource. The time complexity of the proposed algorithm

is analyzed by observing the real-time performance of the proposed algorithm on different problems of varying sizes.

In the AJSSP, the size of the problem can be significantly altered by varying two features: 1). the number of jobs and 2). the number of assembly levels. Therefore, to evaluate the time complexity of the proposed algorithm, two groups of simulation were designed. In the first simulation, 16 test problems were created and each problem had equal number of assembly levels but different number of total job. In the second simulation, 6 test problems were created and each test problem had equal number of jobs however, the total number of assembly levels in the test problems were varied. Next, the proposed algorithm was used to optimize the test problems with an iteration limit of 100 and the and the runtime was recorded. Finally, the recorded runtime is plotted and polynomial fitting was used to estimate the growth function of the running time. The time complexity of the algorithm is then determined from the fitted growth function. Figure 19-A shows the actual running time (in seconds) versus the number of iterations and the fitted model used to predict the behavior of the runtime for simulation set 1 while Figure 19-B shows it for simulation set 2.

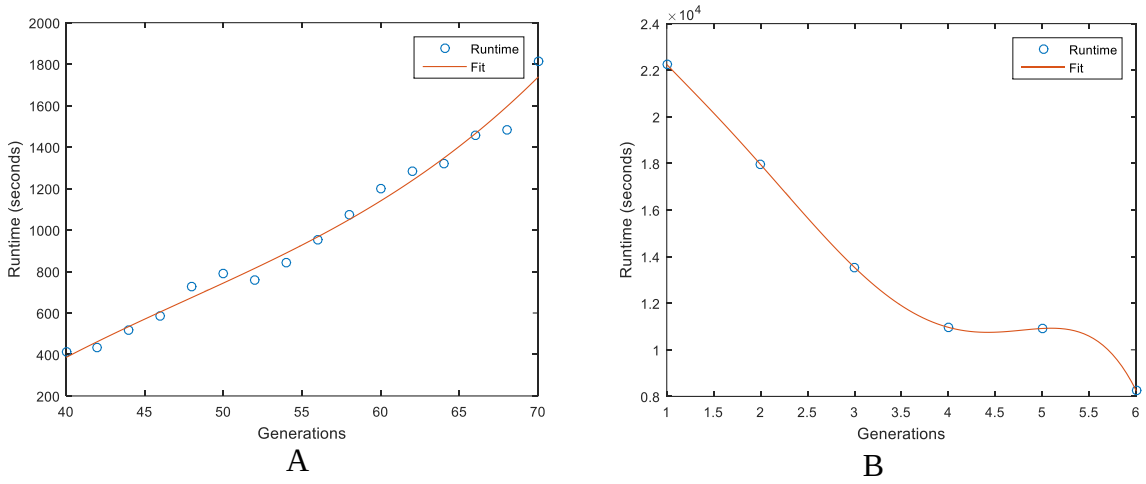

Figure 19. Polynomial fitting of runtime versus number of generations for A. Simulation Set 1 and B. Simulation Set 2

As it can be seen from Figure 19, the algorithm runs in polynomial time of some degree, and the fitted models estimate the order of polynomial time the algorithm runs in for each simulation set. For simulation set 1, the time complexity is $O(n^3)$ indicating that the algorithm runs in polynomial time of degree 3. This shows that as the total number of jobs increase while the number of assembly levels are kept constant, the runtime of the algorithm increases. However, for simulation set 2, the complexity is - $O(n^5)$ indicating that as the number of assembly levels are increased while the total number of jobs kept constant, the runtime of the algorithm decreases in polynomial time of degree 5. Since all the problems in simulation set 2 have the exact same chromosome length (determined by the total number of jobs), the feasible solution space decreases as the number of assembly levels increase. This consequently leads to a decrease in the runtime of the algorithm as the proposed algorithm only needs to search a smaller region for an optimal solution.

4.5 Summary

In this chapter, the algorithm proposed in CHAPTER 3 for HCCO is utilized for the optimization of AJSSP. Its performance is validated by using it to optimize several different AJSSP. In Section 4.1, AJSSP is introduced along with the previous work done in optimization of AJSSP.

Section 4.2 presents the first case study used to validate the performance of the proposed algorithm. In this section, several different AJSSP cases are formulated with different number of assemblies, jobs, and processing times. These problems are first utilized to demonstrate how AJSSP can be classified as a HCCOP. Next, the operators of the proposed algorithm are used to demonstrate the generation of a feasible initial solution as well as generation of feasible solutions during the iterations of the algorithm for the optimization of AJSSP. The performance of the proposed algorithm is compared to the performance of SGA on the formulated problems and the results show that the proposed algorithm has superior performance compared to SGA. The APD of the proposed algorithm is also less than 2% which indicates its stability. The convergence plots for selected problems also indicate the robustness of the proposed algorithm.

In Section 4.3, the performance of the proposed algorithm is measured against the algorithm proposed by Dileeplal *et al.* The AJSSP cases proposed by Dileeplal *et al.* are used to compare the two algorithms and the results show that the proposed algorithm was able to find a better solution for 21 out of the 28 problems. The proposed algorithm was able to improve upon the makespan by as much as 17%.

In Section 4.4, a time complexity analysis of the proposed algorithm is performed by varying the number of assembly levels and the number of jobs. The algorithm has a time complexity of $O(n^3)$ with respect to the number of jobs and a time complexity of - $O(n^5)$ with respect to the number of assembly levels.

CHAPTER 5. SOLVING HIERARCHICALLY COUPLED CONSTRAINT PROBLEM IN SIMULTANEOUS OPTIMIZATION OF NEURAL NETWORK STRUCTURE AND WEIGHTS

In this chapter, the methodology developed for the optimization of HCCOPs in CHAPTER 3 is utilized for the simultaneous optimization of Neural Network (NN) structure and weights. It is first demonstrated how simultaneous optimization of NN structure and weights can be classified as a HCCOP using the definitions developed in the CHAPTER 3. Next, the initial solution generator, the level-barrier based crossover and mutation operators are utilized to demonstrate the creation of a feasible initial solution and feasible solutions during the iterations of the algorithm. To validate the performance of the proposed algorithm, it is used to create a NN model for four different problems and its performance is compared against other algorithms used to create a NN.

5.1 Introduction

NNs [59], are a soft computing techniques that are commonly used in a variety of fields for the purpose of input-output mapping due to their ability to map from one multivariable space to another and to approximate functions according to the desired degree of accuracy. Another major advantage of NN is that the shape of the approximation function does not have to be assumed before training. During the training phase of the NN, its weight and bias values are updated with the aim of minimizing the difference between the predicted and the actual values. Over the years, many researchers have used different

techniques for training the weight and bias values to ensure convergence to optimal or near-optimal solutions. These techniques include Genetic Algorithm (GA) [60,61,62,63,64], Artificial Bee Colony (ABC) [65, 66, 67, 68], Particle Swarm Optimization (PSO) [69,70,71,72] etc. Though these techniques have significantly improved the prediction capabilities of the NNs, determining the optimal structure of NN (number of hidden layers, hidden neurons in each hidden layer, and transfer function of each hidden layer) is still a tedious task. Determining the optimal NN structure is an important task the correct NN structure can increase its prediction accuracy tremendously. However, NN structure are highly problem dependent and are determined by human experts using a trial-and-error based approach. Due to the large combination of the NN parameters, finding the optimal NN structure through a trial-and-error based approach is infeasible and as such, several researchers have attempted to create an algorithm for NN structure and weight optimization.

Loghmanian *et al.* [73] used a multi-objective genetic algorithm for the structure optimization of NN. In their study, the number of hidden nodes and input and output lags were optimized, while the number of hidden layers were kept constant. Shao and Li [74] used PSO to optimize the number of hidden nodes and weights of NN. Their proposed method was used for finding the best NN structure that would create a prediction model for real world medical problems. Mendivil *et al.* [75] optimized the number of hidden layers and number of hidden neurons by using binary encoding with fixed chromosome length of GA. Kopel [76] also used GA to optimize the number of hidden layers, number of hidden neurons, and the range of the initial weight values of NN by using binary encoding with fixed chromosome length.

Though these techniques have made the process of finding the optimal NN structure easier, they have the following drawbacks associated with them:

1. They do not allow for flexibility of number of neurons in each hidden layer.
2. They do not allow for optimization of both structure and weights simultaneously.
3. The chromosomes have a fixed length. This indicates that part of the chromosome can contain irrelevant information.
4. The number of hidden layers is fixed and the connections between the hidden layers are altered.

To overcome these drawbacks, the algorithm proposed in CHAPTER 3 is utilized for the simultaneous optimization of NN weights and structure. However, as it will be seen later on in this chapter, the operators proposed in CHAPTER 3 have been slightly modified due to the nature of the problem and to ensure feasible solutions throughout the iterations of the algorithm. In order to apply the proposed algorithm to the problem under consideration, it is also assumed that the NN is a fully connected.

5.2 HCCs in NN weight and structure optimization

The NN structure and weight optimization problem comprises of constants (problem specific) and variables (both independent and dependent). The constants of the problem are the following:

1. Number of inputs, *input.*
2. Number of outputs, *output.*

3. Number of biases of the output layers, $\eta\beta_{output}$.

The NN structure and weight optimization has the following independent variables:

1. Numbers of neurons in the 1st hidden layer, α_1.

The 1st level dependent variables are then given by:

1. Number of weighted connections, $nw_{1,input}$, from the input layer to the 1st hidden layer. These weighted connections are represented by $w_{1,input}$. $nw_{1,input}$ and $w_{1,input}$ are given by:

$$nw_{1,input} = \begin{cases} 0 \text{ if } \alpha_1 = 0 \\ input * \alpha_1 \text{ if } \alpha_1 > 0 \end{cases} \tag{27}$$

$$w_{1,input} = \begin{cases} empty\ matrix\ if\ \ \alpha_1 = 0 \\ \alpha_1 * input\ matrix\ if\ \ \alpha_1 > 0 \end{cases} \tag{28}$$

2. Transfer function of the 1st hidden layer, TF_1. Where,

$$TF_1 = \begin{cases} \text{N/A if } \alpha_1 = 0 \\ \text{Hyperbolic- tangent or log sigmoid if } \alpha_1 > 0 \end{cases} \tag{29}$$

3. Number of biases, $\eta\beta_1$, in the 1st hidden layer. These biases are represented by β_1 where:

$$\beta_1 = \begin{cases} empty\ matrix\ if\ \ \alpha_1 = 0 \\ \alpha_1 * 1\ matrix\ if\ \ \alpha_1 > 0 \end{cases} \tag{30}$$

4. Numbers of neurons in the 2nd hidden layer, α_2.

$$\alpha_2 = \begin{cases} 0 \text{ if } \alpha_1 = 0 \\ 0,1,2,...k_2 \text{ if } \alpha_2 > 0 \end{cases} \tag{31}$$

5. Number of weighted connections, $nw_{output,input}$ from the input layer to the output layer. These weighted connections are represented by $w_{output,input}$. $w_{output,input}$ and $nw_{output,input}$ is given by:

$$nw_{output,input} = \begin{cases} 0 \text{ if } \alpha_1 > 0 \\ input * output \text{ if } \alpha_1 = 0 \end{cases} \tag{32}$$

$$w_{output,input} = \begin{cases} empty\ matrix\ if\ \ \alpha_1 > 0 \\ \alpha_1 * input\ matrix\ if\ \ \alpha_1 = 0 \end{cases} \tag{33}$$

The 2nd level dependent variables are then given by:

1. Number of weighted connections, $nw_{2,1}$, from the 1st hidden layer to the 2nd hidden layer. These weighted connections are represented by $w_{2,1}$. $nw_{2,1}$ and $w_{2,1}$ are given by:

$$nw_{2,1} = \begin{cases} 0 \text{ if } \alpha_2 = 0 \\ \alpha_1 * \alpha_2 \text{ if } \alpha_2 > 0 \end{cases} \tag{34}$$

$$w_{2,1} = \begin{cases} empty\ matrix\ if\ \ \alpha_2 = 0 \\ \alpha_2 * \alpha_1\ matrix\ if\ \ \alpha_2 > 0 \end{cases} \tag{35}$$

2. Transfer function of the 2nd hidden layer, TF_2. Where,

$$TF_2 = \begin{cases} \text{N/A if } \alpha_2 = 0 \\ \text{Hyperbolic- tangent or log sigmoid if } \alpha_2 > 0 \end{cases} \tag{36}$$

3. Number of biases, $\eta\beta_2$, in the 2nd hidden layer. These biases are represented by β_2 where:

$$\beta_2 = \begin{cases} empty\ matrix\ if\ \ \alpha_2 = 0 \\ \alpha_2 * 1\ matrix\ if\ \ \alpha_2 > 0 \end{cases} \tag{37}$$

4. Numbers of neurons in the 3rd hidden layer, α_3.

$$\alpha_3 = \begin{cases} 0 \text{ if } \alpha_2 = 0 \\ 0,1,2,...k_3 \text{ if } \alpha_2 > 0 \end{cases} \tag{38}$$

5. Number of weighted connections, $nw_{output,1}$ from the 1st hidden layer to the output layer. These weighted connections are represented by $w_{output,1}$. $w_{output,1}$ and $nw_{output,1}$ is given by:

$$nw_{output,1} = \begin{cases} 0 \text{ if } \alpha_2 > 0 \\ \alpha_1 * output \text{ if } \alpha_2 = 0 \end{cases} \tag{39}$$

$$w_{output,1} = \begin{cases} empty\ matrix\ if\ \ \alpha_2 > 0 \\ \alpha_1 * output\ matrix\ if\ \ \alpha_2 = 0 \end{cases} \tag{40}$$

The k^{th} level dependent variables are then given by:

1. Number of weighted connections, $nw_{k,(k-1)}$ from the (k-1) layer to the k^{th} hidden layer. These weighted connections are represented by $w_{k,(k-1)}$ and $nw_{k,(k-1)}$ is given by:

$$nw_{k,(k-1)} = \begin{cases} 0 \text{ if } \alpha_k = 0 \\ \alpha_{(k-1)} * \alpha_k \text{ if } \alpha_k > 0 \end{cases} \tag{41}$$

$$w_{k,(k-1)} = \begin{cases} empty\ matrix\ if\ \ \alpha_k = 0 \\ \alpha_k * \alpha_{(k-1)}\ \ if\ \ \alpha_k > 0 \end{cases} \tag{42}$$

2. Transfer function of the k^{th} hidden layer, TF_k . Where,

$$TF_k = \begin{cases} \text{N/A if } \alpha_k = 0 \\ \text{Hyperbolic- tangent or log sigmoid if } \alpha_k > 0 \end{cases} \tag{43}$$

3. Number of biases, $\eta\beta_k$, in the k^{th} hidden layer. These biases are represented by β_k where:

$$\beta_k = \begin{cases} empty\ matrix\ if\ \ \alpha_k = 0 \\ \alpha_k * 1\ matrix\ if\ \ \alpha_k > 0 \end{cases} \tag{44}$$

4. Numbers of neurons in the (k+1) hidden layer, $\alpha_{(k+1)}$.

$$\alpha_{(k+1)} = \begin{cases} 0 \text{ if } \alpha_k = 0 \\ 0,1,2,...k_{(k+1)} \text{ if } \alpha_k > 0 \end{cases} \tag{45}$$

5. Number of weighted connections, $nw_{output,(k-1)}$ from the (k-1) hidden layer to the output layer. These weighted connections are represented by $w_{output,(k-1)}$. $w_{output,(k-1)}$ and $nw_{output,(k-1)}$ is given by:

$$nw_{output,(k-1)} = \begin{cases} 0 \text{ if } \alpha_k > 0 \\ \alpha_{(k-1)} * output \text{ if } \alpha_k = 0 \end{cases} \tag{46}$$

$$w_{output,k} = \begin{cases} empty\ matrix\ if\ \ \alpha_k > 0 \\ \alpha_k * output\ matrix\ if\ \ \alpha_k = 0 \end{cases} \tag{47}$$

5.3 Modelling of the solution

5.3.1 Initial solution creation

To create an array of in the initial solutions, the following steps must be taken:

1. Create three empty sub-chromosome A, B, and C. Sub-chromosome A contains information about the weighted connections between the layers, sub-chromosome

B contains information about the bias values of each layer, and sub-chromosome C contains information about the transfer function of each hidden layer.

2. Determine the number of hidden layers by generate a random integer between 1 and γ_{max} (maximum number of hidden layers).
3. Generate a random integer, α_1, between 1 and α_{max} (maximum number of hidden neurons allowed in any hidden layer) to determine the number of neurons in the 1st hidden layer.
4. For sub-chromosome A, generate $nw_{1,input}$ random numbers between -1 and 1 that occupy the positions 1 through $nw_{1,input}$. Call these the 1st level gene of sub-chromosome A.
5. For sub-chromosome B, generate $n\beta_1$ random numbers between -1 and 1 that occupy the 1 though $n\beta_1$positions. Call these the 1st level gene of sub-chromosome B.
6. For sub-chromosome C, generate a random number between 0 and 1. A value of 0 represents hyperbolic-tangent sigmoid and a value of 1 represents log-sigmoid transfer function. This value will occupy the 1st position of the 1st level gene of sub-chromosome C.
7. Repeat steps 3-6 until the (i-1) genes for sub-chromosome A, B, and C have been created. Append all the genes of each sub-chromosome together in increasing numerical values.
8. For the output layer, for sub-chromosome A, generate $nw_{i,output}$ random numbers between -1 and 1, and append them in front of sub-chromosome A. For sub-chromosome B, generate $n\beta_{output}$ random numbers between -1 and 1 and append

them in front of sub-chromosome B. For sub-chromosome C, assign a value of 2, indicating a linear transfer function and append this value in front of sub-chromosome C.

9. Append all the sub-chromosomes together to create the final chromosome.

An example of the initial solution generator is shown below in Figure 20.

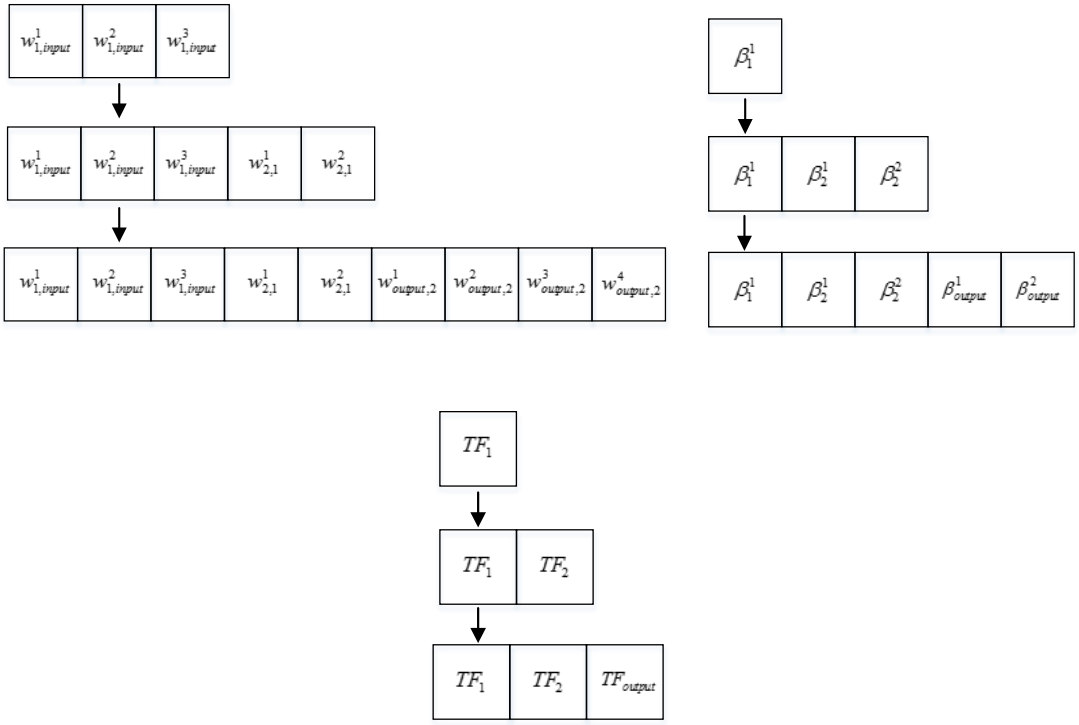

Figure 20. Example showing creation of initial sub-chromosomes with a 3-1-2-2 NN structure. Sub-chromosomes *A* is top left, *B* is top right, and *C* is bottom

A key characteristic of the creation method is that since the *total layers* of each initial solution can be different, each solution can have different number of gene levels. Therefore, if the level-barrier based crossover operators from Chapter 3 are used, they will produce infeasible solutions in subsequent generations. Let's take an example of crossover being performed on the 1st level gene of sub-chromosome *A* of a 3-1-2-2 and a 3-2-2 NN

structure. The 1st level gene of sub-chromosome *A* of the 3-1-2-2 structure will contain 3 numbers representing the weight values from the input layer to the 1st hidden layer, while the 1st level gene of sub-chromosome *A* of the 3-2-2 NN structure will contain 6 numbers representing the weight values from the input layer to the 1st hidden layer. If the previously developed level-barrier based crossover technique is used here, the solution will become infeasible as the 1st level gene of the 3-1-2 NN structure will have extra weight values while the 3-2-2 NN structure will have insufficient weight values. This example is illustrated in Figure 21. When the crossover method is used for sub-chromosome *B*, infeasible solutions can also be created. To avoid these scenarios, a modified version of the level-barrier based crossover operator developed in CHAPTER 3 is proposed below.

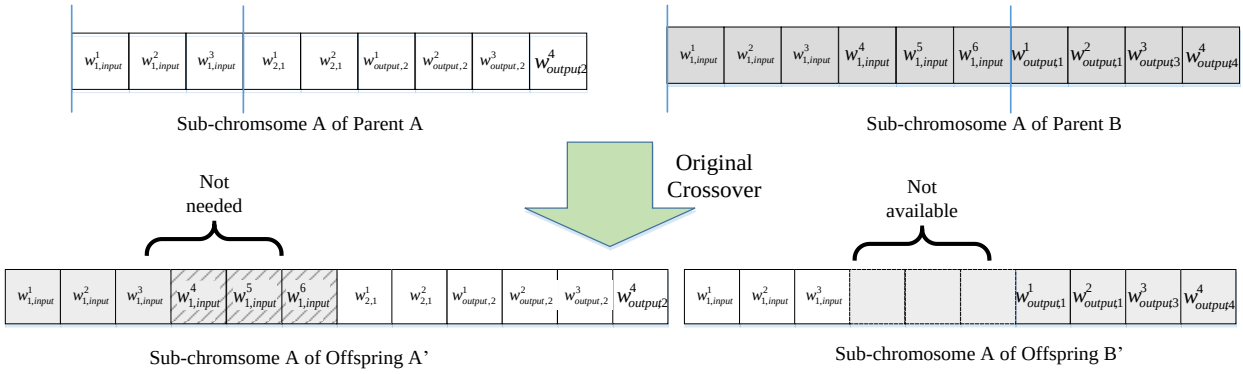

Figure 21. Example of infeasible solution created using the level-barrier based crossover proposed in Chapter 3

5.3.2 Modified level-barrier based crossover operator

The steps of the improved crossover operator are:

1. Select a pair of parents, PA_1 and PA_2, to perform crossover based on roulette wheel selection.

2. Determine the number of levels for each parent, G_{PA_1} and G_{PA_2} respectively.
3. Generate a random integer between 1 and G_{PA_1} or G_{PA_2} (whichever is smaller) to determine the level on which crossover is performed,
4. For sub-chromosome A, determine the length of gene for PA_1 and PA_2 on the crossover level, LG_{PA_1} and LG_{PA_2} respectively. Generate a random integer, RC_1, between 1 and LG_{PA_1} or LG_{PA_2} (whichever is smaller). Switch the bits of the two parents in the crossover level located in the position 1 to RC_1.
5. Repeat step 4 for sub-chromosome B.
6. For sub-chromosome C, switch the bits on the crossover level between the two parents.
7. Repeat steps 3-6 for all the pairs of parents.

An example of crossover between a 3-1-2-2 and a 3-2-2-2 NN structures is shown below in Figure 22.

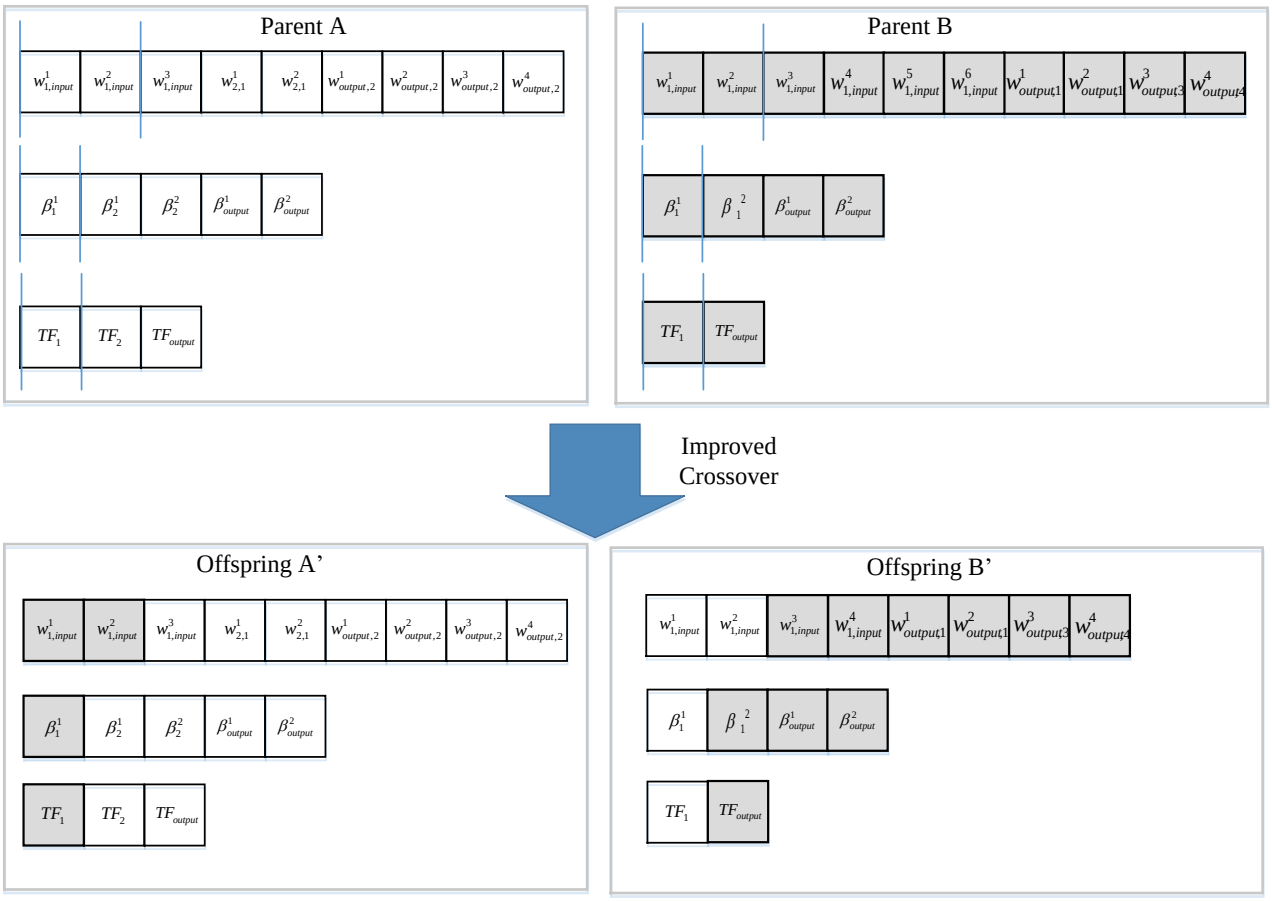

Figure 22. Figure showing an example of the modified level-barrier based crossover operator between 3-1-2-2 and a 3-2-2 NN structures

5.3.3 Modified level-barrier based crossover operator

To use the mutation operator mentioned in Chapter 3, each i^{th} level gene must be comprised of several genes from the (i-1) level as well as the dependent variables of the i^{th} level. As mentioned above, in the NN structure and weight optimization problem, only a single gene from the (i-1) and the dependent variables of the i^{th} level make up the gene of the i^{th} level. Therefore, the level-barrier based mutation operator developed in Chapter 3 *cannot* be used in NN structure and weight optimization. Since the purpose of mutation is to diversify the population, a possible way to achieve this is by:

1. Addition of a hidden neuron in a hidden layer.
2. Removal of a hidden neuron from a hidden layer
3. Addition of a hidden layer
4. Removal of a hidden layer

In each of the scenario, the length of sub-chromosomes *A, B, and C* are adjusted accordingly such that the number of weight values correspond to the new NN structure. An example of how the solution structure would change if a NN structure is mutated from 3-1-2-2 to 3-2-2-2 is shown in Figure 23.

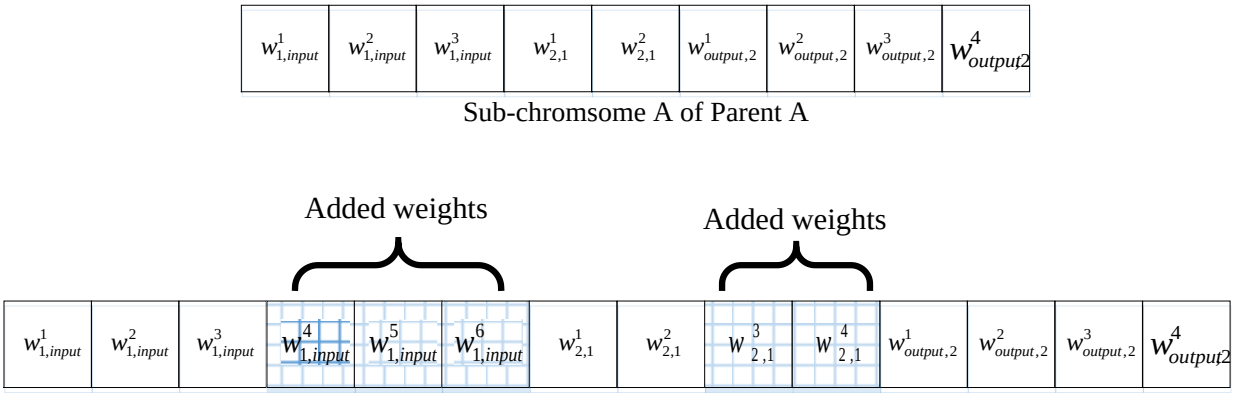

Figure 23. Offspring created after mutation of a 3-1-2-2 to a 3-2-2-2 NN structure

5.4 Application of the modified algorithm to Case Studies

In this section, the modified version of the algorithm was utilized to create a NN for a variety of case studies in order to validate its performance. The case studies included prediction of:

1. Roller length of hydraulic jump on a rough channel bed

2. Surface roughness in CNT nanofluids based grinding process
3. Tensile strength of friction stir weld joints
4. Surface roughness in end face milling process

The parameters of the modified algorithm used for building the NN in these case studies are given in Table 11.

Table 11. Parameters used to create a trained NN in the case studies

Number of Generations	Initial Population Size	Crossover Rate	Mutation Rate	Elite Children	$\alpha_{\max}$	$\gamma_{\max}$
100	100	0.85	0.15	5%	3	10

During the training procedure, it was noticed that since both the structure and the weights of the NN were being optimized simultaneously, the proposed algorithm had difficulty converging to optimal weight values. Therefore, to improve local convergence, once the NN weights and structure were trained using the proposed algorithm, the weights were further trained using Levenberg-Marquardt (LM) [77] algorithm. The data sets in the case studies were divided into training, validation, and testing data sets. During the optimization process, the NN structure and weight values were changed with respect to the mean squared error (MSE), given by Equation (48), of the training data set.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2 \tag{48}$$

To avoid overfitting, the NN structure and weight values that gave the lowest MSE for the validation data set were used. Once the NN weight and structure was completed, its prediction accuracy was validated using the testing data set.

5.4.1 Case Study 1

Azimi *et al.* [78] used an adaptive neuro-fuzzy inference system (ANFIS) proposed by Jang [79] for prediction of roller length of hydraulic jump on a rough channel bed. They trained the ANFIS using a firefly algorithm (FA) [80] to overcome the drawbacks of ANFIS trained using backpropagation. Hydraulic jumps are caused due to the transformation of supercritical to subcritical flow regime. These hydraulic jumps usually occur downstream of structures such as ogee spillways, control gates, and weirs and are used for water purification, irrigation, prevent air locking etc. Identification of hydraulic jumps is important for energy dissipation below hydraulic structures and roller length is one of the key parameters that can help in the accurate identification of the hydraulic jumps [81]. Through their literature review, Azimi *et al*. found that Froude number (Fr), ratio of bed roughness height (*ks*) to flow depth upstream of hydraulic jump (h_1), and ratio of flow depth downstream of hydraulic jump (h_2) to upstream as the factors possibly affecting the ratio roller length to flow depth upstream. In their study, Azimi *et al*. created four different models with different combinations of input parameters to find the best model for predicting the ratio of roller length to flow depth upstream of hydraulic pump. The four models were:

$$\frac{Lr}{h_1} = f\left(Fr, \frac{h_2}{h_1}, \frac{ks}{h_1}\right) \tag{49}$$

$$\frac{Lr}{h_1} = f\left(\frac{h_2}{h_1}, \frac{ks}{h_1}\right) \tag{50}$$

$$\frac{Lr}{h_1} = f\left(Fr, \frac{ks}{h_1}\right) \tag{51}$$

$$\frac{Lr}{h_1} = f\left(Fr, \frac{h_2}{h_1}\right) \tag{52}$$

The data used to create the models was obtained from a study conducted by Carollo *et al.* [82] and was split into training (70%) and testing (30%) data sets by Azimi *et al*. Five indicators were used to measure the performance of created models: mean absolute percentage error (MAPE), root-mean-square error (RMSE), correlation coefficient (R), scatter index (SI), and BIAS. The results obtained in their study were compared to the NN created using the modified HCGA algorithm. To train the NN-HCCP-lm model, the data was split into training (50%), validation (20%), and testing (30%) data sets. The results are shown in Table 12.

Table 12. Comparison of results for the test data sets obtained by Azami *et al.* and the proposed algorithm

Model #	Algorithm	MAPE (%)	RMSE	SI	BIAS	R
1	Proposed Algorithm	6.01	1.53	0.08	0.16	0.97
	ANFIS-FA	7.61	1.77	0.09	0.19	0.97
	% improve	21.04	13.61	14.23	11.87	0.39
2	Proposed Algorithm	6.48	1.74	0.09	0.19	0.97
	ANFIS-FA	9.93	2.20	0.12	0.23	0.95
	% improve	34.71	21.02	23.88	15.79	1.58
3	Proposed Algorithm	7.41	1.90	0.10	0.20	0.96
	ANFIS-FA	10.07	2.62	0.14	0.25	0.93
	% improve	26.36	27.54	28.76	18.69	3.06

Table 12 continued.

Model #	Algorithm	MAPE (%)	RMSE	SI	BIAS	R
4	Proposed Algorithm	8.28	1.95	0.10	0.18	0.97
	ANFIS-FA	10.22	2.19	0.11	0.23	0.95
	% improve	18.94	11.12	7.99	21.40	1.78

As it can be seen from Table 12, for all the models, the NN created using the proposed algorithm had higher prediction capabilities than ANFIS-FA algorithm. The range of increase in MAPE, RMSE, SI, BIAS, and R were 18.94% (4^{th} model) to 34.71% (2^{nd} model), 11.12% (4^{th} model) to 27.54% (3^{rd} model), 7.99% (4^{th} model) to 28.76% (3^{rd} model), 11.87% (1^{st} model) to 21.40% (4^{th} model), and 0.39% (1^{st} model) to 3.06% (3^{rd} model) respectively.

5.4.2 Case Study 2

Grinding is a versatile finishing technique that can produce parts with a very high surface finish commonly and is commonly used in the industry. In recent years, nanofluids have replaced traditional macro fluids as the nanofluids have higher heat transfer capabilities and also produce better surface finish after grinding. Accurately predicting the surface roughness in grinding based on input parameters is an important topic as it can help minimize the number of defected parts and increase the production efficiency. To overcome this challenge, Prabhu et al. [ ] used regression analysis, neural networks, and fuzzy logic to model the relationship between the process parameters (speed, feed, and depth of cut) and the performance indicators (surface roughness with and without nanofluids). In their study, Prabhu et al. carried out grinding of AISI D3 Tool steel based on L8 orthogonal array of Taguchi design of experiments. Each of the input parameters

had 2 levels therefore, a total of 8 experiments were conducted with different combinations of the input parameters. To study the effect of nanofluids on the grinding process, another 8 experiments were performed without the use nanofluids. Once the experimental data was accumulated, the three different techniques mentioned above were used to create an input-output model and study the effects of different parameters on the surface finish. When creating a prediction model with the proposed algorithm, 6 data points were used for training, 1 for validation, and 1 for testing. The relative error obtained using the proposed algorithm as well as those obtained by Prabhu et al. are shown in Table 13 and Table 14.

Table 13. Comparison of predicted surface roughness in the absence of nanofluids

Experiment #	Regression relative error (%)	Classical NN relative error (%)	Fuzzy logic relative error (%)	Proposed algorithm relative error (%)
1	10.86	2.68	1.32	2.63
2	4.15	0.05	1.79	20.51
3	8.51	4.32	1.62	0.00
4	0.34	11.31	0.34	0.00
5	0.67	2.78	12.81	1.75
6	6.01	1.05	0.71	0.00
7	3.72	1.11	1.37	0.00
8	10.87	3.5	7.5	0.00
MAPE (%)	5.64	3.35	3.43	3.11

Table 14. Comparison of predicted surface roughness in the presence of nanofluids

Experiment #	Regression relative error (%)	Classical NN relative error (%)	Fuzzy logic relative error (%)	Proposed algorithm relative error (%)
1	5.87	1.92	2.31	0.00
2	3.40	2.00	2.00	0.00
3	6.63	1.92	9.23	3.85
4	1.48	10.30	3.70	0.00
5	5.22	6.36	1.52	0.00

Table 14 continued.

Experiment #	Regression relative error (%)	Classical NN relative error (%)	Fuzzy logic relative error (%)	Proposed algorithm relative error (%)
6	1.10	5.83	2.78	0.00
7	0.06	4.75	5.00	5.00
8	7.79	0.29	0.88	5.88
MAPE (%)	3.94	4.17	3.43	1.84

As it can be seen from Table 13 and Table 14, the NN created using the proposed algorithm had better prediction capabilities than regression analysis, classical NN, and fuzzy logic. The MAPE obtained using proposed algorithm (3.11%) was 44.82% better than the MPAE obtained using regression analysis, 7.09% better than MAPE obtained using classical NN, and 9.32% better than the MAPE obtained using fuzzy logic. When nanofluid was used, the NN created using the proposed algorithm still had better prediction accuracy than the three techniques used by Prabhu *et al.* The MAPE of 1.84% obtained using the proposed algorithm was 53.31%, 55.86%, and 46.29% better than the MAPE obtained using regression analysis, classical NN, and fuzzy logic respectively.

5.4.3 Case Study 3

Friction-stir-welding (FSW) is a solid-state joining process in which two facing surfaces are joined together by application of heat and followed by the use of mechanical pressure. A major challenge in FSW is selection of welding parameters that would minimize the defect and create a joint with high strength. As stated by Dewan *et al.* [83] the quality of the FSW joint can be affected by both welding process parameters such as, spindle speed, welding speed, plunge depth etc., and environmental factors such as welding

cooling, pre-weld cooling, weld location etc. Since the relationship between the process parameters and weld quality can be highly non-linear and complex and finding the optimal process parameters that would increase the weld quality is very important, the development of an accurate prediction model is necessary. To overcome these challenges, Dewan *et al.* used ANFIS to create a forward prediction model between the process parameters i.e. spindle speed (N), welding speed (V), plunge force (F_z), and empirical force index (EFI) and the performance indicators i.e. ultimate tensile strength (UTS) of the weld. In their experimental procedure, Dewan *et al.* ran 73 experiments while varying the 4 process parameters to observe their effects on the UTS of the weld. In the first step of their modeling procedure, Dewan *et al.* only used a combination of spindle speed, welding speed, and plunge force to predict the UTS of the weld. They also varied the membership function of the input parameters to create an accurate prediction model. RMSE and MAPE of the testing set (30% of all experimental data) was used to evaluate the prediction capabilities of the trained ANFIS. To create an NN prediction model using the proposed algorithm, 43 data sets were used for training, 15 were used for validation, and 15 for testing. A comparison of the results is shown in Table 15.

Table 15. Comparison of results obtained using ANFIS and the proposed algorithm with spindle speed, welding speed, and plunge force as inputs

Model #	Inputs			RMSE			MAPE (%)		
	N	V	F_z	ANFIS	Proposed Algorithm	% improve	ANFIS	Proposed Algorithm	% improve
1	X			50.74	37.26	26.57	14.70	10.34	29.66
2		X		50.80	34.53	32.03	15.17	9.19	39.42
3			X	51.60	39.64	23.18	15.99	11.8	26.20
4	X	X		50.40	37.04	26.51	14.65	10.95	25.26
5		X	X	43.29	36.49	15.71	12.45	11.00	11.65
6	X		X	45.56	33.38	26.73	13.67	9.72	28.90
7	X	X	X	36.87	30.27	17.90	10.92	8.46	22.53

As it can be seen from Table 15, for each of the models, the NN created using the proposed algorithm was able to predict the outputs to a higher degree of accuracy than the ANFIS as indicated by the RMSE and MAPE values. The range of change in MAPE and RMSE was 11.65% to 39.42% and 15.71% to 32.03% respectively. To increase the accuracy of the prediction model Dewan *et al.* then updated their models by included EFI as an input. The combinations of inputs were again varied to find the best combination of inputs that would create a prediction model with the highest predictive capabilities. The same combinations were used in the process of building a NN using the proposed algorithm. These results are shown in Table 16.

Table 16. Comparison of results obtained using ANFIS and the proposed algorithm with EFI as an additional input

Model #	Inputs				RMSE			MAPE (%)		
	N	V	F_z	EFI	ANFIS	Proposed Algorithm	% improve	ANFIS	Proposed Algorithm	% improve
1				X	36.51	34.80	4.68	9.90	7.99	19.29
2	X			X	34.76	30.43	12.46	9.34	8.54	8.57
3		X		X	30.84	26.79	13.13	8.28	7.41	10.51
4			X	X	31.82	29.13	8.45	8.78	7.42	15.49
5	X	X		X	31.02	22.84	26.37	7.96	6.13	22.99
6	X		X	X	31.86	24.34	23.60	7.87	7.24	8.01
7		X	X	X	29.70	24.37	17.95	7.75	6.44	16.90
8	X	X	X	X	30.68	27.85	9.22	7.81	7.39	5.38

When EFI was added as an input both the MAPE and RMSE decreased for both the ANFIS and the NN created using the proposed algorithm. The lowest MAPE and RMSE for the ANFIS and the proposed algorithm were 7.75% and 29.70 (7^{th} model) and 6.13% and 22.84 (5^{th} model) respectively. As shown in Table 16, with the addition of the extra input the MAPE and RMSE provided by the proposed algorithm was better than that

provided by ANFIS. The RMSE changed from 4.68% to 26.37% while the MAPE changed from 5.38% to 22.99%.

In their last modeling procedure, Dewan *et al.* used ANN trained using LM to create a forward prediction model. They again varied the combination of input parameters to create the most accurate prediction model. 70% of the total data set available was used for training, 15% was used for validation and 15% was used to test the trained model. The same input combinations and data splits were for the proposed algorithm and the results are shown below

Table 17. Comparison of results obtained using classical NN and the proposed algorithm with EFI as an additional input

Model #	Inputs				RMSE			MAPE (%)		
	N	V	F_z	EFI	NN	Proposed Algorithm	% improve	NN	Proposed Algorithm	% improve
1	X	X	X	X	41.03	24.56	67.06	12.38	7.40	67.30
2		X	X	X	38.28	19.80	93.33	10.64	5.89	80.65
3	X	X	X		46.36	27.14	70.82	12.90	7.76	66.24
4	X	X		X	42.27	21.03	101.00	11.61	4.69	147.55
5	X		X	X	38.87	28.67	35.58	10.85	7.90	37.34
6	X	X			61.79	37.26	65.83	17.39	9.89	75.83
7	X		X		52.52	23.36	124.83	15.18	6.12	148.04
8	X			X	45.62	20.25	125.28	12.59	6.12	105.72
9		X	X		56.85	33.70	68.69	16.05	8.92	79.93
10		X		X	43.50	28.77	51.20	11.57	7.18	61.14
11			X	X	43.31	27.58	57.03	11.95	8.61	38.79
12				X	49.68	30.44	63.21	13.00	8.91	45.90
13			X		68.71	35.32	94.54	19.14	9.38	104.05
14		X			57.14	30.84	85.28	16.46	8.65	90.29
15	X				53.35	28.10	89.86	15.66	8.00	95.75

The NN created using the proposed algorithm proved to be much more accurate at predicting the UTS for FSW than NN trained using LM as seen from Table 17. The RMSE and the MAPE of the classical NN varied from 38.28 to 61.79 and 10.64% to 19.14%

respectively while the RMSE and MAPE of proposed algorithm varied from 21.03 to 37.26 and 5.89% to 9.89%. These results show that the proposed algorithm was better at creating a prediction model with any combinations of prediction models. Compared to classical NN, the proposed algorithm reduced the RMSE from 35.58% to 125.25% and the MAPE from 37.34% to 148.04%.

5.4.4 Case Study 4

As stated earlier, surface roughness is a required specification of machined products and is used to specify the quality of the finished product. Many researchers [84,85,86] have suggested that the main factors that affect surface roughness are feed, speed, and depth of cut. Selecting the optimal combinations of the process parameters is essential in order to maximize the lifespan of both the workpiece and the tool but an accurate prediction is required to achieve this objective. To avoid costly trial-and-error process in machining parameter determination Zhang *et al.* [87], proposed a Gaussian process regression (GPR) for modeling and predicting the surface roughness in end face milling. In their study, Zhang *et al.* first used ran 48 experiments while varying the spindle speed, depth of cut, and feed rate. The experimental data was then divided into training (36) and testing sets (48). The data provided by Zhang *et al.* was used to create NN using the proposed algorithm with 24 sets being used for training, 12 for validation, and 12 for testing. A comparison of the results obtained using the two techniques is shown in Table 18.

Table 18. Comparison of the results obtained using the proposed algorithm and regression analysis

Experiment #	Gaussian process regression relative error (%)	Proposed algorithm relative error (%)	% improve
1	34.41	7.59	77.95
2	16.24	4.29	73.60
3	19.82	7.61	61.59
4	13.61	1.20	91.16
5	2.15	3.86	-79.54
6	12.17	1.92	84.24
7	13.94	19.48	-39.70
8	15.18	31.45	-107.16
9	24.94	24.98	-0.17
10	9.78	2.24	77.12
11	4.26	1.44	66.08
12	22.03	13.21	40.04
MAPE (%)	15.71	9.94	58.08

As it can be observed from Table 8, for experiment # 5, 7, 8, and 9, the relative error of GPR was lower than that of the NN created using the proposed algorithm. However, NN created using the proposed algorithm had a MAPE of 9.94% while GPR had a MAPE of 15.71% for all the 12 experiments. This indicates that overall, the NN created using the proposed algorithm provided a 58.08% improvement over GPR.

5.5 Summary

In this chapter, the algorithm proposed in CHAPTER 3 for HCCO is utilized for the simultaneous optimization of NN structure and weight values. Its performance is validated by using it to create NN for several different modelling problems. In Section 5.1, NNs are introduced along with the previous work done in simultaneous optimization of NN structure and weight values.

Section 5.2 demonstrates how simultaneous NN structure and weight optimization problem can be classified as a HCCOP. In this section, the different variables are introduced as well as the HCC in the NN structure and weight optimization problem. In Section 5.3 shows how the initial solution is created using the proposed algorithm. Next, the modified operators of the proposed algorithm are introduced and they are utilized to demonstrate how feasible solutions are created during the iterations of the algorithm.

In Section 5.4 the proposed algorithm is utilized to create a NN model for the prediction of roller length of hydraulic jump on a rough channel bed, surface roughness in a grinding process, UTS in friction-stir welding process, and surface roughness in end milling process. The results show that prediction model created using the proposed algorithm had better prediction accuracy than prediction models created using classical NN, ANFIS, regression analysis, and gaussian-process regression.

CHAPTER 6. MULTIMODAL OPTIMIZATION OF HIERARCHICALLY COUPLED CONSTRAINT PROBLEMS

In this chapter, a clustering-optimization algorithm is proposed for the MMO of HCCOP. This is achieved by combining *k-means clustering* algorithm with the algorithm proposed for optimization of HCCOP in CHAPTER 3. In this chapter, the steps of the hybrid algorithm is first outlined. Due to a lack of work in the MMO of HCCOP, the performance of the proposed algorithm is validated by utilizing it for the MMO of benchmark JSSPs. The performance of the proposed algorithm is compared to the performance of other algorithms used for MMO of JSSP. Next, the proposed algorithm and the algorithms used for the MMO of JSSP are modified and used for the MMO of benchmark permutation flow-shop scheduling problem (PFSSP) and their performance is compared. Finally, the algorithms are modified and used for the MMO of AJSSP.

6.1 The clustering-optimization algorithm for MMO

The clustering-optimization approach proposed in this dissertation assumes that a problem may have multiple global optima, however, these global optima will have different features. By combining a clustering algorithm with any metaheuristic algorithm, the hybrid algorithm will not only be able to converge to the multiple global optima but also the unique features that differentiate these global optima. Classical *k-means* clustering algorithm is known as an easy algorithm for clustering and is a widely applied unsupervised learning approach. Therefore, the hybrid MMO algorithm is created by combining *k-means clustering* algorithm with GA for the MMO of JSSP and PFSSP and the algorithm proposed

in CHAPTER 3 for the MMO of AJSSP. Therefore, the proposed algorithm can be utilized for the MMO of both COPs as well as HCCOPs.

In the proposed MMO algorithm, *k-means clustering* algorithm if first used to cluster solutions depending on their features. Next, adapted crossover and mutation operators of GA are utilized within each cluster to generate population for the next generation. Due to the encoding method used, different solutions can lead to the same makespan value in JSSP and AJSSP. Therefore, only optimal solutions that provide a *unique schedule*, are considered. To study the exploration capabilities of the proposed MMO algorithm all optimal solutions that provide a *unique schedule* for JSSP are saved in the optimal solution set. The optimal solution set is then compared to the different number of optimal solutions provided in other literature. The optimal solution set is different than the final solution set i.e. $final\ population \subseteq optimal\ solution\ set$. The basic procedure of proposed MMO algorithm is described as follows and a detailed flow chart is presented in Figure 24. In Figure 24, *S* refers to the solution set in which all unique optimal solution encountered by the algorithm are stored and B is the fitness value of the optimal solution set.

Step 1: Define the fitness function of the specific case study. In this study, the maximum completion time (makespan) was used as the fitness value;

Step 2: Define the parameters for *k-means clustering* algorithm (k value) and GA (population size, crossover rate, mutation rate, generation limit) and initialize the best solution set;

Step 3: Generate the initial population;

Step 4: Evaluate the current population, calculate the feature matrix of every solution and update the best solution set;

Step 5: Using *k-means clustering* algorithm, divide the current population into k clusters, based on the features calculated in Step 4;

Step 6: Within each cluster, select elite individuals that will be a part of the next generations population.

Step 7: Within each cluster, select individuals for crossover using roulette wheel selection to produce offspring for the next generation.

Step 8: Randomly select individuals (based on mutation probability) and mutate them.

Step 7: Repeat Steps 4-8 until the generation limit is reached. Output the final population and the best solution set;

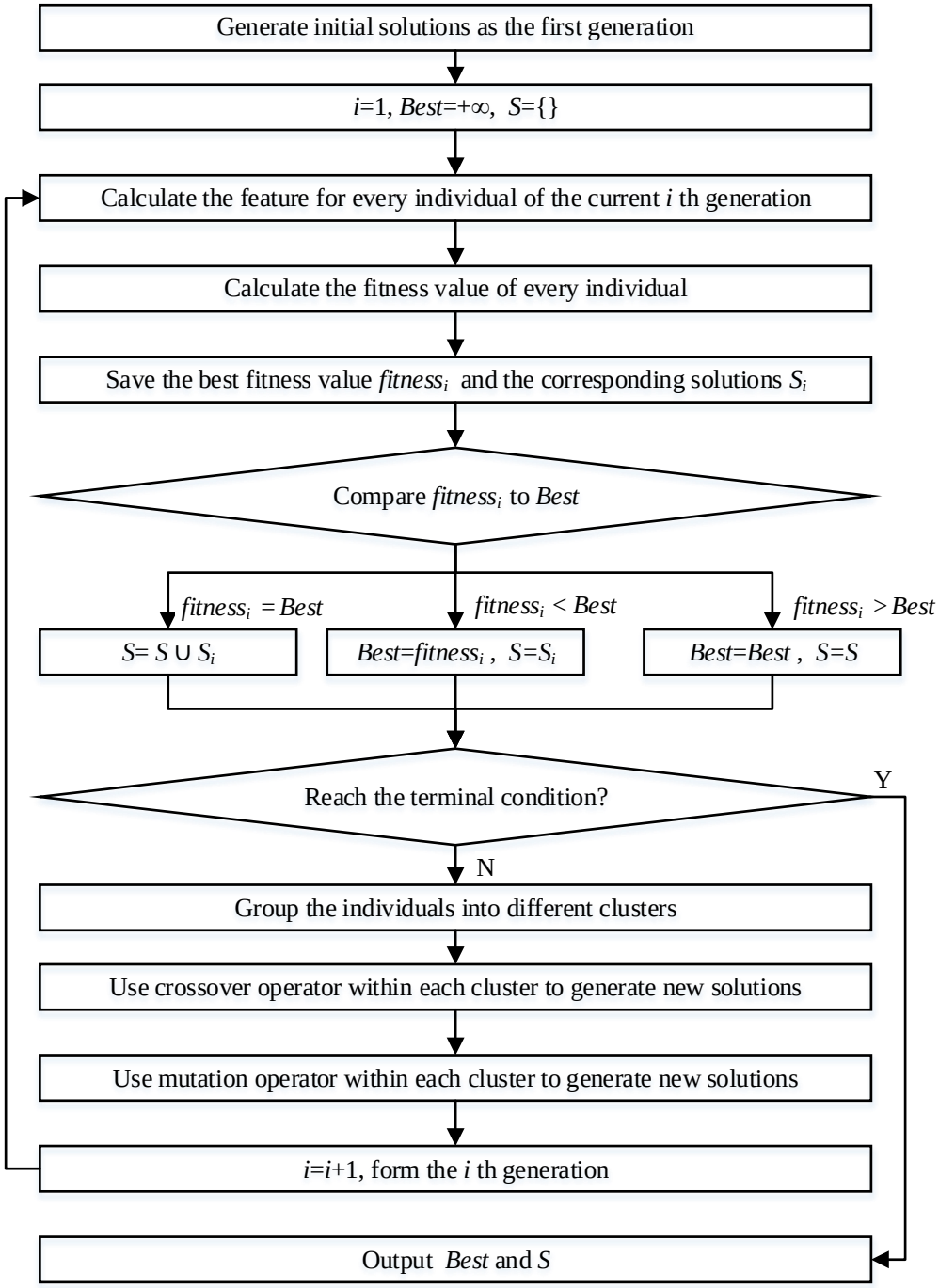

Figure 24. Flow chart of the proposed MMO algorithm

6.2 MMO of benchmark JSSP

Job-shop scheduling problem (JSSP) is a classical problem in operation research in which *n* ideal jobs, J_1, J_2, …, J_n, of varying processing times are assigned to *m* resources (usually machines). The objective is to allocate the available resources to the jobs to maximize (or minimize) performance indicators such as makespan, tardiness etc. Like any scheduling problem, there are constraints and assumptions associated with JSSPs and in this paper these constraints and assumptions are:

1. On the shop floor, all machines are independent from each other.
2. The operation of all the jobs have deterministic processing times.
3. No pre-emption of job operation is permitted
4. The set-up times are not considered in these problems.
5. No machine down-time or maintenance time is considered.
6. The machine used for each operation of a job is known and fixed.
7. A job cannot revisit a machine group for more than one operation.
8. Operation constraints must always be followed.

Known as NP-hard problem, JSSPs has aroused the attention and efforts of many researchers, who have used various techniques to achieve the desired goal. Qing-dao-er-ji and Wang [88] proposed a new hybrid genetic algorithm (GA) for the optimization of JSSP. Their proposed method had modified selection, crossover, and mutation operator that were

designed to increase the diversity of the population. In addition, a local search operator was also developed to improve the quality of the solution, a common drawback of GA. Jorapur *et al.* [89] developed a new technique to enhance the quality of the initial population used in GA. Their proposed method could obtain better percentage deviation for benchmark problems compared to other methods used in literature. Chang *et al*. [90] combined Taguchi method and GA to solve flexible job shop scheduling problem (FJSSP), a special kind of JSSP with parallel machines. To avoid infeasible solutions, the authors developed a novel technique to encode only feasible solutions in the initial chromosomes and then embedded Taguchi method in the mating operator to increase the effectiveness of GA. When applied to benchmark problems, the authors proposed method obtained better performance than other commonly used methods. Bagheri *et al*. [91] used an artificial immune algorithm (AIA) for minimization of makespan of FJSSP. The authors tested several strategies for generating initial population, selection of new individuals, and mutation to converge to the optimal solution. Tsai *et al*. [92] developed a modified Taguchi-immune algorithm (MTIA) that was successfully used for global numerical and JSSP optimization. The authors combined Taguchi-method with AIA to have better local and global convergence and the application of the proposed algorithm to benchmark problems showed the robustness of the proposed algorithm.

In contrast to traditional optimization of JSSP, very little work has been done in the MMO of JSSP. Luh and Chueh [93] proposed a multimodal immune algorithm (MMIA) for MMO of JSSP. The authors tested the ability their proposed algorithm to find multiple global optima on benchmark JSSP. Their experimental results showed that not only were they able to find the global optima for 89.5% of the problems tested but they were also able

to find multiple schedules with the optimal makespan. Perez et al. modified several multimodal GA's for the MMO of benchmark JSSPs. In their study, Perez *et al.* [24] tested sharing fitness GAs, clearing based GAs and species competition based GAs on eight benchmark JSSPs. Their experimental results showed that sharing fitness GAs provided the best results when finding a global optimum was the only objective while clearing based GAs and species competition based GAs helped obtain the most number of global optima.

Though the algorithms developed for the MMO of JSSP are able to provide multiple optimal schedules, they have a few drawbacks. These algorithms either provide very few optimal solutions or are unable to consistently converge to the global optima. Therefore, the MMO algorithm developed in Section 6.1 is utilized to consistently converge to the global optima while providing a sufficient amount of optimal solutions.

6.2.1 Modelling of solution

As mentioned earlier, GA is used for the optimization procedure. The optimization objective is to find the sequence of operations on each machine that would lead to a minimization of the makespan. An important aspect of GA is to determine how to code the solutions. Over the years, many different methods have been used to encode the solutions of JSSP such as the direct method [94], the binary method [95], the circular method [96], and the permutation with repetition [97] method. In the proposed MMO algorithm, permutation with repetition is used to represent the solutions as it avoids infeasible solutions entirely, an important requirement for JSSPs. The length of the chromosome (solution) can be determined by the size of the problem. For example, as shown in Figure 25, for a 3x3 problem (3 jobs with 3 operations each) the chromosome length will be 9.

Figure 25 also demonstrates how the solution is represented using permutation with repetition encoding.

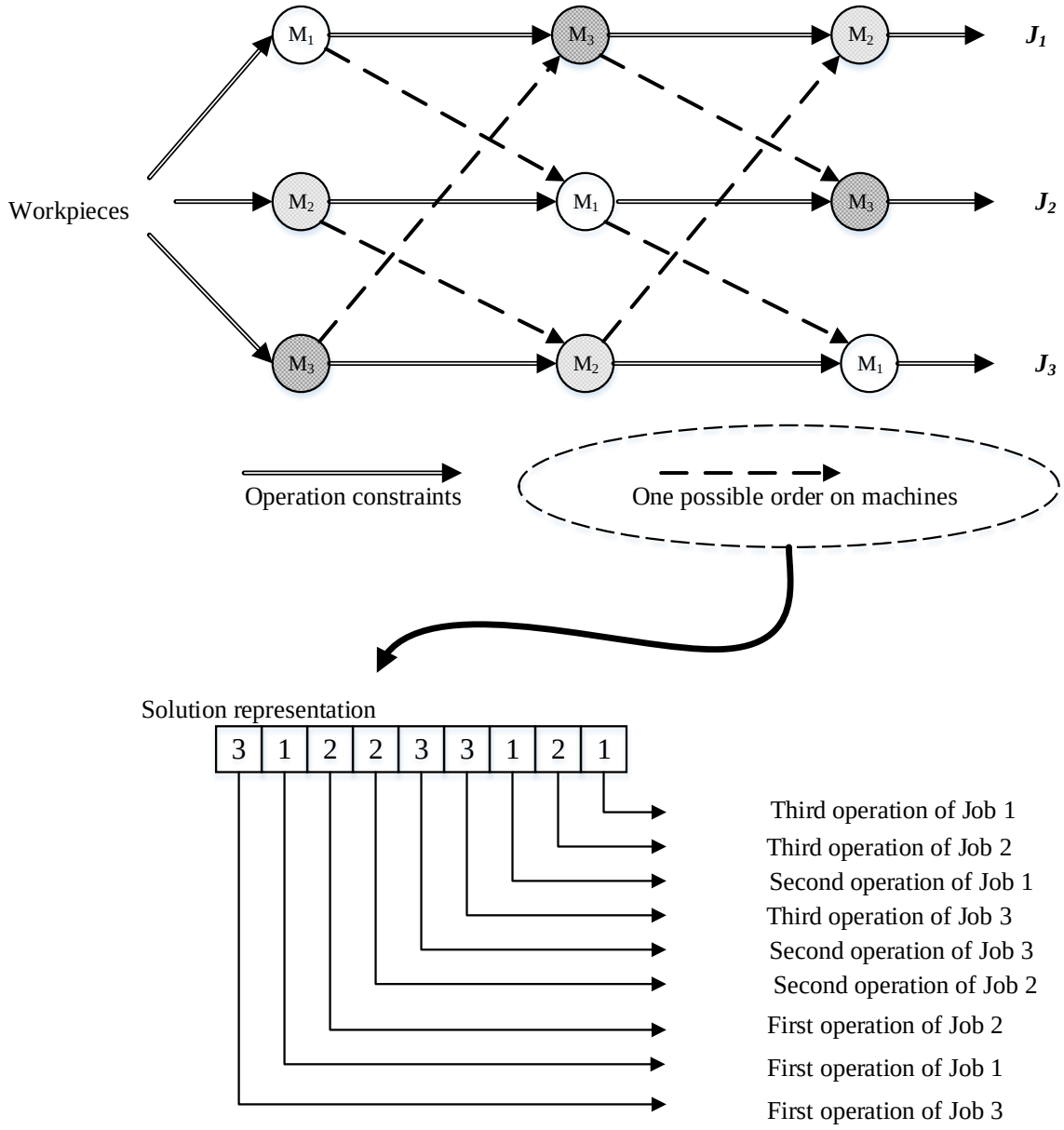

Figure 25. An example of permutation coding in a 3x3 JSSP

6.2.2 Definition of feature

During the optimization process, the solutions have to be clustered together into different clusters using *k-means* clustering algorithm. In order to cluster the solutions together, some features of the solutions have to be defined that indicate the uniqueness of each solution. In this study, the feature of a solution is defined as a matrix of sequence difference between every two machines in the workshop. Figure 26 shows an example of how the feature for a solution is calculated. The number of columns equals to the number of machines *m* and the number of rows equals to the number of combination C_m^2, therefore, the size of the feature matrix is $C_m^2 \times 1$.

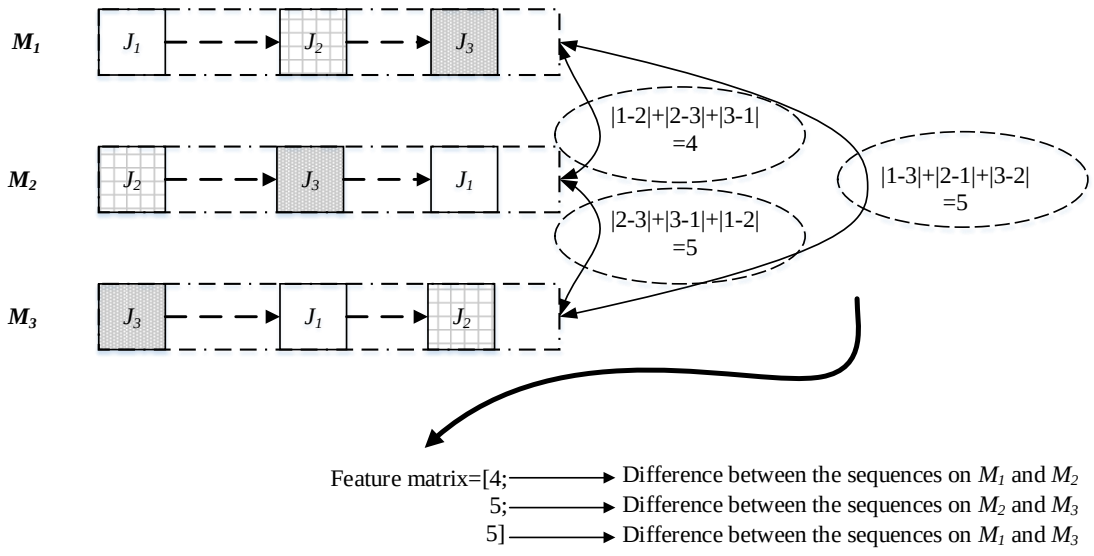

Figure 26. Figure demonstrating an example of the feature matrix for 3x3 JSSP

6.2.3 Adaption of genetic operators

In the optimization process, genetic operators are utilized to produce the population belonging to the next generation. As mentioned earlier, to only deal with feasible solutions,

permutation with repetition encoding method is used. In order to continue to deal with only feasible solutions, appropriate crossover and mutation operator must be utilized as dealing with only feasible solutions can greatly reduce the computation cost. In the crossover operator, the half the number of jobs of the problem under consideration are randomly selected. Next, the corresponding genes are switched in both the parents. An example of the crossover process is shown in Figure 27. In the example, jobs 2 and 3 are randomly chosen and the corresponding genes ([3,2,3,2,3,2] in Parent A and [2,2,2,3,3,3] in Parent B) are switched. The crossover operator is limited to solutions belonging to the same cluster in order to preserve some features of the cluster. Similarly, in the mutation operator, two jobs are randomly chosen and their corresponding location within the solution is switched. An example of the mutation operator for a problem with three jobs and each job having three operations is shown in Figure 28. In the mutation operator, two jobs are first selected (in Figure 28, jobs 2 and 3 are selected) and the location of the genes belonging to the selected jobs is switched. In the example shown in Figure 28, the genes for jobs 2 and 3 appear is the order [3,2,3,2,3,2]. After mutation, the order of the genes is changed to [2,3,2,3,2,3] and the new parent is given as [1,2,3,2,3,2,1,3,1]. As it can be seen from Figure 27 and Figure 28, using the crossover and mutation operator only creates feasible solution.

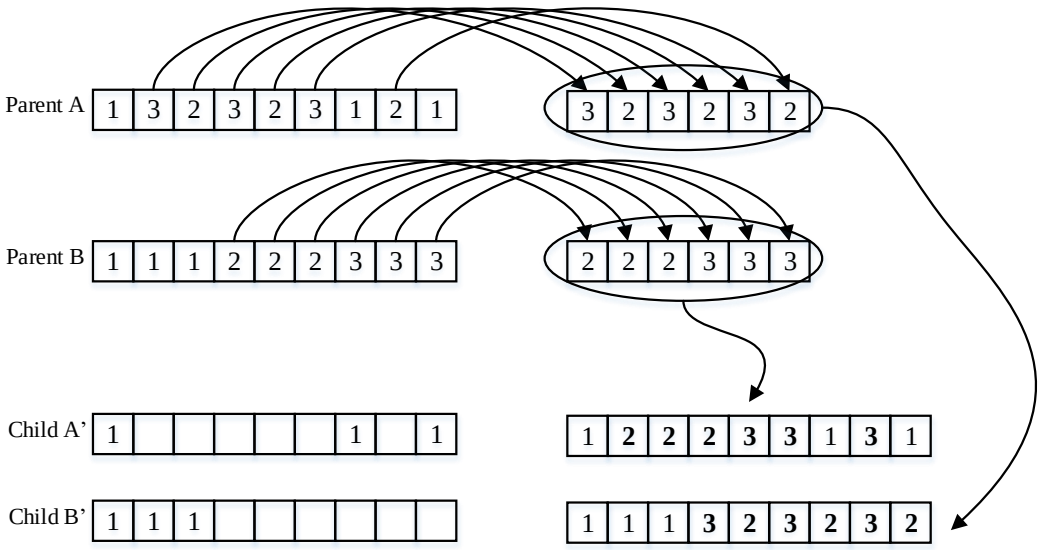

Figure 27. Crossover operator adapted to permutation coding in a 3x3 JSSP

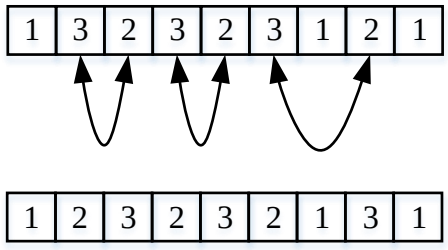

Figure 28. Mutation operator adapted to permutation coding in a 3x3 JSSP

6.2.4 Experimental Study

The effectiveness of the proposed MMO algorithm is measured using two indicators: 1. the ability to converge to the global optima and 2. the ability to find multiple optimal solutions. Furthermore, the performance of the algorithm is also measured against the performance of the algorithms used by Perez et al. and Luh and Chueh for MMO of JSSP.

6.2.4.1 Case Study 1

As mentioned earlier, Luh and Chueh proposed a MMIA for MMO of benchmark JSSP. The authors first tested the ability of their algorithm to converge to the global optima by employing it to 19 benchmark problems (*MT10, MT20, LA01-LA11, LA17, LA19, LA21, LA31*). The 19 benchmark problems were chosen as they had varying dimensions i.e. 6 to 30 jobs and 5 to 10 machines. The parameters of MMIA were adopted based on the size of the problem. For example, since it is relatively easy to find the global optima of *MT06* compared to that of *LA31,* the algorithm was only run for 100 iterations with a population size of 36 when used to optimize *MT06* while the iteration limit and population size were 1000 and 600 when it was used to optimize *LA31*. To ensure fair comparison, the parameters used for the proposed algorithm were the same as those used by Luh and Chueh. These parameters are listed in Table 19 while the best solution found using MMIA and the proposed algorithm as well as the best-known solution for the problems under consideration are listed in Table 20.

For the proposed algorithm, an additional parameter needs to be specified i.e. the value of *k*. A *k* value of 3 was determined to provide the best results, both in terms of converging to and finding multiple optimal solutions, after multiple simulation runs.

Table 19. Parameters used for the proposed algorithm

Instance size (job x machine)	6x6	10x5	15x5	20x5	10x10	15x10	30x10
Generation Limit	100			500			1000
Population Size	Job x Machine			2 x (job x machine)			
Chromosome Length	36	50	75	100	100	150	200

Table 20. Optimal makespan found using MMIA and the proposed algorithm

Instance	Size	Best-known solution	MMIA	Proposed MMO Algorithm
MT06	6x6	55	55	55
MT10	10x10	930	955	**937**
LA01	10x5	666	666	666
LA02	10x5	655	655	655
LA03	10x5	597	597	597
LA04	10x5	590	590	590
LA05	10x5	593	593	593
LA10	15x5	958	958	958
LA14	20x5	1292	1292	1292
LA17	10x10	784	784	784
LA31	30x10	1784	1784	1784

For problems *MT06, LA01-LA05, LA10, LA14, LA17,* and *LA31* both, the proposed MMO algorithm and MMIA are able to converge to the best-known solutions. For *MT10,* neither the proposed MMO algorithm nor MMIA are able to converge to the global optimum however, the proposed MMO algorithm is able to converge to a better solution (937) than MMIA (955). Once the convergence ability of MMIA was verified, Luh and Chueh utilized MMIA for MMO of five benchmark JSSPs. The authors chose the five benchmark JSSPs (10x5, 15x5, 20x5, 10x10, and 30x10) with different dimensions and degree of difficulty to validate the convergence capabilities as well as the ability of the algorithm to find multiple solutions. The proposed MMO algorithm is also utilized to try and find multiple global optima of these five problems. The results obtained using MMIA and the proposed MMO algorithm are given in Table 21.

Table 21. Multiple solutions found using MMIA and the proposed algorithm

Instance	Number of optimal solutions found using MMIA	Number of optimal solutions found using the proposed MMO algorithm
MT06	5	**22**
LA05	5	**41**
LA10	9	**37**
LA14	9	**144**
LA17	3	**26**
LA31	2	**31**

According to Table 4, MMIA was able to find 5, 5, 9, 9, 3, and 2 optimal solutions for *MT06, LA05, LA10, LA14, LA17,* and *LA31* respectively while the proposed MMO algorithm found 22, 41, 37, 144, 26, and 31 solutions for those problems. The results in Table 4 show that even though both the algorithms managed to find multiple optimal solutions for the five problems under consideration, the proposed MMO algorithm is able to find a much larger amount of optimal solutions compared to MMIA. The combined results of Tables 3 and 4 that the proposed MMO algorithm is able to outperform MMIA in terms of converging to and find multiple global optima.

6.2.4.2 Case Study 2

In the second case study, the performance of the proposed MMO algorithm is compared to the performance of the algorithms used by Perez *et al.* Perez *et al.* used adaptive niche hierarchical genetic algorithm (ANHGA), niche identification techniques with sharing fitness (NIT), classical clearing method (CM), restricted competition selection method (RCS), restricted competition selection with pattern search method (RCS_PSM),

species conserving genetic algorithm (SCGA), and quick hierarchical fair competition (QHFC) for MMO of benchmark JSSPs. The authors evaluated the algorithms based on their ability to converge to the global optimum as well as finding multiple global optima. The converging capabilities of the algorithms was determined by their application to *LA01-LA05, MT06, MT10, and MT20.* In the first part of their experiments, the authors utilized each algorithm with the aim of finding the global optima of each problem. To test the stability of each algorithm, 30 repetitions were performed and the average optimal value was reported. Since the authors found that different algorithms were suitable for finding the optimal solution and MMO, the results obtained using those algorithms are given in this section. The parameters used by Perez *et al.* for the simulations were as follows:

- Population size: 100
- Stop conditions: Number of evaluations, 100,000 for *MT06, LA01, and LA05* 300,000 for *LA02, LA03, and LA04*, and 500, 000 for *MT10 and MT20.*
- Runs: 30 repetitions for each problem.

In contrast to the settings used by Perez et al., the 10,000 iterations were used for MT06, LA01-05 and 20,000 for MT10 and MT20. Table 22 shows average optimal solution obtained by Perez et al. as well as those obtained using the proposed algorithm.

Table 22. Optimal makespan found by Perez *et al.* and using the proposed algorithm

Instance	Best-known solution	ANHGA	RCS	RCS_PSM	SCGA	Proposed MMO Algorithm
MT06	**55**	**55.00**	**55.00**	**55.00**	**55.00**	**55.00**
MT10	**930**	979.00	988.17	1006.07	989.70	**958.90**

Table 22 continued.

Instance	Best-known solution	ANHGA	RCS	RCS_PSM	SCGA	Proposed MMO Algorithm
MT20	**1165**	1213.03	1235.47	1284.80	1237.97	**1177.50**
LA01	**666**	**666.00**	**666.00**	**666.00**	**666.00**	**666.00**
LA02	**655**	660.63	663.03	667.50	678.13	**656.80**
LA03	**597**	608.97	609.23	609.90	609.60	**597.00**
LA04	**590**	595.97	606.73	602.80	600.63	**590.90**
LA05	**593**	**593.00**	**593.00**	**593.00**	**593.00**	**593.00**

It can be seen from that all the algorithms are able to find the global optima of *MT06, LA01,* and *LA05* as it is relatively easy to find their global optima. The best average makespan found by Perez *et al.* for *MT10, MT20,* and *LA01-LA04* were 969.00, 1213,03, 660.63, 608.97, and 595.97 respectively (obtained using ANHGA). The proposed MMO algorithm gave an average optimal makespan of 958.90, 1177.60, 656.80, 597.00, and 590.09 for those problems. These results indicate that the proposed MMO algorithm is able to find better optimal solutions while requiring much less iterations.

Next, Perez *et al.* utilized the algorithms to find multiple global optima for the benchmark problems listed in Table 22. Similar to the previous simulations, the authors utilized their algorithm on each problem 30 times and noted the average number of different global optima obtained. If, for example, out of 30 runs, the global optima were only found in 27 runs, then the average was calculated using those 27 runs. It should be noted that the maximum number of different optima that could be found was the lower number between the population size and total number of global optima for the particular problem. For example, the population size used in these simulations was 100, however

there are only 55 different known global optima for *MT06.* Therefore, 55 is the maximum number of different global optima that can be obtained for *MT06.* The average number of global optima obtained by Perez *et al.* and using the proposed MMO algorithm are shown in Table 23.

Table 23. Multiple global optima found by Perez *et al.* and using the proposed MMO algorithm. (*) indicates that there were no global optima obtained and (**) indicates that the global optima was only obtained in a handful of simulations

Instance	ANHGA	RCS	RCS_PSM	SCGA	Proposed MMO Algorithm
MT06	11.70	**49.67**	48.63	48.07	6.4
MT10	**2.22****	1**	*	*	*
MT20	1**	1**	*	*	*
LA01	30.54	**100.00**	99.90	93.37	12.1
LA02	16.86	**100.00**	99.95	99.00	6.20
LA03	**12.40**	*	*	95**	8.90
LA04	**19.50**	50**	50**	94**	7.60
LA05	39.73	**100.00**	**100.00**	98.93	25.50

For *MT06,* RCS used by Perez *et al.* obtained an average of 49.67 global optima while the proposed algorithm only obtained 6.4. For *MT10,* ANHGA was able to obtain 2.22 global optima but very few runs were actually able to converge to the global optima (as indicated by **). The proposed MMO algorithm, on the other hand, is unable to converge to the global optima in any of runs. Similar results are obtained for *MT20.* For *LA01* and *LA02,* RCS was able to obtain 100 different global optima while the proposed MMO algorithm is only able to obtain 12.1 and 6.2 global optima respectively. Though SCGA was able to obtain 95 global optima for *LA03,* very few runs were successful. However, ANHGA was more successful as it was able to obtain 12.40 global optima on

average (for more simulations) while the proposed MMO algorithm obtained 8.9. For *LA04*, ANHGA obtained an average of 19.50 global optima while the proposed algorithm obtained 7.60. Lastly, for *LA05*, both RCS and RCS_PSM obtained 100.00 global optima while the proposed algorithm only obtained 25.50 It is evident from the results in Table 6 that the proposed MMO algorithm is unable to obtain as many global optima as the algorithms used by Perez *et al.*. It should be noted that the iteration limit was set to 10,000 for *MT06, LA01-LA05* and 20,000 for *MT10* and *MT20* when being solved using the proposed algorithm. These numbers were much higher, 100,000 – 300,000, when solved by Perez *et al.*. The low number of iterations could be a cause of the fewer number of global optima obtained by the proposed algorithm. Thought the number of global optima were lower, the proposed MMO algorithm has equal or better convergence than any of the algorithms used by Perez *et al.* while requiring fewer iterations

Lastly, as stated earlier, the total number of global optima encountered by the proposed MMO algorithm during its iterations are also stored. These numbers are given in Table 24 and along with the known number of different global optima.

Table 24. Known global optima to date and different global optima encountered by the proposed algorithm

	MT06	***MT10***	***MT20***	***LA01***	***LA02***	***LA03***	***LA04***	***LA05***
Different global optima known to date	55	4	1	26813	5321	706	143	48471
Global optima found by proposed MMO algorithm	22	-	-	**64921**	2388	194	**1225**	**335606**

Since the proposed MMO algorithm is unable to converge to the global optima for MT10 and MT20, the corresponding values are 0 in Table 24. For LA01, LA04, and LA05 the proposed MMO algorithm is able to find 38108, 1082, and 287,135 new solutions. Though it only found some of the known solutions for the remaining problems, the number of global optima encountered for problems LA01-LA05 are much larger than the population size (100). These results show that the algorithm is able to search the solution space effectively and find a large amount of global optima and the number of encountered by the algorithm is not just limited to the population size.

6.3 MMO of benchmark PFSSP

Permutation flow-shop scheduling problem (PFSSP), a variation of the classical flow-shop scheduling problem (FSSP), is a problem in operation research in which n ideal jobs, J_1, J_2, …, J_n, of varying processing times are assigned to m resources (usually machines). The objective is to allocate the available resources to the jobs to maximize (or minimize) performance indicators such as makespan, tardiness etc. In classical FSSP, for n jobs on m machines, there are $(n!)^m$ different alternatives for sequencing jobs on machines, while in permutation problems, the search space is reduced to $n!$. PFSSP has the same constraints as JSSP but with an additional constraint that all the jobs must enter the machines in the same order.

Since PFFSP are commonly encountered in production environment, researchers have come up with many techniques to optimize their key performance indicators (KPIs). Liu and Liu [98] used a hybrid discrete artificial bee colony algorithm for minizine the makespan in PFSSP. They utilized Greedy Randomized Adaptive Search Procedure

(GRASP) to create the initial population and operators such as insert, swap, path relinking to generate new solutions. The authors also improved the local search by further optimizing the best solution. When applied to PFFP, their algorithm had superior performance compared to other algorithms such as particle sawm optimization (PSO) algorithm, hybrid genetic algorithm (HGA), etc. Govindan *et al.* [99] combined decision tree (DT) and scatter search (SS) algorithms to solve PFSSP. DT was used initially to convert the problem into a tree structure using the entropy function followed by SS to do an extensive investigation of the solution space. Simulation results and statistical test comparisons showed the advantage to the authors proposed algorithm. Ancău [100] proposed two algorithms for solving PFSSP 1. A constructive greedy heuristic (CG) and 2. Stochastic greedy heuristic (SG). The CG was based on a greedy selection while the SG was a modified version of CG with iterative stochastic start. The authors validated the proposed algorithms by using them to solve benchmark problems and the results showed that the algorithm was able to come within 6% of the best-known solution. Zobolas *et al.* [101] created a hybrid algorithm by combining greedy heuristic, GA, and variable neighborhood search (VNS) algorithm. The hybrid algorithm was able to take advantage of both GA and VNS and obtain the best-known makespan for the benchmark problems in short computational time.

In contrast to the work available in the optimization of PFSSP, no work has been done in the MMO of PFSSP. Therefore, the proposed MMO algorithm proposed is utilized for the MMO of PFSSP. However, a few changes are made to the operators of GA as the constraints in PFSSP are different than those in JSSP.

6.3.1 Modelling of PFSSP

Similar to the MMO of JSSP, the objective in MMO of PFSSP is to find the sequence of operations that would lead to a minimization of makespan. As the jobs must be processed in the same order on each machine in PFSSP, a solution is created using permutation encoding for a single machine. However, unlike JSSP, the length of the chromosome would only be three for a 3x3 problem.

In the modified crossover operator, half the number of jobs of the problem under consideration are randomly select. Next, the corresponding genes are switched in both the parents. An example of the crossover process is shown in Figure 29 for a FSSP with nine jobs. In the example, jobs 2, 3, 4, 5, 6, and 9 are randomly chosen and the corresponding genes ([3,2,4,9,5,6] in Parent A and [2,3,6,9,5,4] in Parent B) are switched. The crossover operator is again limited to solutions belonging to the same cluster in order to preserve some features of the cluster. Similarly, in the mutation operator, half the jobs are randomly chosen and their corresponding location within the solution is switched. An example of the mutation operator is shown in Figure 30. As it can be seen from Figure 29and Figure 30, using the crossover and mutation operator only creates feasible solution.

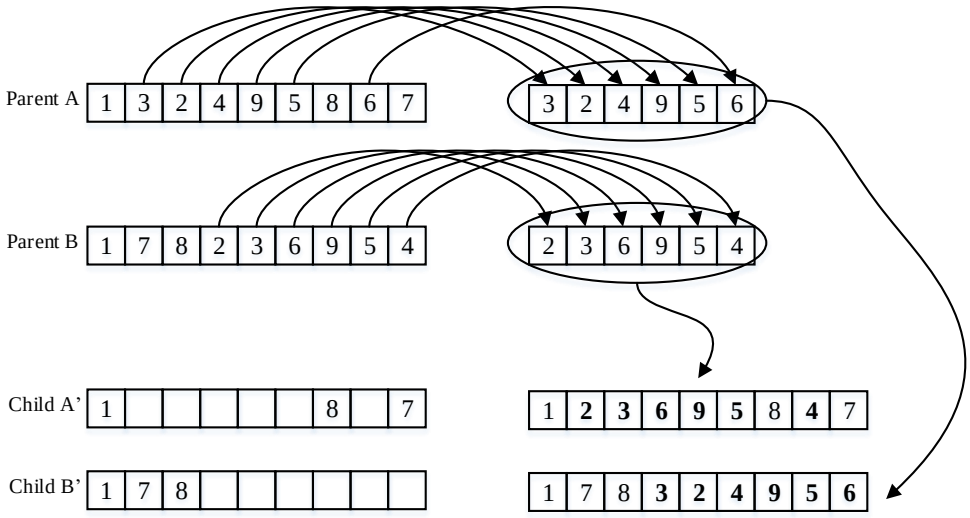

Figure 29. Example demonstrating the modified crossover operator used in the MMO of PFSSP

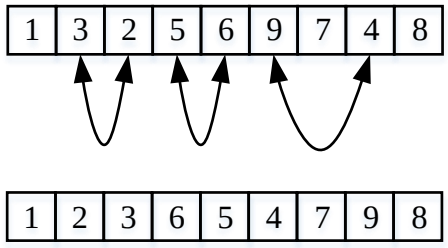

Figure 30. Example demonstrating the modified mutation operator used in the MMO of PFSSP

The feature used to cluster each solution was defined as the order of operations of jobs of that particular solution. For example, the feature of Parent A shown in Figure 29 would be [1, 3, 2, 4, 9, 5, 8, 6, 7] while that of Parent B would be [1, 7, 8, 2, 3, 6, 9, 5, 4].

6.3.2 Experimental Study

Next, to measure the performance of the proposed MMO algorithm, it is used for the MMO of multiple PFSSP. The sharing fitness algorithm and clearing method (CM) used by Perez *et al.* for the MMO of JSSP are also utilized for the MMO of PFSSP. The performance of the three algorithms are then compared using 18 benchmark problems from various datasets. The first nine problems are proposed by Carlier (1978), the next eight are proposed by Reeves (1995) and the last one is proposed by Heller (1960). To better assess the algorithms, 10 independent simulations are performed for each test problems. The performance of the algorithms used is then measured based on four indicators: 1. The best solution found in the 10 simulations, 2. The mean value and the standard deviation of the optimal solution found for the 10 simulations, 3. The number of times the algorithms converged to the best optimal solution and 4. The average number of best optimal solutions found for the 10 simulations. For indicator number 4, only simulations in which the algorithm converged to the best solution from (1) are taken into consideration i.e. if the algorithm only converged to the best solution in 4 out of the 10 simulations, then the average number of optimal solutions found is calculated using those 4 simulations. The parameters used for the three algorithms are determined after numerous simulations and are given in Table 25.

Table 25. Parameters used for the three algorithms

	Generations	Population Size	Algorithm Specific Parameters	
Proposed Algorithm	1000	200	*Number of clusters = 3*	
Sharing Fitness			$\sigma_{share} = 0$	$\beta = 1$
Clearing Method			$\sigma_{share} = 10$	*Maximum niche size = 1*

6.3.2.1 The best solution found

As mentioned above, 10 independent simulations are ran for each test problem using the algorithms. The first method used to compare the performance of these algorithms is by observing the best solution obtained in 10 simulations. The results obtained by these algorithms are also compared to the best-known solution in available in literature. These results are shown in Table 26.

Table 26. Best optimal solution obtained by the different algorithm

Problem	Best-known solution in literature	Proposed MMO Algorithm	Sharing fitness	CM
Rec-01	1247	1249	1249	1379
Rec-03	1109	1111	1114	1187
Rec-05	1242	1245	1245	1307
Rec-07	1566	**1566**	1589	1733
Rec-09	1537	**1537**	1600	1706
Rec-11	1431	**1431**	1475	1614
Rec-13	1930	1954	1980	2145
Rec-15	1950	1962	2007	2156
Rec-17	1902	1919	1989	2178
Car-01	7038	**7038**	**7038**	**7038**
Car-02	7166	**7166**	**7166**	7565

Table 26 continued.

Problem	Best-known solution in literature	Proposed MMO Algorithm	Sharing fitness	CM
Car-03	7312	**7312**	**7312**	7399
Car-04	8803	**8803**	**8003**	8410
Car-05	7720	**7720**	**7720**	**7720**
Car-06	8505	**8505**	**8505**	**8505**
Car-07	6590	**6590**	**6590**	**6590**
Car-08	8366	**8366**	**8366**	**8366**
Hel-02	137	**137**	139	154

According to Table 26, of the algorithms used for the MMO of PFSSP, both the proposed MMO algorithm and sharing fitness algorithm are able to find the best-known solutions for all the Carlier benchmark problems while CM is able to find the best-known solution for Car-01, 05, 06, 07, and 08. For the Reeves problems, the proposed MMO algorithm has the best performance as it is able to converge to the best-known solution for Rec-07 and 09 and is within 1% for rest of the problems. Sharing fitness and CM are unable to converge to any of the best-known solutions for the Reeves problems and are within 4% and 15% of the best-known solutions. Similarly, only the proposed MMO algorithm converges to the best-known solution of the Heller problem while sharing fitness and CM are unable to. These results indicate that the proposed MMO algorithm has the best performance in terms of finding the best optimal solution.

6.3.2.2 Mean value and standard deviation of the optimal solution

Next, to evaluate the stability of the algorithms, the mean value and the standard deviation of the optimal solution found in the 10 simulations are calculated. Since these

are 10 independent simulations, the starting points are randomized in each simulation. These results are shown in Table 27.

Table 27. Mean value and standard deviation of the optimal solution found after 10 independent simulations using the different algorithms

Problem Number	Mean value and standard deviation of the optimal solution found using the proposed MMO algorithm	Mean value and standard deviation of the optimal solution found using the sharing fitness	Mean value and standard deviation of the optimal solution found using CM
Rec-01	**1249.00 ± 0.00**	1302.30 ± 30.65	1429.40 ± 29.14
Rec-03	**1111.00 ± 0.00**	1136.4 ± 15.61	1246.50 ± 29.01
Rec-05	**1245.00 ± 0.00**	1266.20 ± 39.63	1360.90 ± 25.59
Rec-07	**1579.00 ± 8.07**	1622.20 ± 25.21	1771.20 ± 24.48
Rec-09	**1567.70 ± 16.54**	1625.40 ± 12.29	1767.60 ± 30.15
Rec-11	**1440.80 ± 11.78**	1505.2 ± 21.82	1669.00 ± 29.45
Rec-13	**1956.90 ± 2.28**	2029.50 ± 30.87	2218.70 ± 38.50
Rec-15	**1974.30 ± 6.31**	2026.80 ± 12.92	2189.30 ± 20.34
Rec-17	**1944.30 ± 11.62**	2024.60 ± 20.67	2219.13 ± 21.71
Car-01	**7038.00 ± 0.00**	**7038.00 ± 0.00**	7468.50 ± 276.02
Car-02	**7166.00 ± 0.00**	**7166.00 ± 0.00**	7862.30 ± 163.70
Car-03	**7312.00 ± 0.00**	7351.40 ± 43.06	7832.50 ± 266.94
Car-04	**8803.00 ± 0.00**	8813.40 ± 32.89	8468.30 ± 70.84
Car-05	**7720.00 ± 0.00**	7759.70 ± 38.77	7731.20 ± 16.48
Car-06	**8505.00 ± 0.00**	8550.50 ± 31.40	8511.50 ± 20.55
Car-07	**6590.00 ± 0.00**	6600.60 ± 22.37	**6590.00 ± 0.00**
Car-08	**8366.00 ± 0.00**	**8366.00 ± 0.00**	**8366.00 ± 0.00**
Hel-02	**137.00 ± 0.00**	143.00 ± 1.89	155.90 ± 1.45

For the Reeves problems, the proposed MMO algorithm has the best performance as its average optimal value is much closer to the best-known solution than that of sharing fitness and CM. The standard deviation varies between 0.00 and 16.54 for the proposed MMO algorithm while it varied between 15.61 and 39.63 for sharing fitness and 20.34 and 30.15 for CM. Similar results are obtained for the Carlier problems. The proposed MMO

algorithm is always able to converge to the best-known solution and therefore, has a standard deviation of 0.00. Sharing fitness has a 100% convergence rate to the best-known solutions of problems 1, 2, and 8 while CM has a 100% convergence rate to the best-known solutions of problems 7 and 8. For the rest of the problems, the standard deviation varies between 22.37 and 43.06 for sharing fitness and 20.55 and 276.02 for CM. Lastly, for Heller, the proposed MMO algorithm again has a 100% convergence rate to the best-known solution while sharing fitness and CM are unable to do so. These results show the robustness of the proposed MMO algorithm i.e. its ability to converge to the best optimal solution consistently.

6.3.2.3 Number of times the algorithm converged to the optimal solution

To ensure that the algorithms can find multiple solutions consistently, the number of simulations in which the algorithms converge to the best-known solution (N_C) is also recorded. Since none of the algorithms converge to majority of the best-known solution for the Reeves problems, the best optimal solution found is used instead. For example, the best-known solution for Rec-01 is 1247 while the best optimal solution found by the algorithms is 1249, therefore, 1249 is used to calculate N_C. The results for N_C are given in Table 28.

Table 28. Number of simulations in which the algorithms are able to converge to the best optimal solution (N_C)

Problem Number	N_C obtained using the proposed MMO algorithm	N_C obtained using sharing fitness	N_C obtained using CM
Rec-01	**2**	1	0
Rec-03	**8**	0	0
Rec-05	**1**	0	0

Table 28 continued.

Problem Number	NC obtained using the proposed MMO algorithm	NC obtained using sharing fitness	NC obtained using CM
Rec-07	**10**	0	0
Rec-09	**10**	0	0
Rec-11	**4**	0	0
Rec-13	**1**	0	0
Rec-15	**2**	0	0
Rec-17	**1**	0	0
Car-01	**10**	**10**	2
Car-02	**10**	**10**	0
Car-03	**10**	4	0
Car-04	**10**	9	0
Car-05	**10**	3	6
Car-06	**10**	3	9
Car-07	**10**	8	**10**
Car-08	**10**	**10**	**10**
Hel-02	**10**	0	0

Since CM has the worst optimal solution of the three algorithms for the Reeves problems its N_C*'s* are 0. Similarly, sharing fitness is unable to find the best optimal solution for all but problem 1 and therefore, its N_C*'s* for those problems is 0. The proposed MMO algorithm has the best performance of the three algorithms compared. It has a N_C of 10 for problems 7 and 9, 1 for problems 5, 13, and 17, 2 for problems 1 and 15, 4 for problem 11, and 8 for problem 3.

As the proposed MMO algorithm has a 100% convergence rate to the best-known solution for all the Carlier problems, its N_C*'s* for them is 10. Sharing fitness also has a N_C of 10 for problems, 1, 2, and 8 and has a N_C of 4, 9, 3, 3, and 8 for the remaining problems. CM has the worst performance as it is unable to converge to the best-known solution for

problems 2, 3, and 4. It has an N_C of 2, 6, and 9 for problems 1, 5, and 6 respectively and had an N_C of 10 for problems 7, and 8. Lastly, for the Heller problem, the proposed MMO algorithm has an N_C of 10 while the other two algorithms have an N_C of 0. These results further solidify the argument that the proposed MMO algorithm has the best performance of the three algorithms as it is able to converge to the best optimal solution consistently.

6.3.2.4 Average number of best optimal solutions found (N_O)

Lastly, the average number of best optimal solutions found for each test problem using the different algorithms is compared. For this index, only simulations that have a N_C of greater than 0 are used. For example, since the proposed MMO algorithm has a N_C of two for Rec-01, N_O is calculated using the results of those two simulations. N_O for the different test problems using the different algorithms are given in Table 29.

Table 29. Average number of best optimal solutions found ((NO) using the different algorithms

Problem Number	N_O obtained using the proposed MMO algorithm	N_O obtained using sharing fitness	N_O obtained using CM
Rec-01	6.60	**18.00**	0.00
Rec-03	**6.80**	0.00	0.00
Rec-05	**4.20**	0.00	0.00
Rec-07	**3.00**	0.00	0.00
Rec-09	**4.00**	0.00	0.00
Rec-11	**3.00**	0.00	0.00
Rec-13	**3.00**	0.00	0.00
Rec-15	**2.00**	0.00	0.00
Rec-17	**3.00**	0.00	0.00
Car-01	37.90	40.00	**58.50**
Car-02	**30.40**	9.89	0.00
Car-03	**6.20**	1.25	0.00

Table 29 continued.

Problem Number	N_O obtained using the proposed MMO algorithm	N_O obtained using sharing fitness	NO obtained using CM
Car-04	**13.5**	8.67	0.00
Car-05	**1.90**	1.00	1.00
Car-06	**1.00**	1.00	1.00
Car-07	**1.00**	1.00	1.00
Car-08	**1.00**	1.00	1.00
Hel-02	**6.30**	0	0.00

Like the previous three indicators, the proposed MMO algorithm has significantly better performance than sharing fitness and CM. Though sharing fitness is able to find 18 solutions for Rec-01, it is unable to find multiple optimal solutions for rest of the Reeves problems. The proposed MMO algorithm is able to find 2 to 6.8 optimal solutions for all the Reeves problems. CM is unable to find multiple best optimal solutions for any of the Reeves problems as it never converged to the best optimal solution.

Though both CM and sharing fitness are able to find more best optimal solutions compared to the proposed algorithm for Car-01, their performance is worse for problems 2,3,4, and 5. The proposed MMO algorithm found 30.40, 6.20, 13.50, and 1.90 different best-known solutions on average for those problems while sharing fitness only obtained 9.89, 1.25, 8.67 and 1.00 and CM obtained 0.00 for all of them. All the algorithms are only able to find 1.00 different optimal solutions for problems 6-8 indicating that the global optimal solution is a unique one. Lastly, for the Heller problem, the proposed MMO algorithm again has the best performance as it is able to obtain 6.30 different best-known solutions on average while both sharing fitness and CM obtained 0.00. These results

indicate that the proposed algorithm is able to consistently converge to the best optimal solution for all the test problems while finding multiple best optimal solutions.

6.3.2.5 Hybrid Algorithm

An attempt is also made to combine the proposed MMO algorithm with sharing fitness to take advantage of both the algorithms i.e. the ability of the proposed algorithm to consistently converge to the best optimal solution and the ability of sharing fitness to find multiple best optimal solutions. The above task is accomplished by calculating the distance used in the sharing fitness method using the feature matrix of the proposed MMO algorithm. This distance is calculated using Equation (53).

$$d(i, j) = \left\| F_i - F_j \right\| \tag{53}$$

Where, F_i is the feature matrix of Individual i and F_j is the feature matrix of Individual j. For the hybrid algorithm, a σ_{share} value of 10 is used. The results obtained using the new hybrid algorithm are compared to the best results obtained from the proposed MMO algorithm, sharing fitness and CM. These results are given

Table 30. Comparison of the best optimal solution found using the new hybrid algorithm and the previous three algorithms

Problem Number	Best optimal solution found using the new hybrid algorithm	Best optimal solution found using previous three algorithms
Rec-01	1280	**1249**
Rec-03	1124	**1111**
Rec-05	1249	**1245**
Rec-07	1584	**1566**
Rec-09	1590	**1537**

Table 30 continued.

Problem Number	Best optimal solution found using the new hybrid algorithm	Best optimal solution found using previous three algorithms
Rec-11	1457	**1431**
Rec-13	2003	**1954**
Rec-15	2012	**1962**
Rec-17	2008	**1919**
Car-01	**7038**	**7038**
Car-02	**7166**	**7166**
Car-03	**7312**	**7312**
Car-04	**8803**	**8803**
Car-05	**7720**	**7720**
Car-06	**8505**	**8505**
Car-07	**6590**	**6590**
Car-08	**8366**	**8366**
Hel-02	139	**137**

Table 31. Comparison of the mean value and standard deviation of the optimal solutions found using the new hybrid algorithm and from the previous three algorithms

Problem Number	Best mean value and standard deviation of the optimal solution found using the new hybrid algorithm	Best mean value and standard deviation of the optimal solution found using the previous three algorithms
Rec-01	1310.60 $\pm$ 17.66	**1249.00 $\pm$ 0.00**
Rec-03	1139.80 $\pm$ 13.10	**1111.00 $\pm$ 0.00**
Rec-05	1269.60 $\pm$ 11.28	**1245.00 $\pm$ 0.00**
Rec-07	1623.90 $\pm$ 29.44	**1579.00 $\pm$ 8.07**
Rec-09	1617.70 $\pm$ 18.60	**1567.70 $\pm$ 16.54**
Rec-11	1457.20 $\pm$ 17.76	**1440.80 $\pm$ 11.78**
Rec-13	2042.20 $\pm$ 21.90	**1956.90 $\pm$ 2.28**
Rec-15	2047.60 $\pm$ 30.31	**1974.30 $\pm$ 6.31**
Rec-17	2028.00 $\pm$ 13.72	**1944.30 $\pm$ 11.62**
Car-01	7038.00 $\pm$ 0.00	**7038.00 $\pm$ 0.00**
Car-02	7187.00 $\pm$ 66.41	**7166.00 $\pm$ 0.00**
Car-03	7382.10 $\pm$ 27.89	**7312.00 $\pm$ 0.00**

Table 31 **continued.**

Problem Number	Best mean value and standard deviation of the optimal solution found using the new hybrid algorithm	Best mean value and standard deviation of the optimal solution found using the previous three algorithms
Car-04	8803.00 ± 0.00	**8803.00 ± 0.00**
Car-05	7745.70 ± 18.37	**7720.00 ± 0.00**
Car-06	8561.20 ± 71.27	**8505.00 ± 0.00**
Car-07	6605.90 ± 25.60	**6590.00 ± 0.00**
Car-08	8366.00 ± 0.00	**8366.00 ± 0.00**
Hel-02	143.10 ± 1.66	137.00 ± 0.00

Table 32. Comparison of the N_c found using the new hybrid algorithm and the previous three algorithms

Problem Number	N_C obtained using the new hybrid algorithm	Best N_C obtained using the previous three algorithms
Rec-01	0	**2**
Rec-03	0	**8**
Rec-05	0	**1**
Rec-07	0	**10**
Rec-09	0	**10**
Rec-11	0	**10**
Rec-13	0	**1**
Rec-15	0	**2**
Rec-17	0	**1**
Car-01	**10**	**10**
Car-02	9	**10**
Car-03	1	**10**
Car-04	**10**	**10**
Car-05	2	**10**
Car-06	4	**10**
Car-07	7	**10**
Car-08	**10**	**10**

Hel-02	0	**10**

Table 33. Comparison of the N_O using the new hybrid algorithm and the previous three algorithms

Problem Number	N_O obtained using the new hybrid algorithm	Best N_O obtained using the previous three algorithms
Rec-01	0.00	**18.00**
Rec-03	0.00	**4.20**
Rec-05	0.00	**3.00**
Rec-07	0.00	**4.00**
Rec-09	0.00	**3.00**
Rec-11	0.00	**3.00**
Rec-13	0.00	**2.00**
Rec-15	0.00	**3.00**
Rec-17	0.00	**6.80**
Car-01	33.80	**58.50**
Car-02	17.11	**30.40**
Car-03	1.00	**6.20**
Car-04	12.20	**13.5**
Car-05	1.00	**1.90**
Car-06	1.00	**1.00**
Car-07	1.00	**1.00**
Car-08	1.00	**1.00**
Hel-02	0.00	**6.30**

As it can be seen from the results above, the new hybrid algorithm is able to achieve the same best optimal makespan for the Carlier problems but not for the Reeves and Heller problems. In terms of the mean value and standard deviation of the optimal solution found, the new hybrid algorithm has similar performance to the best results obtained using the previous three algorithms for Carlier problems 1, 4, and 8 while it has higher mean value and standard deviation for rest of the benchmark problems. When comparing N_c, the hybrid

algorithm is able to converge to the best optimal solutions in all 10 simulations for Carlier problems 1, 4, and 8 but is unable to achieve similar results for the remaining problems. In terms of N_O, the new hybrid algorithm is unable to obtain more optimal solutions for any of the benchmark problems. These results indicate that the new hybrid algorithm did not have a superior performance compared to the best performance obtained using the previous three algorithms.

The results from the MMO of the benchmark PFSSP show that the proposed MMO algorithm has the best performance in terms of the best optimal solution found, the mean value and standard deviation of the optimal solution found, number of times the algorithm converges to the best optimal solution, and different amount of best optimal solutions found.

6.4 MMO of AJSSP

Since the proposed MMO algorithm showed promising results when utilized for the MMO of JSSP and PFSSP, it is utilized for the MMO of AJSSP. The feature matrix used in the proposed algorithm has to be redefined as AJSSP has different features than JSSP and PFSSP i.e. not even job will visit all the machines on the shop floor. Therefore, for AJSSP, the feature matrix of a solution is calculated by taking the difference between two machines with the same amount of jobs.

The AJSSP problems proposed by Dileeplal *et al.* are used to test the algorithms. To better assess the algorithms, 10 independent simulations are performed for each test problems. The performance of the algorithms used is measured based on four indicators: 1. The best solution found in the 10 simulations, 2. The mean value and the standard

deviation of the optimal solution found for the 10 simulations, 3. The number of times the algorithms converged to the best optimal solution and 4. The average number of best optimal solutions found for the 10 simulations. For indicator number 4, only simulations in which the algorithm converged to the best solution from (1) are taken into consideration i.e. if the algorithm only converged to the best solution in 4 out of the 10 simulations, then the average number of optimal solutions found was calculated using those 4 simulations. The parameters used for the three algorithms were determined after numerous simulations and are given in Table 34.

Table 34. Parameters used for the three algorithms

	Generations	**Population Size**	**Algorithm Specific Parameters**	
Proposed Algorithm Sharing Fitness Clearing Method	1000	200	*Number of clusters = 3*	
			$\sigma_{share} = 0$	$\beta = 1$
			$\sigma_{share} = 50$	*Maximum niche size = 1*

6.4.1.1 The best solution found.

As mentioned above, 10 independent simulations are ran for each test problem using the algorithms. The first method used to compare the performance of these algorithms is observing the best solution obtained by each algorithm during those 10 simulations. The results obtained by these algorithms are also compared to the results obtained by Dileeplal *et al.* These results are shown in Table 34.

Table 35. Best optimal solution obtained by the different algorithms

Problem Number	Best optimal solution found by the Dileeplal *et al.*	Best optimal solution found by the proposed MMO algorithm	Best optimal solution found using sharing fitness	Best optimal solution found using CM
1	78	**77**	78	81
2	**88**	**88**	**88**	93
3	100	**98**	99	101
4	**110**	**110**	**110**	113
5	**135**	**135**	**135**	136
6	**125**	**125**	**125**	128
7	118	**110**	118	117
8	**129**	**129**	134	136

According to Table 35, of the algorithms used for the MMO, the proposed MMO algorithm is able to find the best solutions for problems 1 (77), 3 (98), 7 (110), and 8 (129). The solutions for problems 1, 3, and 7 represent an improvement of 1.28%, 2.00%, and 6.78% respectively compared to the solutions found by Dileeplal *et al.*. For problems 2, 4, 5, and 6 both, the proposed MMO algorithm and sharing fitness are able to find the same best optimal solutions of 88, 110, 135, and 125 respectively. For all the test problems, CM is unable to obtain the best optimal solution, however, it has slightly better performance than sharing fitness for problem 7 (117 vs 118). These results indicate that the proposed MMO algorithm has the best performance in terms of finding the best optimal solution.

6.4.1.2 Mean value and standard deviation of the optimal solution.

Next, to evaluate the stability of the algorithms, the mean value and the standard deviation of the optimal solution found in the 10 simulations is calculated. Since these are

10 independent simulations, the starting points are randomized in each simulation. Since no information was provided by Dileeplal *et al.* about the mean value and standard deviation of the best optimal solution found, the results were only compared for the proposed algorithm, sharing fitness, and CM. These results are shown in Table 36.

Table 36. Mean value and standard deviation of the optimal solution found after 10 independent simulations using the different algorithms

Problem Number	Mean value and standard deviation of the optimal solution found using the proposed MMO algorithm	Mean value and standard deviation of the optimal solution found using the sharing fitness	Mean value and standard deviation of the optimal solution found using CM
1	**77.80 ± 0.42**	81.40 ± 1.51	84.40 ± 1.90
2	**88.40 ± 0.97**	90.30 ± 1.70	99.50 ± 2.84
3	**99.50 ± 0.71**	101.60 ± 0.97	101.78 ± 0.44
4	**110.00 ± 0.00**	111.50 ± 1.96	114.80 ± 1.62
5	**135.00 ± 0.00**	**135.00 ± 0.00**	142.10 ± 3.28
6	**125.00 ± 0.00**	125.80 ± 1.40	133.80 ± 4.92
7	**113.90 ± 1.60**	118.90 ± 0.74	120.70 ± 1.95
8	**130.70 ± 0.95**	136.40 ± 1.17	138.90 ± 2.13

For problems 1, 2, 3, 4, 6, 7, and 8 the proposed MMO algorithm is able to obtain better a mean value and standard deviation of the optimal solution compared to sharing fitness and CM. For problems 4, 5, and 6 the proposed MMO algorithm has a mean value of 110.00, 135.00, and 125.00 and standard deviation of 0.00 indicating that the algorithm is able to converge to the best optimal solution in all of the simulations. Sharing fitness is also able to converge to the best optimal solution in all of the simulations for problem 5 and has better performance compared to CM. These results show the robustness of the

proposed MMO algorithm i.e. its ability to converge to the best optimal solution consistently.

6.4.1.3 Number of times the algorithm converged to the optimal solution.

To ensure that the algorithms can find multiple solutions consistently, the number of simulations in which the algorithms converge to the best optimal solution (N_C) is also recorded. Here, only simulations in which the algorithms converge to the best optimal solution is recorded i.e. if the best optimal solution is 77 while the algorithm only converged to this solution 4 out of the 10 solutions, then N_C is 4. The results for N_C are given in Table 4.

Table 37. Number of simulations in which the algorithms were able to converge to the best optimal solution (N_C)

Problem Number	N_C obtained using the proposed MMO algorithm	N_C obtained using sharing fitness	N_C obtained using CM
1	**2**	0	0
2	**8**	0	0
3	**1**	0	0
4	**10**	6	0
5	**10**	**10**	0
6	**10**	3	0
7	**1**	0	0
8	**2**	0	0

Since CM is unable to find the best optimal solution (Table 35), its N_C for all problems is 0. Similarly, sharing fitness is to find the best optimal solution for problems 1, 2, 3, 7, and 8, therefore, its N_C for those problems is 0. Sharing fitness, however, is able to converge to the best optimal solution for problems 4, 5, and 6 and its N_C for those problems

is 6, 10 and 3 respectively. The proposed MMO algorithm has the best performance of the three algorithms compared. It had a N_C of 10 for problems 4, 5, and 6, 8 for problem 2, 2 for problems 1, and 8, and 1 for problem 3 and 7. These results further solidify the argument that the proposed MMO algorithm is able to converge to the best optimal solution consistently.

6.4.1.4 Average number of best optimal solutions found (N_O)

Lastly, the average number of best optimal solutions found for each test problem using the different algorithms is compared. Similar to the section above, only simulations that have a N_C of greater than 0 are used. For example, since sharing fitness has a N_C of 6 for problem 4, N_O is calculated using the results of those 6 simulations. N_O for the different test problems using the different algorithms are given in Table 5.

Table 38. Average number of best optimal solutions found *(N_O)* using the different algorithms

Problem Number	N_O obtained using the proposed MMO algorithm	N_O obtained using sharing fitness	N_O obtained using CM
1	**6.50**	0.00	0.00
2	**5.75**	0.00	0.00
3	**2.00**	0.00	0.00
4	16.40	**200.00**	0.00
5	33.40	**200.00**	0.00
6	14.60	**199.71**	0.00
7	**2.00**	0.00	0.00
8	**8.00**	0.00	0.00

Unlike the previous three indicators, sharing fitness has significantly better performance than the proposed algorithm and CM. For the problems where sharing fitness

has an N_C of greater than 0 (4, 5, and 6), it is able to find almost 200 different best optimal solutions. In contrast, the proposed MMO algorithm is only able to find 16.40, 33.40, and 14.60 different optimal solutions for those problems. However, unlike sharing fitness, since the proposed MMO algorithm is able to converge to the best optimal solutions for the problems 1, 2, 3, 7, and 8, it has a N_O of 6.50, 5.75, 2.00, 2.00, and 8.00 for those problems respectively. As CM is unable to converge to the best optimal solution for any of the problems, its N_O is 0 for all of them. These results indicate that though the proposed MMO algorithm is unable to find as many best optimal solutions as sharing fitness, it is able to consistently converge to the best optimal solution for all the test problems while finding multiple best optimal solutions.

6.4.1.5 Hybrid algorithm

As sharing fitness is able to find much more best optimal solutions than the proposed algorithm, it is again combined with the proposed algorithm in order to create a hybrid algorithm. Similar to the hybrid algorithm used in Section 6.2 and Section 6.3, the hybrid algorithm in this section utilizes the feature matrix used in the proposed algorithm for sharing fitness. For the hybrid algorithm, the value of σ_{share} is changed to 500. The performance of the hybrid algorithm is then compared to the performances of best solutions found using the three methods. The results are given in the Tables below.

Table 39. Comparison of the best optimal solution found using the new hybrid algorithm and the previous three algorithms

Problem Number	Best optimal solution found using the hybrid algorithm	Best optimal solution found using previous three algorithms
1	78	**77**
2	**88**	**88**
3	**98**	**98**
4	**110**	**110**
5	**135**	**135**
6	**125**	**125**
7	114	**110**
8	131	**129**

Table 40. Comparison of the mean value and standard deviation of the optimal solutions found using the new hybrid algorithm and from the previous three algorithms

Problem Number	Best mean value and standard deviation of the optimal solution found using the hybrid algorithm	Best mean value and standard deviation of the optimal solution found using the previous three algorithms
1	78.00 ± 0.02	$\mathbf{77.80 \pm 0.42}$
2	88.70 ± 0.95	$\mathbf{88.40 \pm 0.97}$
3	$\mathbf{99.20 \pm 0.63}$	99.50 ± 0.71
4	$\mathbf{110.00 \pm 0.00}$	$\mathbf{110.00 \pm 0.00}$
5	$\mathbf{135.00 \pm 0.00}$	$\mathbf{135.00 \pm 0.00}$
6	$\mathbf{125.00 \pm 0.00}$	$\mathbf{125.00 \pm 0.00}$
7	117.20 ± 1.32	$\mathbf{113.90 \pm 1.60}$
8	132.70 ± 1.42	$\mathbf{130.70 \pm 0.95}$

Table 41. Comparison of the N_c found using the hybrid algorithm and the previous three algorithms

Problem Number	N_C obtained using the hybrid algorithm	Best N_C obtained using the previous three algorithms
1	0	**2**
2	6	**8**
3	**1**	**1**
4	**10**	**10**
5	**10**	**10**
6	**10**	**10**
7	0	**1**
8	0	**2**

Table 42. Comparison of the N_O using the hybrid algorithm and the previous three algorithms

Problem Number	N_O obtained using the hybrid algorithm	Best N_O obtained using the previous three algorithms
1	**3.17**	**6.50**
2	**3.00**	**5.75**
3	**3.00**	2.00
4	3.50	**200.00**
5	5.70	**200.00**
6	3.10	**199.71**
7	0.00	**2.00**
8	0.00	**8.00**

As it can be seen from Table 39, the hybrid algorithm is able to achieve the same best optimal makespan for problems 2-6 but is unable to do so for problems 1, 7, and 8. In terms of the mean value and standard deviation of the optimal solution found, the hybrid algorithm has better performance on problem 3 and is able to obtain the same results as the

previous three algorithms for problems 4, 5, and 6. For the remaining problems, the previous three algorithms provide has slightly better solutions than the hybrid algorithm. When comparing N_c, the new hybrid algorithm is able to converge to the best optimal solutions in all 10 simulations for problems 4, 5, and 6 and also has same performance as the previous three algorithms for problem 3. For problems 1, 2, 7, and 8, however, the previous three algorithms have slightly better performance. In terms of N_O, the hybrid algorithm is able to obtain more optimal solutions on average for problem 3, but for the remaining problems, either the proposed algorithm or sharing fitness algorithm is able to provide more optimal solutions. These results indicate that the new hybrid algorithm does not have a superior performance compared to sharing fitness or the proposed algorithm.

The results from the application of the proposed MMO algorithm to various JSSP, PFSSP, and AJSSP demonstrate its capability to converge to best optimal solution consistently while finding multiple best optimal solutions. The hybrid algorithm created using the proposed MMO algorithm and sharing fitness algorithm is unable to match the performance of sharing fitness or the proposed MMO algorithm. It should also be noted that the proposed MMO algorithm is able to obtain similar or better results than sharing fitness and CM for JSSP, PFSSP, and AJSSP while requiring minor changes to the algorithm and its parameters. In contrast, sharing fitness and CM have parameters that are highly problem dependent and require significant fine tuning.

6.5 Conclusion

In this chapter, an algorithm is proposed for the MMO optimization of HCCOP, specifically AJSSP. This is accomplished by utilizing *k-means* clustering algorithm to

cluster the solutions of every generation, based on their features, and then using the algorithm proposed in CHAPTER 3 for optimization. Due to a lack of benchmark AJSSP, the performance of the proposed MMO algorithm is verified by its application for the MMO of JSSP and PFSSP. For the MMO of JSSP and PFSSP, GA is used for optimization instead of the algorithm proposed in CHAPTER 3. In Section 6.1, the proposed MMO optimization algorithm is introduced and its steps are outlined.

In Section 6.2, the proposed MMO algorithm is utilized for the MMO of benchmark JSSP. First, it is demonstrated how GA is used to model a solution for JSSP followed by the feature matrix used to cluster solutions. Next, the crossover and mutation operators used to generate feasible solutions are developed. The performance of the proposed MMO algorithm is compared to the performance of the algorithms proposed by Luh and Chueh and Perez *et al.* The results of the case studies show that the proposed MMO is able to find better and more optimal solutions than the algorithm proposed by Luh and Chueh and better, but fewer, optimal solutions than the algorithm used by Perez *et al.* The proposed MMO algorithm is also able to find new global optima for some of the benchmark problems that have not been reported in previous literature.

In Section 6.3, the proposed MMO algorithm is utilized for the MMO of benchmark PFSSP. First, it is demonstrated how GA is used to model a solution for PFSSP followed by the feature matrix used to cluster solutions. Next, the crossover and mutation operators used to generate feasible solutions are developed. Sharing fitness and CM algorithm are also modified and used for the MMO of PFSSP and the performance of the three algorithms is compared. The results show that the proposed MMO algorithm is able to obtain better and more best optimal solutions than sharing fitness and CM. A hybrid algorithm is next

developed by combining the proposed MMO algorithm and sharing fitness. However, the hybrid algorithm is unable to obtain better results than sharing fitness or the proposed MMO algorithm.

In Section 6.4, the proposed MMO algorithm is utilized for the MMO of benchmark AJSSP. The feature matrix is again redefined as the constraints for AJSSP are different than those of JSSP and PFSSP. The proposed MMO algorithm along with sharing fitness and CM are used for the MMO of the AJSSP problems proposed by Dileeplal *et al.* Similar to the results obtained in Section 6.3, the proposed MMO has better performance than CM and sharing fitness. The hybrid algorithm was also unable to get better results than those obtained using the three algorithms. The results obtained in this Chapter show that the proposed MMO algorithm can be used for the MMO of various problems without having to significantly change its parameters unlike CM and sharing fitness.

CHAPTER 7. CONCLUSIONS AND FUTURE WORK

7.1 Summary

This dissertation presents a method to identify and optimize HCCOPs as well as an algorithm for the MMO of various scheduling and HCCOPs. The algorithm utilized to optimize HCCOPs is based on evolutionary computation while the MMO algorithm is created by combining the proposed optimization algorithm and a clustering technique. In CHAPTER 3, the abstract definition and common principles to identify HCCOPs are first developed. Next, to optimize the HCCOPs, a new versatile algorithm, based on evolutionary computation, is proposed. The algorithm is developed in a way such that only the feasible solution space is searched during the iterations of the algorithm in order to reduce its complexity. This is accomplished by utilizing the proposed initial solution generator, level-barrier based crossover, and level-barrier based mutation operator. The proposed operators ensure that the multi-level HCCs are always satisfied, thereby creating only feasible solutions.

In CHAPTER 4, the proposed algorithm is utilized for the optimization of AJSSP. The optimization of AJSSP is first established as a HCCOP based on the definitions developed in CHAPTER 3. Next, it is demonstrated how the solution can be modeled using proposed operators of the algorithm. The performance of the proposed algorithm is verified by using it to optimize various AJSSP and comparing its performance to the performance of other algorithms used for the optimization of AJSSP. The proposed algorithm is able to obtain better or equivalent results than other algorithms for 75% of the problem. Convergence plots generated by varying the parameters of the algorithm are used to

demonstrate the robustness of the proposed algorithm. A complexity analysis of the proposed algorithm is also performed.

In CHAPTER 4, the proposed algorithm is utilized for the simultaneous optimization of NN structure and weights. First, it is demonstrated how the above problem can be classified as HCCOP using the definitions established in CHAPTER 3 followed by the procedure used to create feasible initial solutions as well as feasible solutions during rest of the iterations using the proposed algorithm. The performance of the proposed algorithm is verified by using it create a NN prediction model for four different case studies and comparing the prediction accuracy of the model created using the proposed algorithm to those created by other algorithms. The results show that prediction model created using the proposed algorithm had better prediction accuracy than prediction models created using classical NN, ANFIS, regression analysis, and gaussian-process regression.

In CHAPTER 6, an algorithm is proposed for the MMO optimization of HCCOPs. This is accomplished by combining *k-means* clustering algorithm with the algorithm proposed for the optimization of HCCOPs. Due to a lack of benchmark problems for AJSSP, the performance of the proposed MMO algorithm is validated by using it for the MMO of JSSP and PFSSP. For the MMO of JSSP and PFSSP, the proposed algorithm for the optimization of HCCOPs is replaced with GA. The performance of the proposed MMO algorithm is compared to other algorithms used for MMO and the results show that the proposed MMO algorithm is able to obtain better but fewer optimal solutions for JSSP and better and more optimal solutions for PFSSP. When utilized for the MMO of AJSSP, the proposed algorithm has much better performance than other algorithms. The MMO results

show that the proposed algorithm can be applied to different problems without significantly changing the parameters of the algorithm.

7.2 Conclusions

The research presented in this dissertation was driven by the need for a methodology to identify HCCOPs and a versatile algorithm for their optimization. The research has shown that there are some common principles amongst different HCCOPs that can be used to classify them as such. The proposed algorithm was based on evolutionary computation and was developed in a way such that only the feasible solution space was searched which reduces its computation complexity. When utilized for the optimization of various AJSSP, the proposed algorithm had better results than SGA and generally better solutions than the algorithm proposed by Dileeplal *et al*. The NN prediction model built using the proposed algorithm also had better prediction accuracy than other methods used in published literature. The successful application of the proposed algorithm to different HCCOPs without requiring immense changes demonstrate its versatility.

MMO optimization of the HCCOPs is also an important topic as a single solution may satisfy the initial objective function, it might not be applicable in real-life. The MMO algorithm was developed by combining a clustering algorithm with the proposed algorithm for the MMO of HCCOPs and GA for the optimization of JSSP and PFSSP. The results from the MMO of JSSP show that the MMO algorithm was able to obtain better results while requiring fewer iterations than other algorithms used. Similar results were obtained when the proposed MMO algorithm was utilized for the MMO of PFSSP and AJSSP. The

proposed MMO algorithm was able to obtain these results without requiring significant changes being made to its parameters which demonstrates its ease of use.

7.3 Contributions

The intellectual contributions of the research presented are as follows:

- Proposed a methodology to identify and model HCCOPs.
- Developed an algorithm to optimize various HCCOPs without requiring significant changes.
- Validated the performance of the methodology and the proposed algorithm by using it to optimize various AJSSP and for the simultaneous optimization of NN structure and weights and comparing the results obtained with published results.
- Developed an algorithm for the MMO of JSSP, PFSSP, and HCCOPs (AJSSP specifically). The proposed MMO is capable of performing MMO for the specified problems without requiring significant changes.
- Validated the performance of the proposed MMO algorithm with other researchers' published results.

The work presented in this dissertation has also been successfully applied for several industrial applications. It has been used to create a NN prediction model for advanced manufacturing processes such as electrochemical micro-machining (EMM) and freeform electrochemical machining (ECM). The proposed MMO algorithm has been utilized to obtain multiple optimal process parameter combinations for the process of EMM and ECM. The methodology developed in CHAPTER 3 has been utilized to classify scheduling

problem encountered in a manufacturing industry as well as the production of stator core as HCCOPs. The algorithm has also been utilized successfully for optimization the process parameters in the production of stator core.

7.4 Limitations

This research explores the identification and optimization of HCCOPs yet it suffers several limitations.

1. **Optimization of AJSSP:** As demonstrated in CHAPTER 4, new solutions are generated using the level-barrier based crossover and level-barrier based mutation operators. However, if the problem under consideration only has one level of dependent variables, then the level-barrier based crossover operator cannot be utilized. A problem with only one level of dependent variables will only have a single 1^{st} level gene. As the level-barrier based crossover operator changes the position of entire genes in two different solutions, switching the position of the 0^{th} or 1^{st} level gene in a problem with one level of dependent variables would result in the duplication of the parents. Similarly, unless each level has multiple genes of the exact same length, the level-barrier based mutation operator cannot be utilized. This severely impacts the performance of the algorithm as mutation is a crucial for searching the unexplored solution space. Also, if the problem only has a single level of dependent variables, then the level-barrier based mutation operator degrades into classical mutation operator.
2. **Simultaneous Optimization of NN structure and weights:** As mentioned in CHAPTER 5, in order to simultaneously optimize NN structure and weight values using the proposed algorithm, a fully connected NN structure has to be assumed. If the

NN is not fully connected, then the connection information needs to be encoded as well. However, this can lead to extremely large chromosomes, thereby severally reducing the computation speed of the algorithm. Furthermore, if the connection information is encoded in the solution, infeasible solutions would also be created using the proposed operators. Infeasible solutions would be created as a result of the level-barrier based crossover and level-barrier based mutation operator i.e. if there is a change in the number of connection between different layers then the corresponding size of the weight and bias matrix would be too large or too small.

3. **MMO of HCCOP:** In order to cluster solutions using *k-means clustering* algorithm, some features of the solutions had to be defined and these features were extracted from the encoded solution. However, if the features are unable to be encoded into the solution, then the proposed algorithm cannot be utilized for MMO. Furthermore, all the limitations that apply to the optimization of AJSSP also apply to the MMO of AJSSP.

7.5 Future Work

The proposed methodology provides a solid foundation for identifying HCCOPs. Though it was used to successfully identify two different problems as HCCOPs, the developed definition and common principles need to be further refined by utilizing them to identify additional manufacturing as well as theoretical problems as HCCOPs.

The proposed algorithm is only used to optimize the makespan of AJSSP. However, in scheduling problems, many other KPIs such as tardiness, maximum lateness etc. are also important. Also, the minimization of one KPI can lead to a worse value of another KPI (for example, minimization of makespan can lead to a worse tardiness value) [58]. Therefore,

the performance of the proposed algorithm with other KPIs as the objective needs to be studied. Multi-objective optimization of AJSSP is another area in which the proposed algorithm should be used. Furthermore, real-life AJSSP are more complex due to the presence of constraints. Since the algorithm was utilized successfully for test problems, it should be utilized to optimize real-life AJSSP and improvements should be made based on the results obtained.

When utilized for the simultaneous optimization of NN weight and structure, the training of the weights had to be supplemented with the use of LM algorithm due to issues with local convergence. Further investigation needs to be performed in order to improve the local convergence capabilities of the proposed algorithm. Also, limitations were put on the maximum number of hidden layers and maximum number of neurons per hidden layer in order to reduce the computation time. This aspect of the algorithm also needs to be further investigated.

Similar to the optimization of AJSSP, the proposed MMO algorithm was only utilized to minimize the makespan of JSSP, PFSSP, and AJSSP. Further analysis needs to be performed by utilizing the proposed MMO with other KPIs as the objective. The MMO algorithm should also be used to optimize real-life manufacturing problems which are more complex and consist of real-life constraints. Lastly, only a few MMO techniques were tested in dissertation. Further analysis should be done by modifying other existing MMO techniques and comparing the results obtained with those presented in this dissertation.

With the above enhancements to the current methodology and the algorithms, the techniques presented can progress towards becoming more reliable, and robust method for

identifying HCCOPs and optimizing them. The results will be realized as a useful tool in solving various optimization problems faced in the manufacturing industry as well as theoretical application.

REFERENCES

[1] Hartigan, J.A. and Wong, M.A., 1979. Algorithm AS 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1), pp.100-108.

[2] Coello, C.A.C., 2002. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Computer methods in applied mechanics and engineering, 191(11), pp.1245-1287.

[3] De Melo, V.V. and Iacca, G., 2014. A modified covariance matrix adaptation evolution strategy with adaptive penalty function and restart for constrained optimization. Expert Systems with Applications, 41(16), pp.7077-7094.

[4] Gandomi, A.H., Yang, X.S., Alavi, A.H. and Talatahari, S., 2013. Bat algorithm for constrained optimization tasks. Neural Computing and Applications, 22(6), pp.1239-1255.

[5] Pearson, J.W., Stoll, M. and Wathen, A.J., 2014. Preconditioners for state-constrained optimal control problems with Moreau–Yosida penalty function. Numerical Linear Algebra with Applications, 21(1), pp.81-97.

[6] Watanabe, K. and Hashem, M.M.A., 2004. Evolutionary Optimization of Constrained Problems. In Evolutionary Computations (pp. 53-64). Springer Berlin Heidelberg.

[7] Ali, M.M., Golalikhani, M. and Zhuang, J., 2014. A computational study on different penalty approaches for solving constrained global optimization problems with the electromagnetism-like method. Optimization, 63(3), pp.403-419.

[8] Snyman, J.A., Stander, N. and Roux, W.J., 1994. A dynamic penalty function method for the solution of structural optimization problems. Applied Mathematical Modelling, 18(8), pp.453-460.

[9] Koziel, S. and Michalewicz, Z., 1999. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization.Evolutionary computation, 7(1), pp.19-44.

[10] Monson, C.K. and Seppi, K.D., 2005, September. Linear equality constraints and homomorphous mappings in PSO. In 2005 IEEE Congress on Evolutionary Computation (Vol. 1, pp. 73-80).

[11] Michalewicz, Z. and Janikow, C.Z., 1996. GENOCOP: a genetic algorithm for numerical optimization problems with linear constraints. Communications of the ACM, 39(12es), p.175.

[12] Chootinan, P. and Chen, A., 2006. Constraint handling in genetic algorithms using a gradient-based repair method. Computers & operations research,33(8), pp.2263-2281.

[13] Pal, K., Saha, C., Das, S. and Coello, C.A.C., 2013, June. Dynamic constrained optimization with offspring repair based gravitational search algorithm. In 2013 IEEE Congress on Evolutionary Computation (pp. 2414-2421).

[14] Michalewicz, Z. and Nazhiyath, G., 1995, December. Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In Evolutionary Computation, 1995., IEEE International Conference on (Vol. 2, pp. 647-651).

[15] Ameca-Alducin, M.Y., Mezura-Montes, E. and Cruz-Ramírez, N., 2015, July. A Repair Method for Differential Evolution with Combined Variants to Solve Dynamic Constrained Optimization Problems. In Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (pp. 241-248).

[16] Fan, Z., Li, W., Cai, X., Lin, H., Xie, S. and Goodman, E., 2015. A new repair operator for multi-objective evolutionary algorithm in constrained optimization problems. arXiv preprint arXiv:1504.00154.

[17] Paredis, J., 1995. Coevolutionary computation. Artificial life, 2(4), pp.355-375.

[18] Schoenauer, M. and Xanthakis, S., 1993, July. Constrained GA optimization. In ICGA (pp. 573-580).

[19] Deb, K., 2000. An efficient constraint handling method for genetic algorithms. Computer methods in applied mechanics and engineering,186(2), pp.311-338.

[20] Ma, H. and Simon, D., 2011. Blended biogeography-based optimization for constrained optimization. Engineering Applications of Artificial Intelligence,24(3), pp.517-525.

[21] Chen, H., Zhou, Y., Guo, P., Ouyang, X., He, S. and Zheng, H., 2013. A hybrid invasive weed optimization with feasibility-based rule for constrained optimization problem. Przegląd Elektrotechniczny, 89(4), pp.160-167.

[22] Zhou, Y., Zhou, G. and Zhang, J., 2013. A hybrid glowworm swarm optimization algorithm for constrained engineering design problems. Appl. Math. Inf. Sci, 7(1), pp.379-388.

[23] Mohamed, A.W. and Sabry, H.Z., 2012. Constrained optimization based on modified differential evolution algorithm. Information Sciences, 194, pp.171-208.

[24] Pérez, E., Posada, M. and Herrera, F., 2012. Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling. Journal of Intelligent manufacturing, 23(3), pp.341-356.

[25] Goldberg, D.E. and Richardson, J., 1987, July. Genetic algorithms with sharing for multimodal function optimization. In Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms (pp. 41-49). Hillsdale, NJ: Lawrence Erlbaum.

[26] De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems (Doctoral dissertation, University of Michigan). Dissertation Abstracts International, 36(10), 5140B. (University Microlms No. 76-9381).

[27] Li, X., Epitropakis, M.G., Deb, K. and Engelbrecht, A., 2017. Seeking multiple solutions: an updated survey on niching methods and their applications. IEEE Transactions on Evolutionary Computation, 21(4), pp.518-538.

[28] Harik, G.R., 1995, July. Finding Multimodal Solutions Using Restricted Tournament Selection. In ICGA (pp. 24-31).

[29] Pétrowski, A., 1996, May. A clearing procedure as a niching method for genetic algorithms. In Evolutionary Computation, 1996., Proceedings of IEEE International Conference on (pp. 798-803). IEEE.

[30] Ursem, R.K., 1999. Multinational evolutionary algorithms. In Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on (Vol. 3, pp. 1633-1640). IEEE.

[31] Li, J.P., Balazs, M.E., Parks, G.T. and Clarkson, P.J., 2002. A species conserving genetic algorithm for multimodal function optimization. Evolutionary computation, 10(3), pp.207-234.

[32] Brits, R., Engelbrecht, A.P. and van den Bergh, F., 2007. Locating multiple optima using particle swarm optimization. Applied Mathematics and Computation, 189(2), pp.1859-1883.

[33] Bird, S. and Li, X., 2006, July. Adaptively choosing niching parameters in a PSO. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (pp. 3-10). ACM.

[34] Parrott, D. and Li, X., 2006. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Transactions on Evolutionary Computation, 10(4), pp.440-458.

[35] Zhan, Z.H., Zhang, J., Li, Y. and Chung, H.S.H., 2009. Adaptive particle swarm optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(6), pp.1362-1381.

[36] Fleetwood, K., 2004, November. An introduction to differential evolution. In Proceedings of Mathematics and Statistics of Complex Systems (MASCOS) One Day Symposium, 26th November, Brisbane, Australia.

[37] Biswas, S., Kundu, S. and Das, S., 2015. Inducing niching behavior in differential evolution through local information sharing. IEEE Transactions on Evolutionary Computation, 19(2), pp.246-263.

[38] Biswas, S., Kundu, S. and Das, S., 2014. An improved parent-centric mutation with normalized neighborhoods for inducing niching behavior in differential evolution. IEEE transactions on cybernetics, 44(10), pp.1726-1737.

[39] Hui, S. and Suganthan, P.N., 2016. Ensemble and arithmetic recombination-based speciation differential evolution for multimodal optimization. IEEE transactions on cybernetics, 46(1), pp.64-74.

[40] Hui, S. and Suganthan, P.N., 2016. Ensemble and arithmetic recombination-based speciation differential evolution for multimodal optimization. IEEE transactions on cybernetics, 46(1), pp.64-74.

[41] Della Cioppa, A., De Stefano, C. and Marcelli, A., 2007. Where are the niches? Dynamic fitness sharing. IEEE Transactions on Evolutionary Computation, 11(4), pp.453-465.

[42] Li, J.P., Balazs, M.E., Parks, G.T. and Clarkson, P.J., 2002. A species conserving genetic algorithm for multimodal function optimization. Evolutionary computation, 10(3), pp.207-234.

[43] Pereira, M.T. and Santoro, M.C., 2011. An integrative heuristic method for detailed operations scheduling in assembly job shop systems. International Journal of Production Research, 49(20), pp.6089-6105.

[44] Fattahi, P., Hosseini, S.M.H., Jolai, F. and Tavakkoli-Moghaddam, R., 2014. A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations. Applied Mathematical Modelling, 38(1), pp.119-134.

[45] Komaki, G.M. and Kayvanfar, V., 2015. Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. Journal of Computational Science, 8, pp.109-120.

[46] Yan, H.S., Wan, X.Q. and Xiong, F.L., 2014. A hybrid electromagnetism-like algorithm for two-stage assembly flow shop scheduling problem. International Journal of Production Research, 52(19), pp.5626-5639.

[47] Komaki, G.M., Teymourian, E. and Kayvanfar, V., 2016. Minimising makespan in the two-stage assembly hybrid flow shop scheduling problem using artificial immune systems. International Journal of Production Research, 54(4), pp.963-983

[48] Wong, T.C. and Ngan, S.C., 2013. A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop. Applied Soft Computing, 13(3), pp.1391-1399

[49] Natarajan, K., Mohanasundaram, K.M., Babu, B.S., Suresh, S., Raj, K.A.A.D. and Rajendran, C., 2007. Performance evaluation of priority dispatching rules in multi-level assembly job shops with jobs having weights for flowtime and tardiness. The International Journal of Advanced Manufacturing Technology, 31(7), pp.751-761.

[50] Paul, M., Sridharan, R. and Ramanan, T.R., 2015. An Investigation of Order Review/Release Policies and Dispatching Rules for Assembly Job Shops with Multi Objective Criteria. Procedia-Social and Behavioral Sciences, 189, pp.376-384.

[51] Chan, F.T.S., Wong, T.C. and Chan, L.Y., 2008. Lot streaming for product assembly in job shop environment. Robotics and Computer-Integrated Manufacturing, 24(3), pp.321-331

[52] Guo, Z.X., Wong, W.K., Leung, S.Y.S., Fan, J.T. and Chan, S.F., 2006. Mathematical model and genetic optimization for the job shop scheduling problem in a mixed-and multi-product assembly environment: a case study based on the apparel industry. Computers & Industrial Engineering, 50(3), pp.202-219.

[53] Thiagarajan, S. and Rajendran, C., 2003. Scheduling in dynamic assembly job-shops with jobs having different holding and tardiness costs. International Journal of Production Research, 41(18), pp.4453-4486.

[54] Fuji, W., Jianwei, M., Di, S., Wei, L. and Xiaohong, L., 2012, July. Research on repair operators in the whole space search genetic algorithm of assembly job shop scheduling problem. In 2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 1922-1927

[55] Liao, C.J., Lee, C.H. and Lee, H.C., 2015. An efficient heuristic for a two-stage assembly scheduling problem with batch setup times to minimize makespan. Computers & Industrial Engineering, 88, pp.317-325

[56] Seidgar, H., Kiani, M., Abedi, M. and Fazlollahtabar, H., 2014. An efficient imperialist competitive algorithm for scheduling in the two-stage assembly flow shop problem. International Journal of Production Research, 52(4), pp.1240-1256

[57] Seidgar, H., Zandieh, M., Fazlollahtabar, H. and Mahdavi, I., 2016. Simulated imperialist competitive algorithm in two-stage assembly flow shop with machine breakdowns and preventive maintenance. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 230(5), pp.934-953.

[58] Dileeplal, J. and Narayanan, K.P., 2012. Multi-objective assembly job shop scheduling using genetic algorithm and tabu search (Doctoral dissertation, Cochin University of Science and Technology).

[59] Narendra, K.S. and Parthasarathy, K., 1990. Identification and control of dynamical systems using neural networks. IEEE Transactions on neural networks, 1(1), pp.4-27.

[60] Sedki, A., Ouazar, D. and El Mazoudi, E., 2009. Evolving neural network using real coded genetic algorithm for daily rainfall–runoff forecasting. Expert Systems with Applications, 36(3), pp.4523-4527.

[61] Karegowda, A.G., Manjunath, A.S. and Jayaram, M.A., 2011. Application of genetic algorithm optimized neural network connection weights for medical diagnosis of pima Indians diabetes. International Journal on Soft Computing,2(2), pp.15-23.

[62] Ding, S., Su, C. and Yu, J., 2011. An optimizing BP neural network algorithm based on genetic algorithm. Artificial Intelligence Review, 36(2), pp.153-162.

[63] Kim, S. and Kim, H.S., 2008. Neural networks and genetic algorithm approach for nonlinear evaporation and evapotranspiration modeling. Journal of Hydrology, 351(3), pp.299-317.

[64] Fu, Z., Mo, J., Chen, L. and Chen, W., 2010. Using genetic algorithm-back propagation neural network prediction and finite-element model simulation to optimize the process of multiple-step incremental air-bending forming of sheet metal. Materials & design, 31(1), pp.267-277.

[65] Karaboga, D. and Ozturk, C., 2009. Neural networks training by artificial bee colony algorithm on pattern classification. Neural Network World, 19(3), p.279.

[66] Ozturk, C. and Karaboga, D., 2011, June. Hybrid artificial bee colony algorithm for neural network training. In 2011 IEEE Congress of Evolutionary Computation (CEC) (pp. 84-88). IEEE.

[67] Irani, R. and Nasimi, R., 2011. Application of artificial bee colony-based neural network in bottom hole pressure prediction in underbalanced drilling.Journal of Petroleum Science and Engineering, 78(1), pp.6-12.

[68] Zhang, Y., Wu, L. and Wang, S., 2011. Magnetic resonance brain image classification by an improved artificial bee colony algorithm. Progress In Electromagnetics Research, 116, pp.65-79.

[69] Mirjalili, SeyedAli, Siti Zaiton Mohd Hashim, and Hossein Moradian Sardroudi. "Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm." Applied Mathematics and Computation 218.22 (2012): 11125-11137.

[70] Amjady, N., Keynia, F. and Zareipour, H., 2011. Wind power prediction by a new forecast engine composed of modified hybrid neural network and enhanced particle swarm optimization. IEEE transactions on sustainable energy, 2(3), pp.265-276.

[71] Bashir, Z.A. and El-Hawary, M.E., 2009. Applying wavelets to short-term load forecasting using PSO-based neural networks. IEEE transactions on power systems, 24(1), pp.20-27

[72] Das, G., Pattnaik, P.K. and Padhy, S.K., 2014. Artificial Neural Network trained by Particle Swarm Optimization for non-linear channel equalization.Expert Systems with Applications, 41(7), pp.3491-3496.

[73] Loghmanian, S.M.R., Jamaluddin, H., Ahmad, R., Yusof, R. and Khalid, M., 2012. Structure optimization of neural network for dynamic system modeling using multi-objective genetic algorithm. Neural Computing and Applications, 21(6), pp.1281-1295.

[74] Zhang, C., Shao, H. and Li, Y., 2000. Particle swarm optimisation for evolving artificial neural network. In Systems, Man, and Cybernetics, 2000 IEEE International Conference on (Vol. 4, pp. 2487-2490). IEEE

[75] Mendivil, S.G., Castillo, O. and Melin, P., 2008. Optimization of artificial neural network architectures for time series prediction using parallel genetic algorithms. In Soft Computing for Hybrid Intelligent Systems (pp. 387-399). Springer Berlin Heidelberg

[76] Kopel, A., 2012. NEURAL NETWORKS PERFORMANCE AND STRUCTURE OPTIMIZATION USING GENETIC ALGORITHMS (Doctoral dissertation, California Polytechnic State University, San Luis Obispo)

[77] Levenberg, K., 1944. A method for the solution of certain non-linear problems in least squares. Quarterly of applied mathematics, 2(2), pp.164-168.

[78] Azimi, H., Bonakdari, H., Ebtehaj, I. and Michelson, D.G., A combined adaptive neuro-fuzzy inference system–firefly algorithm model for predicting the roller length of a hydraulic jump on a rough channel bed. Neural Computing and Applications, pp.1-10.

[79] Jang, J.S., 1993. ANFIS: adaptive-network-based fuzzy inference system.IEEE transactions on systems, man, and cybernetics, 23(3), pp.665-685

[80] Yang, X.S., 2010. Firefly algorithm, stochastic test functions and design optimisation. International Journal of Bio-Inspired Computation, 2(2), pp.78-84.

[81] Carollo, F.G., Ferro, V. and Pampalone, V., 2012. New Expression of the Hydraulic Jump Roller Length. Journal of Hydraulic Engineering, 138(11), pp.995-999.

[82] Carollo, F.G., Ferro, V. and Pampalone, V., 2007. Hydraulic jumps on rough beds. Journal of Hydraulic Engineering, 133(9), pp.989-999.

[83] Dewan, M.W., Huggett, D.J., Liao, T.W., Wahab, M.A. and Okeil, A.M., 2016. Prediction of tensile strength of friction stir weld joints with adaptive neuro-fuzzy inference system (ANFIS) and neural network. Materials & Design, 92, pp.288-299.

[84] Wang, X. and Feng, C.X., 2002. Development of empirical models for surface roughness prediction in finish turning. The International Journal of Advanced Manufacturing Technology, 20(5), pp.348-356.

[85] Lin, W.S., Lee, B.Y. and Wu, C.L., 2001. Modeling the surface roughness and cutting force for turning. Journal of Materials Processing Technology,108(3), pp.286-293.

[86] Kirby, E.D., Zhang, Z. and Chen, J.C., 2004. Development of an accelerometer-based surface roughness prediction system in turning operations using multiple regression techniques. Journal of Industrial Technology, 20(4), pp.1-8.

[87] Zhang, G., Li, J., Chen, Y., Huang, Y., Shao, X. and Li, M., 2014. Prediction of surface roughness in end face milling based on Gaussian process regression and cause analysis considering tool vibration. The International Journal of Advanced Manufacturing Technology, 75(9-12), pp.1357-1370.

[88] Wang, Y., 2012. A new hybrid genetic algorithm for job shop scheduling problem. Computers & Operations Research, 39(10), pp.2291-2299.

[89] Jorapur, V.S., Puranik, V.S., Deshpande, A.S. and Sharma, M., 2016. A promising initial population based genetic algorithm for job shop scheduling problem. Journal of Software Engineering and Applications, 9(05), p.208.

[90] Chang, H.C., Chen, Y.P., Liu, T.K. and Chou, J.H., 2015. Solving the flexible job shop scheduling problem with makespan optimization by using a hybrid Taguchi-genetic algorithm. IEEE Access, 3, pp.1740-1754.

[91] Bagheri, A., Zandieh, M., Mahdavi, I. and Yazdani, M., 2010. An artificial immune algorithm for the flexible job-shop scheduling problem. Future Generation Computer Systems, 26(4), pp.533-541.

[92] Improved immune algorithm for global numerical optimization and job-shop scheduling problems.

[93] Luh, G.C. and Chueh, C.H., 2009. A multi-modal immune algorithm for the job-shop scheduling problem. Information Sciences, 179(10), pp.1516-1532.

[94] Bruns, R., 1993, June. Direct chromosome representation and advanced genetic operators for production scheduling. In Proceedings of the 5th International Conference on Genetic Algorithms (pp. 352-359). Morgan Kaufmann Publishers Inc.

[95] Nakano, R. and Yamada, T., 1991, July. Conventional genetic algorithm for job shop problems. In ICGA (Vol. 91, pp. 474-479)

[96] Fang, H.L., Ross, P. and Corne, D., 1993. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems (pp. 375-382). University of Edinburgh, Department of Artificial Intelligence.

[97] Mattfeld, D.C., 2013. Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling. Springer Science & Business Media

[98] Liu, Y.F. and Liu, S.Y., 2013. A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. Applied Soft Computing, 13(3), pp.1459-1463.

[99] Govindan, K., Balasundaram, R., Baskar, N. and Asokan, P., 2017. A hybrid approach for minimizing makespan in permutation flowshop scheduling. Journal of Systems Science and Systems Engineering, 26(1), pp.50-76.

[100] Ancău, M., 2012. On solving flowshop scheduling problems. Proceedings of the Romanian Academy. Series A, 13(1), pp.71-79

[101] Zobolas, G.I., Tarantilis, C.D. and Ioannou, G., 2009. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Computers & Operations Research, 36(4), pp.1249-1267