MATHEMATICAL FORMULATION AND SCHEDULING HEURISTICS FOR CYCLIC PERMUTATION FLOW-SHOPS

A dissertation presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Doctor of Philosophy

Arun N. Nambiar
August 2007
This dissertation entitled

MATHEMATICAL FORMULATION AND SCHEDULING HEURISTICS
FOR CYCLIC PERMUTATION FLOW-SHOPS

by

ARUN N.NAMBIAR

has been approved for

the Department of Industrial and Systems Engineering
and the Russ College of Engineering and Technology of Ohio University by

Robert P. Judd
Professor of Industrial and Systems Engineering

Dennis Irwin
Dean, Russ College of Engineering and Technology
Abstract

NAMBIAR, ARUN N., Ph.D., August 2007. Industrial and Systems Engineering

MATHEMATICAL FORMULATION AND SCHEDULING HEURISTICS FOR CYCLIC PERMUTATION FLOW-SHOPS (169pp.)

Director of dissertation: Robert P. Judd

Scheduling is a decision-making process that concerns the allocation of limited resources to a set of tasks with the view of optimizing one or more objectives. The primary focus of this work is the cyclic permutation flow-shop problem where a set of parts is repeatedly produced (cyclic) and the sequence of parts on all the machines remains the same (permutation). A mathematical formulation for the above problem is developed using max-plus algebra. A new concept called opportunities that identifies potential areas for improving the existing schedule is also presented. New heuristic approaches are proposed to find the optimal or sub-optimal solutions to the scheduling problem using the aforementioned mathematical formulation. The analysis of the results obtained using the developed heuristics and some of the existing heuristics on Taillard’s benchmark problems have shown that the developed heuristics produce solutions of better quality and incur significantly lower computation time than the existing heuristics that were investigated.

Approved: 

Robert P. Judd
Professor of Industrial and Systems Engineering
Acknowledgments

First of all, I express my sincere thanks to my dissertation advisor, Dr. Robert P. Judd without whose expert guidance and support this research would not have been possible. I also thank all the members of my dissertation committee for their invaluable suggestions and assistance in bringing this work to completion. I also thank my brother for all the morale-boosting and confidence-inspiring conversations that were of significant help to me while I worked on my dissertation. I thank my parents for having stood by me through all these years that I spent working on my dissertation.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>Acknowledgments</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>List of Figures</strong></td>
<td>8</td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>11</td>
</tr>
<tr>
<td><strong>2 Literature Review</strong></td>
<td>13</td>
</tr>
<tr>
<td>2.1 Enumerative Techniques</td>
<td>13</td>
</tr>
<tr>
<td>2.2 Approximation Algorithms</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Heuristic Algorithms</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1 Construction Algorithms</td>
<td>15</td>
</tr>
<tr>
<td>2.3.2 Improvement Algorithms</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2.1 Ant Colony Optimization</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2.2 Artificial Immune Systems</td>
<td>18</td>
</tr>
<tr>
<td>2.3.2.3 Genetic Algorithm</td>
<td>19</td>
</tr>
<tr>
<td>2.3.2.4 Neural Network</td>
<td>22</td>
</tr>
<tr>
<td>2.3.2.5 Simulated Annealing</td>
<td>25</td>
</tr>
<tr>
<td>2.3.2.6 Tabu Search</td>
<td>28</td>
</tr>
<tr>
<td>2.3.2.7 Hybrid Metaheuristics</td>
<td>31</td>
</tr>
<tr>
<td>2.3.3 Rescheduling</td>
<td>33</td>
</tr>
<tr>
<td>2.3.4 Cyclic Systems</td>
<td>36</td>
</tr>
<tr>
<td><strong>3 Problem Statement</strong></td>
<td>38</td>
</tr>
<tr>
<td>3.1 Drawbacks/Limitations of existing approaches</td>
<td>38</td>
</tr>
<tr>
<td>3.1.1 Period versus Makespan</td>
<td>38</td>
</tr>
<tr>
<td>3.1.2 Buffer versus No Buffer</td>
<td>41</td>
</tr>
<tr>
<td>3.1.3 Static versus Dynamic</td>
<td>41</td>
</tr>
<tr>
<td>3.2 Objectives and Significant Contributions</td>
<td>42</td>
</tr>
<tr>
<td>3.3 Assumptions</td>
<td>43</td>
</tr>
<tr>
<td><strong>4 Max Plus Algebra</strong></td>
<td>45</td>
</tr>
<tr>
<td>4.1 Basics</td>
<td>45</td>
</tr>
<tr>
<td>4.2 Eigenvalue and Eigenvectors</td>
<td>46</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>8.1.2.2</td>
<td>Improvement Heuristics</td>
</tr>
<tr>
<td>8.2</td>
<td>Results for Taillard's Benchmark Problems</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Deviation from Maximum Load</td>
</tr>
<tr>
<td>8.2.1.1</td>
<td>Construction Heuristics</td>
</tr>
<tr>
<td>8.2.1.2</td>
<td>Improvement Heuristics</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Computation Time</td>
</tr>
<tr>
<td>8.2.2.1</td>
<td>Construction Heuristics</td>
</tr>
<tr>
<td>8.2.2.2</td>
<td>Improvement Heuristics</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Number of Swaps and Moves</td>
</tr>
<tr>
<td>8.3</td>
<td>Early Termination</td>
</tr>
<tr>
<td>8.4</td>
<td>Batch Size-based Comparison</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Deviation from Maximum Load</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Computation Times and Number of Sequences</td>
</tr>
<tr>
<td>9</td>
<td>Conclusions and Future Work</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Algorithms</td>
</tr>
<tr>
<td>A.1</td>
<td>Algorithm to find the best perturbation using opportunities</td>
</tr>
<tr>
<td>A.2</td>
<td>Algorithm to find the best perturbation by exploring all possible alternatives</td>
</tr>
<tr>
<td>A.3</td>
<td>Algorithm to compute the best possible location to move a part</td>
</tr>
<tr>
<td>A.4</td>
<td>Algorithm for MP_CSoM heuristic</td>
</tr>
<tr>
<td>A.5</td>
<td>Algorithm for MP_OSoM heuristic</td>
</tr>
<tr>
<td>A.6</td>
<td>Algorithm for MP_PFOSoM heuristic</td>
</tr>
<tr>
<td>A.7</td>
<td>Algorithm to find the opportunities and the corresponding pivot parts</td>
</tr>
<tr>
<td>A.8</td>
<td>Algorithm to find the best possible swap pair</td>
</tr>
<tr>
<td>B</td>
<td>Results</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Gantt Chart for a flow-shop</td>
<td>39</td>
</tr>
<tr>
<td>5.1 Gantt Chart showing the start times on the first machine</td>
<td>48</td>
</tr>
<tr>
<td>5.2 Gantt Chart showing the start times on the last machine</td>
<td>49</td>
</tr>
<tr>
<td>5.3 Gantt Chart showing the start times on the intermediate machines</td>
<td>49</td>
</tr>
<tr>
<td>6.1 Gantt Chart for a flow-shop identifying opportunities</td>
<td>57</td>
</tr>
<tr>
<td>6.2 Efficient computation of intermediate product matrices for move</td>
<td>60</td>
</tr>
<tr>
<td>6.3 Efficient computation of intermediate product matrices for swap</td>
<td>60</td>
</tr>
<tr>
<td>7.1 Comparison of results for ACO heuristic</td>
<td>71</td>
</tr>
<tr>
<td>7.2 Comparison of percentage deviation from optimal for CDS heuristic</td>
<td>72</td>
</tr>
<tr>
<td>7.3 Comparison of relative makespan for 10 machines for CGA heuristics</td>
<td>74</td>
</tr>
<tr>
<td>7.4 Comparison of relative makespan for 15 machines for CGA heuristic</td>
<td>75</td>
</tr>
<tr>
<td>7.5 Comparison of average makespan for HAMC heuristic</td>
<td>77</td>
</tr>
<tr>
<td>7.6 Comparison of average makespan for CR heuristic</td>
<td>78</td>
</tr>
<tr>
<td>7.7 Comparison of relative makespan for ISA heuristic</td>
<td>79</td>
</tr>
<tr>
<td>7.8 Comparison of relative makespans for TS heuristic</td>
<td>80</td>
</tr>
<tr>
<td>7.9 Comparison of relative errors for NEH heuristic</td>
<td>82</td>
</tr>
<tr>
<td>7.10 Comparison of relative errors for WGA heuristic</td>
<td>83</td>
</tr>
<tr>
<td>8.1 Comparison of Construction heuristics for small problems</td>
<td>86</td>
</tr>
<tr>
<td>8.2 Results for the Random heuristic for small problems</td>
<td>87</td>
</tr>
<tr>
<td>8.3 Comparison of Improvement heuristics for small problems</td>
<td>89</td>
</tr>
<tr>
<td>8.4 Comparison of computation times for small problems</td>
<td>90</td>
</tr>
<tr>
<td>8.5 Comparison of computation time for 6p, 6m</td>
<td>92</td>
</tr>
<tr>
<td>8.6 Comparison of computation time for 7p, 7m</td>
<td>93</td>
</tr>
<tr>
<td>8.7 Comparison of computation time for 8p, 8m</td>
<td>94</td>
</tr>
<tr>
<td>8.8 Comparison of computation time for 8p, 4m</td>
<td>95</td>
</tr>
<tr>
<td>8.9 Comparison of computation time for Rand6-8</td>
<td>96</td>
</tr>
<tr>
<td>8.10 Comparison of computation time for Rand4-8</td>
<td>97</td>
</tr>
<tr>
<td>8.11 Comparison of Construction heuristics based on period for Taillard’s problems</td>
<td>99</td>
</tr>
<tr>
<td>8.12 Results for the Random heuristic for Taillard’s problems</td>
<td>100</td>
</tr>
<tr>
<td>8.13 Comparison of Improvement heuristics based on period for Taillard’s Problems</td>
<td>102</td>
</tr>
<tr>
<td>8.14 Comparison of computation time for Construction heuristics</td>
<td>103</td>
</tr>
<tr>
<td>8.15 Comparison of computation time of Improvement heuristics for 20p, 5m</td>
<td>105</td>
</tr>
<tr>
<td>8.16 Comparison of computation time of Improvement heuristics for 50p, 5m</td>
<td>106</td>
</tr>
<tr>
<td>8.17 Comparison of computation time of Improvement heuristics for 100p, 5m</td>
<td>107</td>
</tr>
</tbody>
</table>
8.18 Comparison of computation time of Improvement heuristics for 20p, 10m
8.19 Comparison of computation time of Improvement heuristics for 50p, 10m
8.20 Comparison of computation time of Improvement heuristics for 100p, 10m
8.21 Comparison of computation time of Improvement heuristics for 20p, 20m
8.22 Comparison of computation time of Improvement heuristics for 50p, 20m
8.23 Comparison of computation time of Improvement heuristics for 100p, 20m
8.24 Comparison of number of swaps for developed improvement heuristics
8.25 Comparison of number of moves for developed improvement heuristics
8.26 Progress of period over iterations using MP_COSoM for 20p, 20m
8.27 Percentage reduction in period over iterations from 2\textsuperscript{nd} iteration using MP_COSoM for 20p, 20m
8.28 Progress of period over iterations using MP_PFoSOM for 20p, 20m
8.29 Percentage reduction in period over iterations from 2\textsuperscript{nd} iteration using MP_PFoSOM for 20p, 20m
8.30 Comparison of Max-Plus based heuristics over batch size 1 and 3 for 20p, 20m
8.31 Comparison of computation times of Max-Plus based heuristics over batch sizes 1 and 3 for 20p, 20m
8.32 Comparison of number of sequences examined for Max-Plus based heuristics over batch sizes 1 and 3 for 20p, 20m
List of Tables

<table>
<thead>
<tr>
<th>Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Processing Times for a system of 5 parts and 5 machines</td>
<td>39</td>
</tr>
<tr>
<td>3.2 Completion Times for sequence 1-2-3-5-4</td>
<td>40</td>
</tr>
<tr>
<td>3.3 Completion Times for sequence 1-2-4-5-3</td>
<td>40</td>
</tr>
<tr>
<td>3.4 Completion Times for sequence 1-4-2-5-3</td>
<td>41</td>
</tr>
<tr>
<td>5.1 Processing Times</td>
<td>50</td>
</tr>
<tr>
<td>7.1 Sample problems used by Campbell et al.</td>
<td>70</td>
</tr>
<tr>
<td>7.2 Sample problems used by Chen et al.</td>
<td>73</td>
</tr>
<tr>
<td>7.3 Sample problems used by Ishibuchi et al.</td>
<td>76</td>
</tr>
<tr>
<td>B.1 Percentage Deviation from Maximum Load for Construction Heuristics on Small Problems</td>
<td>161</td>
</tr>
<tr>
<td>B.2 Percentage Deviation from Maximum Load for Improvements Heuristics on Small Problems</td>
<td>162</td>
</tr>
<tr>
<td>B.3 Execution Time (ms) of Construction Heuristics on Small Problems</td>
<td>162</td>
</tr>
<tr>
<td>B.4 Execution Time (ms) of Improvement Heuristics on Small Problems</td>
<td>163</td>
</tr>
<tr>
<td>B.5 Percentage Deviation from Maximum Load for Construction Heuristics on Large Problems</td>
<td>164</td>
</tr>
<tr>
<td>B.6 Percentage Deviation from Maximum Load for Improvement Heuristics on Large Problems</td>
<td>165</td>
</tr>
<tr>
<td>B.7 Execution Time (ms) for Construction Heuristics on Large Problems</td>
<td>166</td>
</tr>
<tr>
<td>B.8 Execution Time (ms) for Improvement Heuristics on Large Problems</td>
<td>167</td>
</tr>
<tr>
<td>B.9 Percentage Reduction in Period at Each Iteration for MP_PFOSoM Heuristic</td>
<td>168</td>
</tr>
<tr>
<td>B.10 Percentage Reduction in Period at Each Iteration for MP_COSoM</td>
<td>169</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

Scheduling is a decision-making process that concerns the allocation of limited resources to a set of tasks with the view of optimizing one or more objectives. In today’s world of global competition, effective scheduling becomes imperative in order to meet customer requirements as quickly as possible while maximizing the profits. Scheduling in manufacturing systems is typically associated with scheduling a set of jobs on a set of machines in order to maximize the throughput. Manufacturing systems can be broadly classified into two categories viz. job-shop and flow-shop. In a job-shop, each job undergoes a set of operations on a set of machines and the order of these operations on each job has no bearing on the order of operations on the other jobs. In a flow-shop, each job undergoes a set of operations on a set of machines in the same order. The primary focus of this work is the cyclic permutation flow-shop problem where a set of parts is repeatedly produced (cyclic) and the sequence of parts on all the machines remains the same (permutation). In the regular permutation flow-shop problem, there are $n!$ number of sequences in which $n$ parts can be scheduled on a set of machines. In order to find the best possible optimal sequence which maximizes the throughput of the system, it becomes necessary to examine all $n!$ possible sequences. Since the same set of parts is repeatedly produced in a cyclic permutation flow-shop, the total number of sequences in which $n$ parts can be scheduled on the machines reduces to $(n-1)!$. It can be seen that even for a small system of 10 parts, the total number of sequences is a very large figure (362,880). Hence, researchers have focused on developing heuristic methods to obtain the solution that is either optimal or near-optimal. The objective of this work is to develop a new mathematical formulation for the cyclic permutation flow-shop problem.
and to develop heuristic algorithms using the mathematical formulation to obtain the
best possible solution which maximizes the throughput of the system. Throughout
this work, it is assumed that there are no buffers in between machines and the parts
are scheduled on the machines as soon as the machine becomes available.

Chapter 2 gives an extensive literature review of the various approaches currently
available to solve the flow-shop scheduling problem. Chapter 3 briefly states the
problem and its attributes. Chapter 4 gives an overview of Max-Plus algebra which
lays the groundwork for the mathematical model developed. Chapter 5 explains the
mathematical formulation developed in this work along with a brief description of
certain pertinent concepts. Chapter 6 gives a detailed explanation of all the heuristic
algorithms developed. Chapter 7 goes into the details of the experimental analysis
carried out in order to compare the various heuristic algorithms the results of which
are presented in Chapter 8. Chapter 9 presents the conclusions based on the results
of the experimental analysis.
Chapter 2: Literature Review

Permutation flow-shop sequencing problem is a known NP-hard problem. Only exhaustive search guarantees the optimal permutation. But these can become prohibitively expensive to compute even for small problems \((66)\). It has been shown \((17)\) that an optimal permutation schedule yields performance comparable to that of an optimal general flow-shop schedule. In this section, we look into the existing techniques that have been explored to solve the flow-shop sequencing problem. These may be broadly classified into three categories viz. *enumerative*, *approximation*, and *heuristics*. These categories have been extensively studied below.

### 2.1 Enumerative Techniques

One of the enumerative techniques to obtain an optimal solution is the straightforward enumeration where every possible solution is explored in order to find the optimal solution. However, due the computational complexity that this method engenders, it is not practical even for problems of moderate sizes. In the case of implicit enumeration, some of the solutions can be eliminated due to feasibility issues and hence this reduces the number of solutions to be evaluated marginally.

Another enumerative technique is the branch and bound technique where the known upper bounds or lower bounds for the solution are used to restrict the search space. The efficiency of these algorithms is dependent on the quality of the lower bounds. Ignall and Schrage \((38)\) developed the first branch and bound algorithms for the permutation flow-shop problem with makespan minimization. Carlier et al. \((12)\) propose two branch and bound algorithms for the permutation flow-shop problem. They use disjunctive graphs to model the flow-shop problem. The authors associate
with each operation on a machine, a unique value of head, and tail. The head is length of the longest path in the disjunctive graph from the source to the operation and tail represents the length of the longest path from the current operation to the end. The branching rule used by the authors is such that the first branching sequences a job at the beginning of the sequence, and the second branching sequences a job at the end of the sequence. All the remaining branches place a part alternatively at the beginning and end of the sequence. The node with the smallest value for the lower bound is selected as the branching node.

Ladhari et al. (42) reviews some of the existing branch and bound algorithms and the various lower bound computations. The authors also propose a new branch and bound algorithm with tighter lower bounds. The authors use the same branching rules and branching node selection methods as Carlier et al. (12). Based on experiments on benchmark problems (91), the authors have found that their algorithm not only outperforms other branch and bound algorithms but also can solve problems with 8000 operations. Temiz et al. (95) propose a modified branch and bound algorithm for a three machine flow-shop sequencing problem. The authors have modified the branch and bound algorithm proposed by Ignall and Schrage (38) to handle varying processing times.

2.2 Approximation Algorithms

Another approach often found in literature to handle the intractability of permutation flow shop problem is to devise polynomial time approximation algorithms which guarantee schedules within an acceptable range of the lower bound. Hall (29) proposes an approximation algorithm based on the outline scheme. In this scheme, the set of feasible schedules is divided into groups based on common characteristics.
The scheme then generates the best schedule for each group. The scheme guarantees a schedule that is as close to the optimal schedule as possible because the best schedule must lie in one of the groups. Although this algorithm yields schedules within an additive term of the minimum makespan, the underlying outline scheme is computationally very expensive. Sviridenko (89) proposes two approximation algorithms to minimize the makespan in a permutation flow shop. The author also uses the outline scheme mentioned but partitions jobs into groups. The schedule is generated by randomly choosing jobs from within a group and across groups such that the makespan is within a factor of the simple lower bound computed by taking the maximum of the loads on machines and total processing times of jobs.

2.3 Heuristic Algorithms

Since the permutation flow-shop problem is a NP-hard problem with computations being prohibitively expensive, a practical approach is to use heuristic methods to obtain near-optimal solutions. These heuristic methods can typically be classified into two categories viz. constructive and improvement (61). In constructive heuristics, once a permutation sequence is obtained, it is fixed and cannot be altered whereas in improvement heuristics, an initial solution is iteratively improved upon.

2.3.1 Construction Algorithms

Johnson first studied the two-machine problem with the objective of minimizing makespan (4). This approach basically divides the jobs into two categories and sequences them from left to right and right to left respectively. It is also referred to as the (SPT (1)-LPT (2)) problem where SPT is the Shortest Processing Time and
LPT stands for Largest Processing Time. Many researchers have generalized this to \( n \) machines.

Palmer (63) proposed ranking the jobs based on a slope index computed from the processing times thereby giving preference to jobs that tend to progress from low to high processing times. Gupta (28) modified Palmer’s slope index based on the principle of sorting \( n \) items based on a function. Hundal et al. (37) also propose a heuristic for the flow-shop scheduling problem based on Palmer’s heuristic. Some of the other heuristics include CDS (11) (applies Johnson’s algorithm in stages), NEH (54) (a constructive heuristic where jobs are sorted based on LPT and are scheduled one at a time), and RA algorithm (17) (a weighting scheme based on Palmer’s slope index (63), and CDS method (11)).

McCormick et al. (49) proposed a Profile Fitting (PF) heuristic based on the departure times of the jobs computed after each job has been scheduled. Ronconi (73) develops a MinMax (MM) algorithm which also addresses the flow-shop makespan minimization problem with no buffers. The author found that PF heuristic when used in the initial stage of NEH heuristic yield better results for large problems. The author also found that the both MM and PF heuristics when used in the initial stage of NEH outperforms the original NEH which uses LPT in the initial stage.

Kalczynski et al. (40) propose a construction heuristic for minimizing the makespan in a no-idle permutation flow-shop. The authors compare their algorithm with the modified NEH algorithm and show that for the benchmark problems (91) their algorithm performs better than the modified NEH algorithm.
2.3.2 Improvement Algorithms

Meta-heuristic is an algorithmic framework that can be applied to optimization problems based on the improvement methodology. It is essentially a randomized improvement heuristic (61). The basic principle behind improvement algorithms is to perturb existing solutions in an effort to achieve improved solutions. The perturbation mechanisms usually employed may be classified into two broad categories viz. neighborhood mechanism where the original solution is altered marginally and recombination mechanism where a new solution is obtained by combining parts of different good solutions (47). Osman et al. (61) provide an extensive bibliography on metaheuristics and its applications in combinatorial optimization problems. Some of the widely researched improvement algorithms include adaptive search techniques like genetic algorithm (GA), artificial immune system (AIS) and neural networks, neighborhood search techniques like simulated annealing (SA), and tabu search, and their hybrid versions. Besides these, there are other improvement algorithms that are still in their nascent stages of research like variable neighborhood search (32) which involves exploring multiple neighborhoods in the vicinity of the current solution.

2.3.2.1 Ant Colony Optimization

Ant Colony Optimization (ACO) algorithms, based on the food-hunting patterns of real ants, are population-based search algorithms that utilize agents (ants), single or multiple, to construct the optimal solution iteratively. The trails serve the purpose of remembering the previous solutions and have to be updated at the end of each iteration. An overview of ant colony optimization can be found in Maniezzo and Carbonaro (45).
Rajendran et al. (70) propose two ACO-based algorithms to minimize the total flow time in permutation flow-shops. The initialization parameters for the first algorithm are based on the MMAS of Stuetzle (87) and the selection of jobs to be appended is based on the summation rule proposed by Merkle and Middendorf (51). The algorithm generates an initial seed sequence and also carries out a local-search at the end of each iteration. The local search procedure is applied three times. The second algorithm differs from the first algorithm in the initialization and construction phases. A probability-based construction procedure is used to iteratively generate the solution. The local-search algorithm is applied in this case too. These two algorithms were run on Taillard’s benchmark problems (91) along with the current best algorithms (69; 43) for these problems. The authors observed that it is not possible to identify one heuristic that is best for the entire set of benchmark problems. The authors also observed that their proposed algorithms yield better results compared to the heuristic methods proposed by Liu and Reeves (43) and Rajendran and Ziegler (69).

Ying et al. (106) have developed a meta-heuristic based on ACO for the \( n/m/P/C_{\text{max}} \) problem where \( n \) is the number of parts, \( m \) is the number of machines, \( P \) represents a permutation sequence and \( C_{\text{max}} \) represents makespan by modeling the system using disjunctive graphs. For a more detailed explanation on the notations, the reader is directed to (66). Shyu et al. (82) develop an ACO-based algorithm to solve the two-machine flow-shop problem with no waiting between operations.

2.3.2.2 Artificial Immune Systems

Artificial immune system (AIS) is an adaptive heuristic technique inspired by the human immune system. The immune system consists of mainly lymphocytes. There
are two types of lymphocytes viz. the ones that originate and develop in the bone marrow (B-cells), and the ones that originate in the bone marrow and move to the thymus to grow (T-cells). Both these cells have receptors on their surface that attach themselves to disease causing pathogens thereby triggering the immune response. The two integral aspects in the immune response mechanism are the clonal selection principle and the affinity maturation the detailed account of which can be found in De Castro et al. (18).

Although both AIS and GA-based methods are population-based and evolutionary algorithms with a similar mechanism of population representation (chromosome versus antibody), and population evaluation functions (fitness function versus affinity function), AIS differs from GA in its evolutionary mechanism. In AIS, the population evolves using cloning mechanisms and mutation. The cloning mechanism is based on probability of selection which implies that there will be more clones of antibodies that are more likely to be selected. Mutation is applied only to those antibodies that have a lower probability of selection and the mutated antibodies replace the original ones if they are better in the same iteration.

Engin et al. (22) propose an AIS-based heuristic for hybrid flow shops. The authors applied their algorithm to the benchmark problems for hybrid flow shops and concluded that their algorithm performs better than Branch and Bound algorithms in terms of deviation from the known lower bounds for the benchmark problems.

2.3.2.3 Genetic Algorithm

Genetic Algorithm (GA)-based algorithms, inspired by Darwin's theory of natural selection and survival of the fittest, were invented by Holland (35) and first proposed for scheduling problems by Goldberg (26). In the Simple GA-based approach, the
various stages like evaluation, selection, crossover, and mutation are repeatedly executed after initialization until a stopping criterion is met. The algorithm works on multiple solutions simultaneously.

Chen et al. (15) propose a GA-based algorithm to minimize makespan for a flow-shop scheduling problem. The authors use CDS (11) and RA (17) methods to generate the initial population. Based on their experiments, the authors claim that increasing the population size beyond sixty does not improve the solution significantly. The genetic operator used in this method is Goldberg’s PMX operator (26). The authors also claim that a crossover rate of 1.0 and mutation rate of 0.0 yielded good results on all the trial examples. Hence the authors have considered only the crossover operator to generate offspring. The authors use 20 generations as the termination criterion for the heuristic. The results were compared with the results obtained using Widmer’s heuristic (101) and Ho and Chang’s heuristic (34). The authors evaluated their heuristic on 200 problems for 20 different combinations of parts and machines with processing times for operations uniformly distributed between 1 and 30. The authors conclude that the GA-based heuristic yields better average results (both in terms of makespan and CPU times) compared to the other two heuristics.

Rajendran and Chaudhuri (68) have proposed a heuristic algorithm to minimize flow time for a flow-shop scheduling problem using three heuristic criteria. The first criterion deals with the sum of idle times. The second criterion incorporates the sum of idle times and the waiting times. The third criterion includes the completion times of the partial schedule at various stages along with the above mentioned two criteria. Yamada et al. (104) propose an enhanced local search procedure to search valleys of solution space encountered in a GA-based heuristic.
Ravindran et al. (71) propose three new hybrid algorithms (HAMC1, HAMC2, and HAMC3) based on the GA approach to minimize makespan and total flow time in a flow-shop scheduling problem. The sequence obtained from Rajendran and Chaudhuri (68) is used as the initial seed sequence. Using the initial seed sequence, a population is generated by pair-wise swapping of parts during each iteration and the algorithm is executed for a randomly selected number (between 10 and 20) of iterations. The three algorithms differ in the mechanism of selecting the final sequence. In HAMC1, sequence with lowest makespan over all iterations is selected. In HAMC2, the sequence with minimum flow time is selected from sequences generated during all the iterations. In HAMC3, the sequence generated in the final iteration is selected. The authors show that the proposed hybrid approaches yield better results compared to Rajendran’s heuristic (68) for the Taillard’s benchmark problems (91).

Tang and Liu (93) propose a modified GA algorithm (MGA) to minimize the mean flow time for a flow-shop scheduling problem. The modification aims to achieve an efficient trade-off between solution quality and computational complexity. Two new operations are incorporated into the traditional GA-based approach. The filtering operation replaces the worst solution obtained in a generation with the best solution obtained in the previous generation in order to facilitate faster convergence. The cultivation operation handles the issue of premature convergence. If no significant improvement is achieved in a specified number of generations, a single iteration of adjacent pair-wise exchange search is carried out and the solution obtained is included in the next generation. The authors have shown that the MGA not only converges faster compared to the traditional GA but also achieves better solution quality.
2.3.2.4 Neural Network

Neural network-based models are mathematical models that are based on the biological nervous systems and their adaptive behavior. The network consists of layers of interconnected nodes or neurons. Typically, there are at least two layers, an input layer and an output layer. One of the most common networks is the Back Propagation Network (BPN) which consists of an input layer, and an output layer with one or more intermediate hidden layers. The networks need to be first trained using a set of examples before they can be employed to the actual problem. The method used to generate the examples to train the network and the training algorithm employed has a significant impact on the performance of the neural network-based model. One of the training algorithms used is the Back-Propagation algorithm (BP) (48). This algorithm aims to reduce the deviation between the desired objective function value and the actual objective function value. Since this algorithm has a slow convergence speed and has a tendency to get stuck in the local optima, another algorithm called the Delta-bar-delta (DBD) is often used in which the learning rate of the process is adjusted based on the variation of the training error. The DBD algorithm has a faster convergence speed.

Sabuncuoglu (75) provides a review of the existing literature in the area of application of artificial neural networks to scheduling problems. The author has classified the neural network applications into three broad categories based on the type of neural network employed viz. Hopfield networks, Competitive networks and Multi-layer perceptrons (MLP) or Back-propagation networks (BPN). Hopfield networks consists of a single layer of interconnected neurons which functions as both the input and the output elements of the network. The author has observed that although Hopfield
networks have been applied to scheduling problems yielding promising results, it involves excessive computation for larger size problems. Competitive networks contain inhibitory links that are dynamically added to the network due to the competition between parts to be scheduled as early as possible. According to the author, competitive networks have not been applied extensively to scheduling problems.

Multi-layer perceptron comprises of layers of neurons through which data propagates in the forward direction. When a MLP uses Back-Propagation as its learning algorithm, it is called the Back-propagation network. In the feed-forward phase, the network ”learns the dependencies between various variables and in the generalization phase, it estimates the output for new examples”. The author has concluded that back-propagation networks have been typically employed to select dispatching rules and do not perform well on extrapolation (performance deteriorates for larger size problems).

Sabuncuoglu et al. (76) have proposed a competitive neural network model for minimizing the makespan of a job-shop. In their model, the sequence of jobs on each machine is represented by an \( nxn \) matrix of neurons with each row representing a job. The job’s position in the sequence is represented along the columns. A neuron \( a_{i,j} \) is considered active if job \( i \) is assigned to the \( j^{th} \) position. There will be \( m \) such matrices for an \( m \)-machine problem. Each such matrix is called a layer. The final neutron matrix with the activation values obtained at the end of the process gives the feasible schedule. The authors have observed that the network produced optimal results in 18 out of the 25 problems considered.

Akyol (2) uses BPN network to model six different heuristic algorithms for the permutation flow-shop sequencing problem with unlimited buffer space associated with each machine. The BPN network is generally employed to predict the optimal
sequence for the flow-shop. However, the author has used it to predict the completion times of the jobs. These completion times are then compared with the actual completion times obtained through the heuristic algorithms in order to determine the performance of the neural network.

El-Bouri et al. (21) propose the use of neural networks to enhance local search in a permutation flow-shop scheduling problem. The authors train the neural network using a set of examples with 10 parts whose optimal solutions are known. The trained network is then used to predict a good initial sequence for search-based algorithms. Since the initial solution is reasonably accurate, the speed with which the search-based algorithms converge on the optimal or near-optimal solution is drastically improved. The authors compared the initial sequences obtained from their algorithm with those obtained using the NEH algorithm and the LWKR dispatching rule. Comparing the deviation of the initial sequences from the optimal solutions, the authors found that their algorithm performed as well as the NEH algorithm and the percentage of jobs that were more than 5 positions away from their optimal positions was found to be the least in the sequences obtained using their algorithm. Further, adjacent pair wise interchange of parts and general pair wise interchange of parts were employed as the two search procedures using the initial sequences obtained from the three algorithms. It was found that the neural network-based algorithm resulted in a faster convergence of the search procedures and the quality of the solution thus obtained (average flow time) was also much better than those obtained using the initial sequences generated by the other two algorithms.

Tang et al. (94) propose a neural network-based model and algorithm for a hybrid flow-shop with dynamic job arrivals. In a hybrid flow-shop, there are multiple identical machines in at least one of the stages and a job may be processed by any
of the machines in a stage. The neural network model consists of three sub-networks each corresponding to a distinct performance criterion like minimizing flow time, minimizing the tardy times and minimizing tardy jobs. The authors use simulation to generate the set of training examples and employ the faster DBD training algorithm. The model basically predicts the next job to be scheduled on a machine when the machine becomes available. The authors have compared their algorithm with the standard dispatching rules like SPT, FCFS, LWKR, and CR. Panwalkar et al. (64) give a more detailed explanation of these dispatching rules. The authors show that although the proposed algorithm is outperformed by dispatching rules in some cases, it yields consistently good results for the triple performance criteria of minimizing the average flow time, the average tardiness and the number of tardy jobs.

2.3.2.5 Simulated Annealing

Simulated Annealing (SA) algorithms, based on the analogy of annealing process of metals, were proposed by Metropolis (52) and were first applied to combinatorial optimization problems by Kirkpatrick et al. (41). SA is a stochastic optimization method for minimizing a function $f$ over a domain $S$ (41). SA is considered to be an improvement heuristic where a given initial solution is iteratively improved upon. A number of researchers have applied SA to scheduling problems. Some of them include Osman and Potts (62), and Ogbu and Smith (58 59). The advantage of SA-based methods is their ability to avoid getting trapped in local minimums by accepting solutions that worsen the objective function. The number of iterations required to converge to the optimal solution depends on the parameters selected for the heuristic. The parameters may be classified into two categories viz. *problem-specific* like initial solution, neighborhood generation and *generic* like temperature, stopping criterion
among others. It has been shown that a good set of parameters can result in much quicker convergence to the optimal solution (107). Thus, in SA-based methods, it is imperative to obtain the most desirable set of parameters in order to achieve faster convergence within reasonable computation time.

Osman and Potts (62) propose a set of four different SA-based heuristic algorithms for the flow-shop problem and have shown that their algorithms yield better results compared to NEH algorithm which is currently the best construction heuristic for flow-shop sequencing problems. Ogbu and Smith (58) propose a modified SA-based heuristic algorithm using constructive heuristics to obtain initial solution and a novel acceptance probability function.

Zegordi et al. (107) provide a SA-based heuristic for minimizing makespan for a flow-shop. Their heuristic is based on an index called the Move Desirability for Jobs (MDJ) using which pair-wise exchanges are selected. The MDJ index is computed by approximating an $m$-machine $n$-part problem to a 2-machine problem using the Palmer’s index (63). According to Palmer’s index, jobs that progress from lower processing times to higher processing times as they move through the order of machines are given higher priority than jobs that have progressively decreasing processing times. The authors also maintain a tabu-list of forbidden pair-wise exchanges. The initial parameters are computed using 50 random pair-wise exchanges. The algorithm terminates when the final temperature is achieved or when none of the pair-wise exchanges are enabled. The authors have compared their algorithm with NEH, CDS and Osman-Potts SA-based method (62). The authors observed that the proposed algorithm obtained solutions better than the other heuristics like NEH and CDS with comparable computation time. The proposed algorithm also outperforms the other SA-based methods in terms of computational effort.
Since the cooling schedule is one of the factors that influence the performance of SA-based algorithms and it is often rather difficult to accurately determine the optimal cooling schedule, Ishibuchi et al. (39) propose a modified SA-based algorithm which is less sensitive to variations in the cooling schedules. In the traditional SA-based algorithms, the next generation is randomly chosen from the neighborhood of the current generation. The authors modify this generation mechanism by proposing two generation schemes viz. best-move and first-move. In the best-move generation mechanism, the authors select the best candidate in the neighborhood of the current generation based on a biased generation probability mechanism. In the first-move generation mechanism, the authors select the first candidate which is better than the existing solution as the next generation. If no solution in the neighborhood is better than the existing solution, the best among the candidates evaluated is selected. In order to ensure that the computation load of the algorithm remains the same as the traditional SA-based algorithms, the authors increment the number of iterations with a step-size equal to the number of solutions evaluated in the neighborhood of the current solution. The authors use the shift neighborhood mechanism, where the selected part is inserted at a new location, for neighborhood generation since it has been shown (62) that this mechanism is better than interchange neighborhood mechanism in which two parts are swapped. The authors compare their algorithm with the multi-start descent algorithm and tabu search algorithm. In the multi-start descent algorithm, if no candidate in the neighborhood of the current solution is better than the current solution, the algorithm restarts with another initial solution. In the case of tabu search algorithm, a list of shifts already evaluated is maintained and only the candidates in the neighborhood that are not in the tabu list are evaluated. The authors show that the first-move strategy yields better results compared to the
best-move strategy and the proposed heuristic algorithms outperform the multi-start descent and tabu search algorithms for higher number of jobs.

Chakravarthy et al. (13) have proposed a simulated annealing based heuristic for the $m$-machine flow-shop scheduling problem with bi-criteria minimization of makespan and maximum tardiness. They have used three different heuristics to generate three schedules and chose the best among them to be the initial seed sequence for the simulated annealing heuristic algorithm. The three different heuristics used are EDD (Earliest Due Date), LSS (Least Static Slack) and NEH algorithm. They use adjacent interchange method for the neighborhood search in SA. Based on random numbers, the interchange is carried out either to the left or the right of the chosen position.

2.3.2.6 Tabu Search

Tabu Search (TS) is a robust local search based optimization method that begins with an initial solution and successively moves to the best solution in the neighborhood of the current solution while maintaining a list of attributes of solutions to prevent the algorithm from visiting solutions already examined. In addition to an initial solution, a neighborhood generation mechanism, a tabu list and stopping criteria, TS methods also include an aspiration criterion, an intensification scheme and a diversification scheme. A move that is prohibited by the tabu list is accepted if the aspiration criterion is satisfied. The performance of TS approach depends largely on the balance between the intensification and the diversification schemes (25) (65).

Taillard (90) has shown that random pair wise swapping is computationally more expensive compared to random insertion method. Taillard has also shown experimentally that random pair wise swapping does not yield a better convergence to the
optimal solution than the random insertion method. Nowicki et al. (57) propose a fast tabu search algorithm with reduced neighborhood search using a modified NEH algorithm to obtain the initial solution. The authors use block properties to explore the different sequences. By virtue of these properties, the authors were able to eliminate a considerable number of moves thereby reducing the search. The authors also employ the back jump approach in which if there is no change in the solution for a specific number of iterations, the algorithm restarts using the current best solution to create neighboring solutions. This is known as the diversification scheme in TS terminology. The authors have shown that their algorithm yields better results than Taillard (90), Ogbu and Smith (58), and Osman and Potts (62) algorithms with comparable computational expense.

Ben-Daya et al. (7) propose a tabu search heuristic with a mechanism of combining the intensification and diversification schemes that improves the performance of the TS approach. The authors use NEH heuristic to generate the initial solution for their TS approach. A combination of random pair wise swapping, random insertion and block insertion techniques were used for the neighborhood generation process. The first candidate sequence that improved the makespan was selected for the next iteration. A tabu list with capacity seven was used in this approach. The authors construct a frequency matrix that records the number of times each job is located in a particular position among all the examined sequences. A sequence constructed using this matrix is used to randomly restart the neighborhood search process. The authors have compared their algorithm with the TS approach of Taillard (90) and SA-based approach of Ogbu and Smith (58). The authors observed that for the benchmark problems (91), their approach outperformed the SA-based approaches with a marginal increase in computational time. The performance of their approach was comparable
to Taillard’s TS approach (90) at a much lesser computational expense due to the smaller neighborhood space examined.

Grabowski et al. (27) propose a very fast tabu search algorithm for the permutation flow-shop problem to minimize the makespan. The authors evaluate the solutions in the neighborhood space by computing a lower bound on the makespan instead of computing the actual makespan. The authors also propose a dynamic tabu list where the oldest element in the list is deleted if no favorable moves are obtained in a particular iteration and the iteration is continued. Moreover, the size of the list is also determined based on the current iteration. The authors also introduce a perturbation mechanism to diversify the search space in order to improve the performance of the neighborhood search. The perturbation mechanism is activated when there is no improvement in the solution for a fixed number of iterations. This number is chosen experimentally. The authors compared their algorithm with those of Nowicki et al. (57), Taillard (90), Ogbu and Smith (58), Osman and Potts (62) and Reeves (72) using the benchmark problems proposed in Taillard (91). The authors have found that the performance of their algorithm improved with increasing problem size when compared to Nowicki et al. (57). They also observed that their algorithm was much faster than the existing algorithms in yielding comparable solutions.

Brucker et al. (9) have proposed a tabu search algorithm for cyclic machine scheduling problems with specific application to job-shops. The authors use the disjunctive graph representation of the job-shop problem and propose a neighborhood search mechanism based on the weights of the arc which are computed in terms of the precedence of operations. The authors use the best-fit strategy for selection of a neighbor. According to this strategy, the best solution in the neighborhood of the
current solution is selected for the next iteration. The authors assume unlimited buffer capacity.

2.3.2.7 Hybrid Metaheuristics

Talbi (92) gives a detailed classification framework and taxonomy of hybrid metaheuristics. The author also provides an extensive bibliography of literature in this area. Wang et al. (99; 100) propose a hybrid heuristic based on SA and GA approaches in order to avoid the premature convergence of GA-based approaches and improve neighbor search ability of the heuristic. NEH heuristic is employed to generate an initial solution and other solutions are generated randomly to form the initial population of solutions. The authors divide the population of solutions into subpopulations and apply multiple crossover operations to these subpopulations in order to improve the exploring potential of traditional GA-based approaches. The authors also replace the mutation operation with SA’s metropolis sample process (41). The authors have shown that even though the hybrid heuristic is computationally more expensive compared to NEH and traditional GA-based heuristics, it yields better solutions.

Noorul Haq et al. (56) propose a hybrid heuristic based on Ant System (AS) and GA approaches to minimize the makespan for a flow-shop scheduling problem. The solution obtained through the AS technique (20) is fed into the GA-based approach (24) to further improve the solution. The authors show that the hybridization of metaheuristic techniques yields better results when compared to pure metaheuristic techniques.

Among the improvement heuristics for flow-shop scheduling, SA-based and TS-based algorithms have yielded good results for a wide range of scheduling problems.
Low et al. (44) propose a heuristic algorithm that combines the benefits of SA-based algorithms and TS-based algorithms. The authors use a modified NEH algorithm to generate the initial solution for the algorithm. The authors introduce a restarting mechanism into the traditional SA-based algorithm such that if there is no improvement in the solution over a specified number of iterations, a new initial solution is obtained using an auxiliary matrix which is based on the solutions explored so far and the algorithm is repeated. The authors also introduce an additional stopping criterion. The algorithm terminates when the final temperature is achieved or a specified number of iterations are executed. The authors compared their modified SA-based algorithm with other algorithms like the Ogbu and Smith algorithm (58) for the benchmark problems proposed by Taillard (91). They observed that although the difference in the algorithms is marginal for smaller problem sizes, their algorithm yields better results as the problem size increases.

Nearchou (55) proposes a hybrid SA-based algorithm which incorporates features from GA-based and local search heuristics. The algorithm works with a randomly generated population of solutions and generates neighbors using a random exchange perturbation scheme. The author also introduces an iterative hill-climbing procedure to avoid local-minima at each step in the iteration. The author has shown that the results obtained using this algorithm on Taillard’s benchmark problems are comparable to those obtained using Ogbu and Smith (58, 59), Osman and Potts (61), and Reeves (72).

Solimanpur et al. (83) propose a tabu search (EXTS) algorithm based on neural networks for permutation flow-shop scheduling problem. The authors use the modified NEH algorithm proposed by Taillard (90) to generate the initial solution. The authors use the insertion mechanism to generate the neighborhood structure since it was found
to be more effective than a random swap mechanism (90). In order to reduce the number of solutions evaluated in the neighborhood, the block properties proposed by Nowicki et al. (57) have been used. Each move in the neighborhood is represented as a neuron thus generating a neural network. The neuron records the quality of the move called the gain effect and its history for the past $s$ iterations (where $s$ is the number of iterations for which a solution is maintained in the tabu list following which it is removed from the list) called the tabu effect. The proposed algorithm assigns varying degrees of tabu to each move already undertaken as opposed to the traditional tabu search mechanisms where a move is absolutely tabu or not tabu at all. This enables the algorithm to reduce the effect of making a previously tabu move non-tabu. The tabu effect of each neuron is decreased exponentially thereby preventing a drastic impact on the search. The algorithm also incorporates the back-jump mechanism as its diversification scheme. The authors compared their algorithm with the BF-TS heuristic (7) on the set of benchmark problems (91) and observed an average reduction of up to 77% in computation time over BF-TS algorithm while achieving lesser makespan than the BF-TS algorithm.

### 2.3.3 Rescheduling

Manufacturing systems with a diverse range of products and processes are seldom static in nature. Various disrupting factors like demand fluctuations, machine breakdowns, material shortages due to inclement weather or lack of transportation, unavailability of tools and shortage of manpower are not uncommon. Under these circumstances, it might not be optimal or even feasible to execute the originally computed schedule thus making it mandatory to generate a new optimal schedule. This is called reactive scheduling or dynamic scheduling.
There are two main strategies (98) employed for rescheduling viz. *dynamic scheduling* where no prior schedule is generated and decisions are made when a machine becomes available based on dispatching rules and the two-stage *predictive/reactive scheduling* (50; 96) where a robust schedule is generated in an attempt to incorporate most of the possible disruptions and the existing schedule is modified when disruptions occur. Some of the rescheduling policies used for triggering the rescheduling mechanism include *periodic* where the schedule is generated when a fixed time unit lapses, *event-driven* where an event like the number of parts in the queue to be scheduled triggers the rescheduling mechanism (97) and *hybrid* policies. The rescheduling mechanisms may be broadly classified into three categories viz. *right-shift scheduling*, *partial rescheduling*, and *complete rescheduling*. More details on the various rescheduling mechanisms can be found in (98).

Bean et al. (5) propose an heuristic approach for rescheduling where the existing schedule is continued until disruptions occur and in the event of disruptions, a new schedule is computed using dispatching rules like earliest due date (EDD), and shortest processing time (SPT) to match up to the original schedule although with the assumptions that jobs may be preempted and that the inter-disruption times are sufficiently large to allow the new schedule to match up to the original schedule before the next disruption occurs.

Wu et al. (103) and Abumaizar and Svesta (1) propose a 'generalized approach' considering machine breakdown as the only disrupting factor. The authors propose the use of machine break-downs to model most disrupting factors since the net effect of most disruptions is the same - delay. Shafaei et al. (79; 80; 81) propose an integrated framework to generate robust schedules with uncertainties like variation in processing times and machine breakdowns. The authors generate a predictive
schedule and use a combination of the shortest processing time (SPT) and critical ratio (CR) dispatching rules to modify the existing schedule when disruptions occur. Wu et al. (102) propose a rescheduling method based on scheduling graphs for job-shops. A scheduling graph is another representation of a Gantt chart which shows the relationship between operations and machines pictorially. The authors classify the impact of a disruption into two categories viz. time effect where the start time and the end time of operations are affected and relationship effect where the order of operations are affected. Only the nodes of the scheduling graph that are affected by the disruption are recomputed.

Bierwirth et al. (8) propose a genetic algorithm (GA)-based rescheduling algorithm for a dynamic job-shop with intermediate buffers and stochastic part arrivals. The authors decompose the nondeterministic problem into a series of deterministic problems and use GA to arrive at a solution for each of the deterministic problems. The authors improve the performance of the algorithm by having a tunable search space. Rescheduling is initiated only when a job arrives and the current population in the GA method is modified to accommodate the new arrival. Yamamoto et al. (105) have shown that rescheduling using Baker’s (4) active schedule generation mechanism which is based on the branch and bound technique yields better results compared to dispatching procedures and fixed sequencing procedures for handling disruptions due to machine breakdowns. The authors assume that parts are held in pallets once an operation is completed on a machine in order to be transported to the next machine. Cowling et al. (16) propose a framework to utilize real time information for effective dynamic scheduling. The authors propose two decision parameters viz. utility which determines the benefit of rescheduling and stability which determines the drawback
of continuing with the existing schedule. The authors use a combination of these two parameters to decide between rescheduling and repairing the existing schedule.

2.3.4 Cyclic Systems

Hall et al. (30) look into the complexities involved in cyclic shop scheduling for the 2-machine and the 3-machine problems. Hanen (31) shows that the cyclic job-shop sequencing problem is NP-hard and also proposes branch and bound algorithm for cyclic job-shops. Chauvet et al. (14) propose an event-graph based construction algorithm for optimizing cyclic production systems with infinite buffer capacity assuming that the schedule for the bottleneck machine is given. Event graphs are specialized Petri nets with each place having only one input and one output transition with unit weights. The algorithm determines the schedule for the remaining machines. Sauer (77) proposes a heuristic algorithm based on perturbation analysis for stochastic cyclic manufacturing systems with buffers to obtain a schedule for a given cycle time. Serafini et al. (78) propose a mathematical model based on network flows for periodic scheduling problems with emphasis on job-shop problems. The authors also propose an implicit enumeration algorithm to schedule events within the given time constraints. McCormick et al. (49) model a cyclic flow shop with buffers by considering buffers to be resources with variable processing times.

Matsuo (46) shows that there exists polynomial time algorithms to solve the two-machine flow-shop scheduling problem with no buffers which is equivalent to no-wait flow shops. Song et al. (85, 84) have shown that a cyclic job shop with no buffers has an optimal schedule that remains the same over each minimum part set (MPS). The authors also propose a mixed integer programming model using petri nets to solve the cyclic job shop problem with no buffers. Hsu et al. (36) propose a GA-based
algorithm for cyclic scheduling with the objective of minimizing the work-in-process while generating a schedule to match the cycle time which is computed as the load on the bottleneck machine. However, the authors have demonstrated their algorithm only on problems with comparatively smaller number of operations.
Chapter 3: Problem Statement

In order maximize profits, manufacturing companies often strive to utilize the machines to the fullest possible extent while producing parts as quickly as possible. Hence it becomes imperative to find the best possible schedule which maximizes the throughput of the system. The previous chapter covered an extensive literature review of the existing research addressing the permutation flow-shop scheduling problem. This chapter looks into some of the limitations of the existing approaches and also describes the various assumptions made in this study.

3.1 Drawbacks/Limitations of existing approaches

From the aforementioned literature survey, it can be observed that most of the research has been in the area of developing heuristics approaches for the non-cyclic flow-shop problem with the aim of satisfying criteria such as minimization of makespan, flow time or in some cases both among others. Moreover, most of the systems that have been studied assume limited or infinite buffer in between machines which results in an increased inventory. The following sections look into some of the important drawbacks/limitations of existing heuristic approaches.

3.1.1 Period versus Makespan

It is worth recollecting at this moment that makespan is the time from the beginning of the processing of the first part on the first machine to the completion of the last part on the last machine. It is essentially the time taken to finish processing a set of parts as shown in Figure 3.1. In non-cyclic manufacturing systems, makespan is definitely an important criterion to maximize the throughput of the system. In
the case of cyclic systems, makespan is not as important a criterion for maximizing throughput as the period since the objective of cyclic systems is to primarily produce a set of parts repeatedly. Period of a cyclic manufacturing is the time it takes for any machine to completely process a single set of parts at steady-state when the system has stabilized itself after the initial stages of processing as shown in Figure 3.1. The differences between makespan and period can be illustrated using an example shown in Table 3.1. The example consists of 5 parts and 5 machines. There are 4! or 24 possible cyclic permutation sequences. All possible cyclic permutation sequences were examined and the completion time of each operation for the sequence which gives

Table 3.1: Processing Times for a system of 5 parts and 5 machines

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>M2</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>M3</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>M4</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>M5</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
minimum makespan is shown in Table 3.2. Table 3.3 shows the completion time of the operations for the sequence which minimizes the period. It can be seen that the sequence that minimizes the period does not minimize the makespan and vice-versa. Table 3.4 shows the completion time of operations for another sequence which min-

<table>
<thead>
<tr>
<th>M1</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2</td>
<td>10</td>
<td>16</td>
<td>20</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>M3</td>
<td>12</td>
<td>20</td>
<td>25</td>
<td>31</td>
<td>39</td>
</tr>
<tr>
<td>M4</td>
<td>17</td>
<td>22</td>
<td>31</td>
<td>36</td>
<td>45</td>
</tr>
<tr>
<td>M5</td>
<td>18</td>
<td>31</td>
<td>36</td>
<td>45</td>
<td>48</td>
</tr>
</tbody>
</table>

**Table 3.2:** Completion Times for sequence 1-2-3-5-4

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>5</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>M2</td>
<td>10</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>M3</td>
<td>12</td>
<td>20</td>
<td>25</td>
</tr>
<tr>
<td>M4</td>
<td>17</td>
<td>22</td>
<td>31</td>
</tr>
<tr>
<td>M5</td>
<td>18</td>
<td>31</td>
<td>36</td>
</tr>
</tbody>
</table>

Period   | 33  |
Makespan | 48  |

**Table 3.3:** Completion Times for sequence 1-2-4-5-3

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P4</th>
<th>P5</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>3</td>
<td>10</td>
<td>18</td>
<td>27</td>
</tr>
<tr>
<td>M2</td>
<td>8</td>
<td>14</td>
<td>23</td>
<td>32</td>
</tr>
<tr>
<td>M3</td>
<td>13</td>
<td>18</td>
<td>24</td>
<td>33</td>
</tr>
<tr>
<td>M4</td>
<td>18</td>
<td>20</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td>M5</td>
<td>19</td>
<td>29</td>
<td>33</td>
<td>44</td>
</tr>
</tbody>
</table>

Period   | 32  |
Makespan | 50  |

imizes the period. It can be seen that the makespan is different from that shown in Table 3.3. Thus, it can be seen that in order to improve the utilization of a cyclic manufacturing system, it is imperative to minimize the period of the system. In this work, period has been used as the performance measure to determine the optimal schedule.
Table 3.4: Completion Times for sequence 1-4-2-5-3

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P4</th>
<th>P2</th>
<th>P5</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>3</td>
<td>11</td>
<td>18</td>
<td>27</td>
<td>32</td>
</tr>
<tr>
<td>M2</td>
<td>8</td>
<td>16</td>
<td>22</td>
<td>32</td>
<td>34</td>
</tr>
<tr>
<td>M3</td>
<td>14</td>
<td>19</td>
<td>26</td>
<td>33</td>
<td>39</td>
</tr>
<tr>
<td>M4</td>
<td>19</td>
<td>25</td>
<td>28</td>
<td>37</td>
<td>46</td>
</tr>
<tr>
<td>M5</td>
<td>20</td>
<td>28</td>
<td>37</td>
<td>46</td>
<td>51</td>
</tr>
<tr>
<td>Period</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td>Makespan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>51</td>
</tr>
</tbody>
</table>

3.1.2 Buffer versus No Buffer

Among the cyclic systems that have been studied, most of the literature \([14, 46, 74]\) assumes the presence of at least finite if not infinite buffer space in between machines where parts can be unloaded once the machines finish processing time. This frees the machine to process other parts. Buffers in the system result in an increase in work-in-process inventory which incurs costs in terms of storage space and raw material being held up. This work focuses on cyclic manufacturing systems with no intermediate buffers. Under these conditions, the maximum work-in-process inventory possible at any given time would be the total number of machines.

3.1.3 Static versus Dynamic

Manufacturing systems are seldom static in nature and disturbances in the form of variation in processing times, and change in part types to be produced, often necessitate alterations to the existing schedule. This area has often been categorized as dynamic scheduling or rescheduling. Most of the literature in this area is aimed at either associating probabilistic values with these uncertainties and incorporating them in the original schedule \([79, 80, 81]\) or responding to these disturbances by
reconstructing the schedule from the instant of time at which the disturbance occurs \( (1, 8, 96) \). These approaches seldom make use of the actual system information which is often readily available. This work makes use of the actual information available in the system, such as the current idle times in the machine and the current start times of the various operations, to generate a schedule that improves the utilization of the machine in light of the disturbances.

3.2 Objectives and Significant Contributions

It is evident from the extensive literature review that cyclic flow-shops have received very little attention among researchers in this area. When such a system is considered without intermediate buffers, scheduling equations can be derived to ascertain the exact start times of various operations of the parts in the system. These equations also reveal the time for which the machines are idle either waiting for a part from the preceding machine or waiting for the next machine to become available. These equations have a particular structure involving maximization terms which can be modeled using Max-Plus algebra. A new mathematical formulation of the cyclic permutation flow-shop problem using Max-Plus algebra has been developed in this work. By representing the scheduling equations in Max-Plus algebra format, regular linear algebra concepts like eigenvalues and eigenvectors can be applied to the system of scheduling equations and this helps in the computation of the period of the underlying manufacturing system represented by these system of equations.

In a flow-shop system with no buffers, the machine with the maximum load which is often referred to in scheduling parlance as the bottleneck machine or critical machine plays a significant role in determining the throughput of the system. This is the theoretical lower bound on the period of the system. Hence, theoretically the best
possible schedule is the one which has a period equal to the load on the bottleneck machine\(^{(30)}\). Thus, it is imperative to reduce the idle times of the bottleneck machine in order to reduce the overall throughput of the system. A mechanism that uses the system information available to identify problem areas on the bottleneck machine which if addressed can potentially improve the throughput is also presented in this work.

Using the aforementioned mathematical formulation and the mechanism to identify potential areas of improving the throughput of the system, heuristic algorithms have been developed to obtain a schedule that is as close as possible to the theoretically best schedule. The commonly used perturbation mechanisms of move and swap have been employed to search the solution space. The impact of a good initial solution on the quality of the final schedule is also investigated. In order to reduce the computational complexity of the search mechanism, an efficient methodology that reduces the total complexity by a factor of \(n\) where \(n\) is the number of parts in the system has been developed. The effect of batch size on the schedule of the system both in terms of proximity to the best-possible solution and the computational complexity has also been investigated.

### 3.3 Assumptions

In addressing the aforementioned problem, a few assumptions have been made with regards to the different aspects of the problem. These assumptions have been listed below.

- There are no intermediate buffers between machines.
- Each part is processed on a machine only once.
• Parts cannot be preempted on a machine which implies that once the machine begins processing an operation of a part, it is allowed to complete the operation on the part before switching to the next part.

• Set-up times involved in various operations are independent of the sequence of jobs and are included in the corresponding processing times.

• Machines are never forced to be idle i.e. a part is processed as soon as it becomes available and the corresponding machine is free.

• Each machine can process only one part at any given time.

• Each part can be processed only by one machine at any given time.

In the following chapters, the developed heuristics will be explained in detail and it will be shown that the developed heuristics obtain solutions closer to the theoretical best solution than those obtained by some of the existing heuristics that were compared while incurring a much lower computational expense.
Chapter 4: Max Plus Algebra

Max-plus algebra was introduced by Cunnighame-Green in 1979 (10) in an effort to convert non-linear models of discrete event systems into linear models so that these systems can be solved using conventional linear algebra concepts. A classical reference for max-plus algebra is (3). More information can also be obtained on the internet at (67). A more recent source and perhaps the first book devoted entirely to max-plus algebra and its applications is (33). This gives more information on max-plus algebra and the various tools available. This chapter gives an overview of the notation and concepts involved in max-plus algebra. The notations used and the concepts given here have been adapted from (33).

4.1 Basics

Max-plus algebra consists of two main operators viz. the max operator (maximization) which is denoted by the symbol $\oplus$ and and the plus operator (addition) which is denoted by the symbol $\otimes$. The zero element and the unit element in max-plus algebra are $-\infty$ and 0 respectively and are denoted by $\epsilon$ and $e$ respectively. The max operator is defined as, $x \oplus y = \max(x, y) \forall x, y \in \mathbb{R}_{\text{max}}$ where $\mathbb{R}_{\text{max}}$ denotes the set of real numbers and $\epsilon$, i.e. $\mathbb{R}_{\text{max}} = \mathbb{R} \cup \epsilon$. The plus operator is defined as $x \otimes y = x + y \forall x, y \in \mathbb{R}_{\text{max}}$. It is the relationship between conventional linear algebra and max-plus algebra that allows us to apply linear algebra concepts like eigenvalue, eigenvectors, Cramer’s rule, Cayley-Hamilton theorem to max-plus algebraic models as will be shown later. These operators have the properties of regular algebraic operators namely associativity, distributivity and commutativity.
The set of all matrices in max-plus algebra is denoted by $\mathbb{R}_{\text{max}}^{m \times n}$ where $n$ and $m$ are the number of rows and columns respectively. We use the notation $[A]_{i,j}$ to represent the $(i,j)^{th}$ element of $A$. Matrix multiplication in max-plus algebra is defined as

$$\forall \ A \in \mathbb{R}_{\text{max}}^{l \times m}, \ B \in \mathbb{R}_{\text{max}}^{m \times n}, \ C = A \otimes B$$

where

$$[C]_{i,j} = \bigoplus_{k=1}^{m} ([A]_{i,k} \otimes [B]_{k,j}) \ i = 1,2, \ldots l, \ j = 1,2, \ldots n$$

The power of a matrix in $\mathbb{R}_{\text{max}}^{n \times n}$ is defined as

$$\forall \ A \in \mathbb{R}_{\text{max}}^{n \times n}, \ x \in \mathbb{N}, \ x \neq 0, \ A^{\otimes x} = A^x = A \otimes A \otimes A \otimes \cdots \otimes A \rightarrow x \text{ times}$$

The star operator ($\ast$) on a matrix in $\mathbb{R}_{\text{max}}^{n \times n}$ is defined as

$$A^\ast = I_{n \times n} \oplus A \oplus A^2 \oplus \cdots \oplus A^{n-1}$$

where $I_{n \times n}$ is the identity matrix in $\mathbb{R}_{\text{max}}^{n \times n}$. The trace of the matrix $A \in \mathbb{R}_{\text{max}}^{n \times n}$ is the sum of all the elements along the main diagonal of the matrix $A$ or in max-plus algebra terms,

$$\text{trace}(A) = \bigoplus_{x=1}^{n} ([A]_{x,x})$$

### 4.2 Eigenvalue and Eigenvectors

Just as a system of equations in conventional linear algebra has eigenvalues and corresponding eigenvectors, a system of equations in max-plus algebra also has eigenvalues and corresponding eigenvectors. For every matrix $A$, there exists a corresponding unique directed graph denoted by $G(N, C)$. A directed graph $G(N, C)$ is a set of nodes $N$ connected by a set of unidirectional arcs $C$. An arc from node $x \in N$ to node $y \in N$ exists only if the corresponding value $[A]_{y,x} \neq \epsilon$. A graph $G(N, C)$ is
strongly connected if there exists a series of arcs connecting every pair of nodes in $N$ (86). A matrix $A$ is irreducible if its corresponding directed graph $G(N, C)$ is strongly connected (86). If the equation $Ax = \lambda x$, $A \in \mathbb{R}_{max}^{n \times m}$, $\lambda \in \mathbb{R}$, $x \in \mathbb{R}_{max}^{n \times 1}$ holds, $\lambda$ is called the eigenvalue and $x$ the eigenvector. Note that $x$ is a one-dimensional column vector in $\mathbb{R}_{max}^{n \times 1}$. The system of equations represented by the $A$ matrix has a unique eigenvalue if the $A$ matrix is irreducible (33). There are many algorithms like Karp's algorithm (33), Power algorithm (88) and modified Floyd-Warshall procedure (60) to name a few, that can be used to obtain the eigenvalue and eigenvectors for the system of equations in max-plus algebra.
Chapter 5: Mathematical Formulation

Scheduling equations for a system of \( n \) parts and \( m \) machines can be split into three categories. Let \( s_{j,i} \) and \( c_{j,i} \) represent the start time and completion time respectively of the \( i^{th} \) part on the \( j^{th} \) machine. Let \( p_{j,i} \) represent the processing time of \( i^{th} \) part on \( j^{th} \) machine.

\[
c_{j,i} = s_{j,i} \otimes p_{j,i}, \ i = 1, \cdots, n, \ j = 1, \cdots, m \tag{5.1}
\]

In the case of the first machine, \( s_{1,i} \) depends on \( s_{2,i-1} \) and the completion time of the \((i - 1)^{th}\) part on the first machine as shown in Figure 5.1.

\[
s_{1,i} = c_{1,i-1} \oplus s_{2,i-1}, \ i = 1, \cdots, n \tag{5.2}
\]

Since the first machine is assumed to be never idle, the above equation becomes

\[
s_{1,i} = s_{2,i-1}, \ i = 1, \cdots, n \tag{5.3}
\]

In Equation (5.3), when \( i = 1, \ i - 1 \) refers to the \( n^{th} \) part since we are dealing with cyclic systems. Since \( s_{1,i} \) is equal to \( s_{2,i-1} \), it can be omitted from considerations.

![Figure 5.1: Gantt Chart showing the start times on the first machine](image-url)
The start time \( s_{m,i} \) depends on the completion time of the previous part on machine \( m \) and the completion time of \( i^{th} \) part on the machine \( m-1 \) as shown in Figure 5.2

\[
s_{m,i} = c_{m-1,i} \oplus c_{m,i-1}, \quad i = 1, \ldots, n \quad (5.4)
\]

The start time \( s_{j,i} \) on rest of the machines depends on \( s_{j+1,i-1} \) and the completion time of \( i^{th} \) part on machine \( j-1 \) as shown in Figure 5.3

\[
s_{j,i} = c_{j-1,i} \oplus s_{j+1,i-1}, \quad i = 1, \ldots, n, \quad j = 2, \ldots, m-1 \quad (5.5)
\]

Equations 5.1 through 5.5 can be represented in a more compact form using matrices as

\[
s_i = (A_i \otimes s_i) \oplus (B_i \otimes s_{i-1}), \quad i = 1, \ldots, n \quad (5.6)
\]
Table 5.1: Processing Times

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>M2</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>M3</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>M4</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>M5</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

where \( s_i, A_i, B_i \) are denoted as shown below.

\[
\begin{bmatrix}
  s_{2,i} \\
  s_{3,i} \\
  \vdots \\
  s_{m,i} \\
  c_{m,i}
\end{bmatrix}
, \quad
\begin{bmatrix}
  p_{1,i} & \epsilon & \epsilon & \cdots & \epsilon \\
  \epsilon & \epsilon & \epsilon & \cdots & \epsilon \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  \epsilon & \epsilon & \cdots & \epsilon & \epsilon
\end{bmatrix}
, \quad
\begin{bmatrix}
  \epsilon & \epsilon & \cdots & \epsilon & \epsilon \\
  \epsilon & \epsilon & \cdots & \epsilon & \epsilon \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  \epsilon & \epsilon & \cdots & p_{m,i} & \epsilon
\end{bmatrix}
\]

Consider the example with 5 parts and 5 machines that was shown in Section 3.1.1.

Table 5.1 shows the processing times for the example. For a sequence P1-P2-P3-P5-P4, the start times for part P2 can be constructed as follows.

\[
\begin{bmatrix}
  s_{2,2} \\
  s_{3,2} \\
  s_{4,2} \\
  s_{5,2} \\
  c_{5,2}
\end{bmatrix}
= \begin{bmatrix}
  \epsilon & \epsilon & \epsilon & \epsilon \\
  4 & \epsilon & \epsilon & \epsilon \\
  \epsilon & 4 & \epsilon & \epsilon \\
  \epsilon & \epsilon & 2 & \epsilon \\
  \epsilon & \epsilon & \epsilon & 9
\end{bmatrix}
\otimes
\begin{bmatrix}
  s_{2,2} \\
  s_{3,2} \\
  s_{4,2} \\
  s_{5,2} \\
  c_{5,2}
\end{bmatrix}
\oplus
\begin{bmatrix}
  7 & \epsilon & \epsilon & \epsilon \\
  \epsilon & \epsilon & \epsilon & \epsilon \\
  \epsilon & \epsilon & \epsilon & \epsilon \\
  \epsilon & \epsilon & \epsilon & \epsilon \\
  \epsilon & \epsilon & \epsilon & \epsilon
\end{bmatrix}
\otimes
\begin{bmatrix}
  s_{2,1} \\
  s_{3,1} \\
  s_{4,1} \\
  s_{5,1} \\
  c_{5,1}
\end{bmatrix}
\]

It has been shown in (3) that this reduces to the form shown below.

\[
s_i = E_i \otimes s_j, i = 1, \cdots, n, \quad j = 1, \cdots, n, \quad i \neq j
\]

(5.7)
where $E_i$ is defined as $E_i = A_i^* \otimes B_i$. It is interesting to note that the matrix $E_i$ has the form

$$
E_i = \begin{bmatrix}
p_{1,i} & e & \cdots & e \\
p_{1,i}p_{2,i} & p_{2,i} & \cdots & e \\
p_{1,i}p_{2,i}p_{3,i} & p_{2,i}p_{3,i} & \cdots & e \\
\vdots & \vdots & \ddots & \vdots \\
p_{1,i}\cdots p_{n-1,i} & p_{2,i}\cdots p_{n-1,i} & \cdots & e \\
p_{1,i}\cdots p_{n,i} & p_{2,i}\cdots p_{n,i} & \cdots & p_{n,i}
\end{bmatrix}
$$

where the $p_{j,i}p_{k,i}$ is the same as $p_{j,i} \otimes p_{k,i}$. For the aforementioned example, Equation 5.7 becomes

$$
\begin{bmatrix}
s_{2,2} \\
s_{3,2} \\
s_{4,2} \\
s_{5,2} \\
c_{5,2}
\end{bmatrix} = \begin{bmatrix}
7 & e & e & e \\
11 & 4 & e & e \\
15 & 8 & 4 & e \\
17 & 10 & 6 & 2 \\
26 & 19 & 15 & 11 & 9
\end{bmatrix} \otimes \begin{bmatrix}
s_{2,1} \\
s_{3,1} \\
s_{4,1} \\
s_{5,1} \\
c_{5,1}
\end{bmatrix}
$$

We define $F_{z,y}$ as follows.

$$F_{z,y} = \bigotimes_{x=y}^z E_x$$

Now, if we consider the entire sequence, we can iteratively compute the start times of $k^{th}$ part in the sequence in terms of the first part in the sequence as follows

$$s_k = E_{k-1} \otimes E_{k-2} \otimes \cdots \otimes E_1 \otimes s_1, \ k = 2, \cdots, n$$

or

$$s_k = F_{k-1,1} \otimes s_1, \ k = 2, \cdots, n \quad (5.8)$$
For the partial sequence $P1-P2-P3$, if we are given the start times for $P1$, we can compute the partial matrix as shown below.

$$
\begin{bmatrix}
  s_{2,3} \\
  s_{3,3} \\
  s_{4,3} \\
  s_{5,3} \\
  c_{5,3}
\end{bmatrix}
= 
\begin{bmatrix}
  2 & e & e & e & e \\
  4 & 2 & e & e & e \\
  9 & 7 & 5 & e & e \\
  15 & 13 & 11 & 6 & e \\
  20 & 18 & 16 & 11 & 5
\end{bmatrix}
\otimes
\begin{bmatrix}
  7 & e & e & e & e \\
  11 & 4 & e & e & e \\
  15 & 8 & 4 & e & e \\
  17 & 10 & 6 & 2 & e \\
  26 & 19 & 15 & 11 & 9
\end{bmatrix}
\otimes
\begin{bmatrix}
  s_{2,1} \\
  s_{3,1} \\
  s_{4,1} \\
  s_{5,1} \\
  c_{5,1}
\end{bmatrix}
$$

or more concisely,

$$
\begin{bmatrix}
  s_{2,3} \\
  s_{3,3} \\
  s_{4,3} \\
  s_{5,3} \\
  c_{5,3}
\end{bmatrix}
= 
\begin{bmatrix}
  11 & 4 & e & e & e \\
  15 & 8 & 4 & e & e \\
  20 & 13 & 9 & 5 & e \\
  26 & 19 & 15 & 11 & 9 \\
  31 & 24 & 20 & 16 & 14
\end{bmatrix}
\otimes
\begin{bmatrix}
  s_{2,1} \\
  s_{3,1} \\
  s_{4,1} \\
  s_{5,1} \\
  c_{5,1}
\end{bmatrix}
$$

We denote $F_{n,1}$ which includes matrices for all the $n$ parts in the sequence $S$ as $F$ for convenience purposes. Since we are dealing with cyclic systems, the sequence $S$ is repeated and hence we have the following expression

$$
s_{n+i} = F \otimes s_i, i = 1, 2, \ldots, n$$

(5.9)

The eigenvalue of the matrix $F$ is the period of the cyclic system (3). This equation takes the form shown below for the aforementioned example.

$$
\begin{bmatrix}
  s_{2,1} \\
  s_{3,1} \\
  s_{4,1} \\
  s_{5,1} \\
  c_{5,1}
\end{bmatrix}
= 
\begin{bmatrix}
  31 & 28 & 21 & 17 & 8 \\
  36 & 33 & 26 & 22 & 13 \\
  37 & 34 & 27 & 23 & 16 \\
  43 & 40 & 33 & 30 & 25 \\
  46 & 43 & 36 & 33 & 28
\end{bmatrix}
\otimes
\begin{bmatrix}
  s_{2,1} \\
  s_{3,1} \\
  s_{4,1} \\
  s_{5,1} \\
  c_{5,1}
\end{bmatrix}
$$
The period for the sequence \( P1-P2-P3-P5-P4 \) is 33 which is the eigenvalue of the matrix \( F \).

### 5.1 Efficiency Considerations

It can be seen from the above formulation that this method of computing the period involves \( n \) matrix multiplications for each sequence \( S \) of \( n \) parts. Each multiplication of two matrices takes \( m^2(m - 1) \oplus \) operations and \( m^3 \otimes \) operations (Each cell requires \( m \otimes \) operations and \( m - 1 \oplus \) operations). In other words, the complexity to find \( F \) is \( O(nm^3) \). Furthermore, the complexity to compute the eigenvalue or the period for the system using for example Karp’s algorithm is \( O(nm^2) \). Hence, the total complexity to obtain the period of a system of \( n \) parts with \( m \) machines is given by \( O(nm^3) \).

However, there is a more efficient way to calculate \( F \). By applying equation 5.6 to each column in \( F_{ij} \), we get

\[
F_{i+1,j} = E_{i+1} \otimes F_{ij} = A_{i+1}F_{i+1,j} + B_{i+1}F_{ij} \tag{5.10}
\]

Using the special structure of matrices \( A_i \) and \( B_i \), equation 5.10 can be represented as

\[
[F_{i+1,j}]_{1,k} = ([F_{ij}]_{1,k} \otimes p_{1,i+1}) \oplus [F_{ij}]_{2,k}, \quad k = 1, \ldots, m \tag{5.11}
\]

\[
[F_{i+1,j}]_{l,k} = ([F_{i+1,j}]_{l-1,k} \otimes p_{l,i+1}) \oplus [F_{ij}]_{l+1,k}, \quad k = 1, \ldots, m, \quad l = 2, \ldots, m - 1 \tag{5.12}
\]

\[
[F_{i+1,j}]_{m,k} = ([F_{i+1,j}]_{m-1,k} \otimes p_{m,i+1}), \quad k = 1, \ldots, m \tag{5.13}
\]

We use the \# operator to represent the computation of \( F_{i+1,j} \) using the equations 5.11 through 5.13

\[
F_{i+1,j} = p_{i+1} \# F_{ij} \tag{5.14}
\]
where \( p_{i+1} \) is given by

\[
p_{i+1} = \begin{bmatrix}
p_{1,i+1} \\
p_{2,i+1} \\
\vdots \\
p_{m,i+1}
\end{bmatrix}
\]

The above efficient mechanism is illustrated by the following example.

\[
F_{1.0} = p_1 \# I = \begin{bmatrix}
3 & e & e & e & e \\
5 & e & e & e & e \\
1 & e & e & e & e
\end{bmatrix} = \begin{bmatrix}
3 & e & e & e & e \\
8 & 5 & e & e & e \\
9 & 6 & 1 & e & e
\end{bmatrix}
\]

\[
F_{2.0} = p_2 \# F_{1.0} = \begin{bmatrix}
7 & e & e & e & e \\
4 & 8 & 5 & e & e \\
2 & 14 & 11 & 6 & 5 \\
9 & 15 & 12 & 7 & 6
\end{bmatrix} = \begin{bmatrix}
10 & 7 & e & e & e \\
14 & 11 & 4 & e & e \\
18 & 15 & 8 & 5 & e \\
20 & 17 & 10 & 7 & 2 \\
29 & 26 & 19 & 16 & 11
\end{bmatrix}
\]

The matrix \( F \) for the entire sequence can be iteratively constructed as shown below.

\[
F_{3.0} = p_3 \# F_{2.0}
\]

\[
F_{5.0} = p_5 \# F_{3.0}
\]

\[
F = F_{4.0} = p_4 \# F_{5.0} = \begin{bmatrix}
31 & 28 & 21 & 17 & 8 \\
36 & 33 & 26 & 22 & 13 \\
37 & 34 & 27 & 23 & 16 \\
43 & 40 & 33 & 30 & 25 \\
46 & 43 & 36 & 33 & 28
\end{bmatrix}
\]
It can be seen that this is the same matrix obtained earlier and the eigenvalue of this matrix is 33 which is the period of the system. Thus, the complexity to compute the matrix $F_{i+1,j}$ using equation 5.14 is $O(m^2)$ as opposed to $O(m^3)$ using the full matrix multiplication. The final matrix $F$ shown in equation 5.9 can be computed by repeating the similar computation $n$ times and hence the complexity to find the final matrix $F$ is $O(nm^2)$. 
Chapter 6: Heuristic Algorithm

In this chapter, we first present an overview of the new concepts that have been developed in order to identify potential areas of improving the existing schedule. In the next section, we explain the heuristic algorithms developed in this work.

6.1 Opportunity/Critical Machine

A critical machine is denoted by $m_c$ and is given by

$$m_c = \arg \max_j (\sum_{i=1}^{n} p_{j,i})$$

An opportunity $o = (m_c, i, i_p)$ is defined as a potential target to reduce the period of the system where $i_p$ is called the pivot part and is defined as the part that produces the opportunity. An opportunity exists if either $b_{m_c,i} \neq 0$ or $d_{m_c,i} \neq 0$ is satisfied where $b_{m_c,i}$ and $d_{m_c,i}$ are the times for which the machine $m_c$ is blocked and starved respectively after processing the $i^{th}$ part. The set of all opportunities is represented as $O$. The pivot part $i_p$ is

- the part whose operation on any of the machines following the critical machine $m_c$ causes the critical machine to be blocked. In other words, if $b_{m_c,i} \neq 0$, $\exists i_p = i - 1, i - 2, \ldots, j = m_c, m_c + 1, \ldots, m$ for which $c_{j,i_p} = c_{m_c,i} + b_{m_c,i}$ and $b_{j,i_p} = 0$ as shown in Figure 6.1a (If $i_p \leq 0$, $i_p = i_p + n$) or

- the part whose operation on the machine just before the critical machine $m_c$ causes the critical machine to remain idle. In other words, if $d_{m_c,i} \neq 0$, $\exists i_p = i + 1, j = m_c - 1$ for which $c_{j,i_p} = s_{m_c,i_p}$ and $b_{j,i_p} = 0$ as shown in Figure 6.1b (If $i_p \geq n$, $i_p = i_p - n$)
The algorithm to compute all the opportunities can be summarized as shown by \( \text{Ops}(S_c, m_c, C, P, B, D) \) listed in Appendix A.7 where \( S_c \) is the current sequence, \( m_c \) is the critical machine, and \( C, P, B, D \) are the completion times, processing times, block times and slack times of all operations respectively. It can be seen that the computation involved in identifying a single opportunity is \( O(m) \) since the pivot part and the critical machine can be separated by at most \( m \) levels (machines). The complexity involved in identifying all the opportunities in a sequence of \( n \) parts is given by \( O(nm) \).

### 6.2 Heuristic Algorithm

The central idea behind these algorithms is to begin with an initial solution that is either randomly generated or constructed using one of the heuristics and use a perturbation mechanism to search for better solutions. The perturbation mechanisms used in the developed algorithms to explore the search space are swap and move. The algorithm to choose the best perturbation mechanism based on opportunities is...
BestPerturb($O, S_c$) shown in Appendix A.2. In the algorithm, $O$ is the set of opportunities and $S_c$ represents the current sequence. The algorithm returns the sequence with the best possible location for $i_c$ using either the swap or the move mechanism and its corresponding period. The algorithm to chose the best perturbation mechanism by evaluating all possible alternatives is EvaluateAllPerturb($S_c$) shown in Appendix A.2. The perturbation mechanisms used in the above algorithm are swap and move. The algorithms used to implement the perturbation mechanism are Swap($S_c, i$) and Move($S_c, i$) also listed in the Appendices. In Swap($S_c, i$), $i$ represents the location of the part in $S_c$ that has to be swapped. The algorithm returns the sequence obtained after swapping the best possible two parts. In the case of Move($S_c, i$), $S_c$ represents the current sequence, and $i$ represents the locations of the part that has to be moved. The algorithm returns the new sequence with the best possible location for $i$.

6.2.1 MP_C

Using the NEH approach which has been elaborated in Section 7.1.1.1 a similar construction algorithm was devised using max-plus formulations. As is the case with all construction algorithms, the sequence is constructed iteratively by positioning a part in each step until all parts have been positioned in the sequence.

6.2.2 MP_CSoM

Using the sequence obtained from MP_C approach as the initial sequence, this method iteratively generates new sequences until it converges on an optimal or near-optimal solution by selecting the best of the two mechanisms viz. swap or move during each step. The steps involved in this method are shown in Appendix A.4. The initial solution to the problem is constructed using the MP_C heuristic (Step 0).
The max-plus formulation is used to obtain the period of this initial solution (Step 1). Every location is explored to find the best possible location for the part using $\text{EvaluateAllPerturb}(S_c)$ algorithm (Step 2). This algorithm uses $\text{Swap}(S_c, i)$ and $\text{Move}(S_c, i)$ algorithms to carry out the perturbations.

In the swap mechanism, $(\text{Swap}(S_c, i))$, the location of the $i^{th}$ part is exchanged with other parts to find the location that yields the maximum reduction in period as shown in Appendix A.8. In the move mechanism, the pivot part is inserted at other locations thereby shifting all parts after the selected location by one location. This mechanism can also viewed as a removal-insertion mechanism since the pivot part is removed from its current location and inserted at its new location as seen in Appendix A.3. At each step in the improvement process, the best perturbation mechanism among swap and move that yields the maximum reduction in the period is selected. Once the new solution that reduces the period compared to the previous solution is obtained, the algorithm returns to Step 2 and repeats the process again. The algorithm terminates when no more reduction in the period can be achieved. More details of the supporting algorithms are given in the next section.

However, we found that during the perturbation mechanism, it is not necessary to compute the entire matrix $F$ for each sequence that is examined. Figures 6.2 and 6.3 show the computation required to evaluate all possible locations to move the $i^{th}$ part and all possible candidate parts to swap the $i^{th}$ part with respectively. For e. g. it can be seen in Figure 6.3 that while examining the best possible swap pair for the $i^{th}$ part, once we have the intermediate product matrices in region I, the intermediate product matrices in region II can be directly obtained. Similarly, once we have computed the product matrices in region III, the product matrices in region IV can be directly obtained. Moreover, even the intermediate product matrices within a region can be
obtained by just multiplying the product matrix in the previous step with the matrix for the new part. The number of matrix multiplications (#) to compute the products in the highlighted regions in Figure 6.3 is given by \(2n - 8\) for \(n > 4\). There will also be an additional \(n - 2\) matrix multiplications (#) required to create all of the matrices in Figure 6.2. Hence the total number of operations to create all of these matrices is \((3n - 10)m^2\) for \(n > 4\). The above-mentioned efficient matrix computation has also been incorporated in all of the algorithms that follow.
6.2.3 MP\_OSoM

In this algorithm shown in Appendix A.6 (supporting algorithms have also been listed in the Appendices), we assume a randomly generated sequence as the initial sequence (Step 0). We use the max-plus formulation to obtain the period of this initial solution (Step 1). We then identify the list of opportunities available for this solution using $\text{Ops}(S_c, m_c, C, P, B, D)$ (Step 2). The complexity of this step is $O(nm)$ as shown earlier and there can be at most $n - 1$ opportunities.

We examine all opportunities and identify the best possible location for the pivot part using two perturbation mechanisms viz. swap and move. In the swap mechanism, we exchange the location of the pivot part with other parts and find the location that yields the maximum reduction in period. In the move mechanism, we move the pivot part to other locations thereby shifting all parts after the selected location by one location. This mechanism can also viewed as a removal-insertion mechanism since we remove the pivot part from its current location and insert it at its new location. At each step in the improvement process, we select the best perturbation mechanism among swap and move that yields the maximum reduction in the period (Step 3). The complexity of Step 3 is $O(n^2 m^3)$ as shown in the Appendix A.2. Once we get the new solution that reduces the period compared to the previous solution, we return to Step 3 and repeat the process. It can be seen that the algorithm will eventually terminate when no further reduction in the period can be achieved using the opportunities or there are no opportunities to be evaluated. Since we have no means, at present, of estimating the exact number of times Step 3 will execute, the complexity of the algorithm can be expressed in terms of the complexity of Step 3 which is $O(n^2 m^3)$. 
6.2.4 MP_COSoM

This method is very similar to the previous method except that the initial sequence is obtained using MP_C algorithm instead of being randomly generated. The steps involved in this method are similar to those of the MP_OSoM heuristic shown in Appendix A.5.

6.2.5 MP_PFSoM

In this method, the initial sequence is generated using the PF heuristic (49). The rest of the algorithm is similar to MP_CSoM as shown in Appendix A.4.

6.2.6 MP_PFOSoM

The initial sequence is generated using the PF heuristic (49) in this method. The steps involved in this heuristic are similar to those in MP_OSoM heuristic and is shown in Appendix A.6.
Chapter 7: Experimental Analysis

Chapter 6 explained in detail the heuristic algorithms developed in this work. The relative performance of these heuristic algorithms can be ascertained by comparing the quality of solutions obtained using these heuristic algorithms on the same set of problems. It is also important to compare the performance of these heuristics with those of the existing heuristics. This chapter explains in detail the various existing heuristics that have been implemented. A validation process was also carried out to ensure that these heuristics have been implemented as the original authors intended them to be. The results of the validation process have been given in Section 7.1.3.

7.1 Heuristics Compared

In order to ascertain the performance of the heuristic algorithms based on Max-Plus algebra, a comparative analysis was carried out on the benchmark problems using some of the other heuristics that are available in the literature. Although these heuristics have been devised for flow-shop scheduling problems, none except PF heuristic have yet been applied to the cyclic flow-shop scheduling problem to the best of my knowledge. As mentioned earlier, heuristics can be broadly classified into two main categories viz. construction, and improvement. A brief description of the heuristics compared is given below.

7.1.1 Construction Heuristics

Construction heuristics generate the final sequence of parts by either ordering the parts based on some criterion like total processing times, or generating partial sequences by iteratively placing parts in the sequence. LWKR (Least Work Remaining)
and MWKR (Most Work Remaining) \cite{53}-based construction methods represent the former category of construction heuristics. In these methods, the final sequence of parts is generated by arranging the parts in the increasing (decreasing) order of their total processing times (sum of the processing times on all machines for a given part) respectively. Among the construction heuristics of the latter type, the most popular approaches are that of NEH \cite{54} and PF \cite{49}. These heuristics have been elaborated below.

\subsection{NEH}

NEH heuristic \cite{7,40}, which stands for the first letters of the authors, Nawaz, Enscore and Ham, begins by arranging the parts in the decreasing order of their total processing times. Subsequently, the first two parts in this list are scheduled so that the resulting sequence minimizes partial makespan. For each of the remaining parts in the list, all possible locations in the partial sequence are explored and the best location (the one that minimizes the partial makespan) is chosen thereby preserving the relative order of parts. When the final part in the list is thus inserted into the partial sequence, the resulting sequence is the final sequence of parts which minimizes the total makespan. The complexity of this methodology is given by $O(n^3 m + n^2 m)$ where $n$ is the number of parts and $m$ is the number of machines. Since we are interested in minimizing the period for a cyclic system, NEH heuristic was modified so that the partial sequence which minimizes the partial cycle time or period is selected during each step.
McCormick et al.\cite{49} use the concept of departure times $[D]_{i,j}$ which represents the time at which job $i$ leaves machine $j$ in their Profile Fitting (PF) heuristic. The departure times are calculated as given below (The following equations were obtained from \cite{73}).

\[ [D]_{1,0} = 0 \]

\[ [D]_{1,k} = \sum_{q=1}^{k} p_{i,q}, \quad k = 1, \ldots, m - 1 \]

\[ [D]_{i,0} = [D]_{i-1,1}, \quad i = 2, \ldots, n \]

\[ [D]_{i,k} = \max([D]_{i,k-1} + p_{i,k}, [D]_{i-1,k+1}), \quad i = 2\ldots n, \quad k = 1\ldots m - 1 \]

\[ [D]_{i,m} = [D]_{i,m-1} + p_{i,m}, \quad i = 1\ldots n \]

where $p_{i,j}$ is the processing time of part $i$ on machine $j$. The part which has the lowest total processing time is selected as the first part in the sequence. For each of the remaining positions, the departure times are calculated for each available part and based on the departure times, the value of the following expression is computed for each available part on each machine.

\[ [D]_{j,k} - [D]_{i,k} - p_{j,k}, \quad i = 1\ldots n, \quad j = 1\ldots n, \quad i \neq j, \quad k = 1\ldots m \]

This expression represents the time for which machine $k$ remains idle (blocked or starved) when part $j$ is placed after part $i$ in the sequence. The part with the smallest sum of the above expression for all $k$ is selected as the next part in the partial sequence. This process is repeated until all parts are placed in the sequence.
7.1.2 Improvement Heuristics

The improvement heuristics being compared in this work can be classified as previously mentioned into Genetic Algorithm-based, Simulated Annealing-based, Tabu Search-based and Ant Colony Optimization-based. The heuristics that have been incorporated in this comparison have been detailed below.

7.1.2.1 CGA

Chen et al. (15) propose a GA-based approach which uses an initial population size of 60. The authors form the initial population using the $m-1$ schedules generated by CDS method and the schedule generated by Dannenbring’s RA method. Using these solutions, the authors generate the remaining population by randomly swapping two positions in each of the schedules until the required population size is achieved. The authors use Goldberg’s PMX operator to produce offspring. The PMX operator is explained below. This procedure was adapted from (15):

1. Select an interval to be swapped from each of the parent chromosomes

2. Ascertain for each part of the region to be swapped in each parent, the corresponding part in the other parent.

3. Interchange the selected elements in both the parents

4. Complete the sequence using the mappings devised in Step 2.

The part in a location within the region to be swapped on a parent maps to the corresponding part in the same location on the other parent. Following a swap of regions, duplicate parts are replaced by the parts they are mapped to. The authors assume a crossover rate of 1.0 and a mutation rate of 0.0 which essentially means that
crossover is the only operator considered. The parents for reproduction are selected using a selection probability and the offsprings replace the parents in the population. The termination criterion used is the number of generations which is set to 20.

7.1.2.2 WGA

Wang et al. (99) also propose a GA-based heuristic for flow-shop scheduling using the sequence obtained by the NEH heuristic as one solution of the initial population. The remaining solutions are randomly generated. They consider a population size of 40 and assume $n \times m$ generations as the termination criterion. The initial population is divided into groups. Each group is subjected to a combination of crossover and mutation operator. The crossover operators employed include PMX, and Linear order-based. The mutation operators include SWAP, INSERT, and INV. The crossover and mutation rates are assumed to be 1.0 and 0.05 respectively.

7.1.2.3 HAMC

Ravindran et al. (71) propose a heuristic they call HAMC1 using an initial seed sequence. More sequences are obtained by swapping pairs of parts in the seed sequence. For example, in the first step, the first and second parts are swapped. In the second step, the first and third parts are swapped and so on. The periods for each of the sequences are recorded and the best sequence becomes the seed sequence for the next iteration. The number of iterations considered varies between 10 and 20. The sequences obtained in all the iterations are compared and the one with the minimum period is chosen as the final sequence.
7.1.2.4 ACO

Rajendran et al. [70] uses the NEH algorithm to obtain an initial seed sequence and computes the final sequences using an ACO-based algorithm. The trail intensities are computed for each part in each position using a persistent rate of 0.75. Based on the trail intensities, parts are added to the partial sequence iteratively until the final sequence is obtained. The trail intensities are updated for all parts after each part has been scheduled. Once the final ant sequence is obtained, a local search is carried out which essentially is an insertion mechanism where each part in all possible locations while preserving the relative positions of other jobs in the sequence. The best sequence obtained is taken as the final optimal sequence.

7.1.2.5 OSA

Osman et al. [62] propose a SA-based heuristic for flow-shop scheduling problem. The maximum number of iterations before termination, the initial temperature, the final temperature and the cooling rate are computed based on the values of $n$ and $m$ where $n$ and $m$ are number of parts and machines respectively. The authors employed the shift-neighborhood mechanism to generate solutions in the neighborhood of the current solution. In the shift-neighborhood mechanism, a part is removed from its current location and inserted at another location to generate a new sequence.

7.1.2.6 ISA

Ishibuchi et al. [39] propose a SA-based heuristic for the flow-shop sequencing problem. The authors also employed the shift-neighborhood mechanism used by Osman et al. [62] to generate solutions in the neighborhood of the current solution.
Although the maximum number of iterations is fixed, the cooling schedules along with the initial and final temperatures are dependent on the values of \( n \) and \( m \) where \( n \) and \( m \) are number of parts and machines respectively. The initial solution is a randomly generated one. The authors have developed two strategies viz. best-move strategy and first-move strategy to improve the convergence. According to the best-move strategy, the next sequence is selected from a randomly generated set of neighborhood solutions. The first solution generated in the neighborhood of the current solution is selected as the next sequence in the first-move strategy. The authors have found that the first-move strategy outperforms the best-move strategy.

7.1.2.7 TS

Taillard (90) proposed a TS-based heuristic with shift-neighborhood search and a tabu list of capacity 7. The tabu list maintains the moves that were executed in the previous 7 iterations. The iteration terminates when the number of iterations reaches its limit.

7.1.3 Validity of Implementation

In order to compare the heuristics effectively, it is imperative to ensure the validity of the implementations of the above heuristics. To this end, each of the above heuristics was executed on the corresponding data sets used by the original authors and the results obtained by my implementation was compared with those obtained by the original authors. This necessitated implementation of a few more heuristics and their validation. The results of this validation process are given below.
7.1.3.1 ACO

The original authors of this heuristic compared the performance of their heuristic on Taillard’s benchmark problems. Figure 7.1 below shows a comparison of the results obtained by[69] and the results obtained by my implementation of the same heuristic algorithm. The values are the percentage increase from the upper bound for these problems specified by Taillard. It can be seen that my implementation of the heuristic yields results that are better than those obtained by the original authors. This difference may be accounted to the randomness in selecting the test cases.

7.1.3.2 CDS

Table 7.1 below gives the specifications of the sample problems used by Campbell et al. (11) to ascertain the performance of their algorithm. 60 problems were generated using the above specifications and my implementation of the heuristic algorithm was executed on these problems. Figure 7.2 below shows the comparison of the results obtained by CDS and my implementation. The values are the average percent deviation from the optimal solution for each category. It can be seen that the results obtained are comparable to those obtained by Campbell et al. The difference in results may be on account of the random generation of problems which implies that the problems may not be exactly the same as those used by the original authors.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Processing Time</th>
<th>Sample Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 4, 5, 6, 7, 8</td>
<td>3</td>
<td>[1 – 99]</td>
<td>20</td>
</tr>
<tr>
<td>3, 4, 5, 6, 7, 8</td>
<td>5</td>
<td>[1 – 99]</td>
<td>20</td>
</tr>
<tr>
<td>3, 4, 5, 6, 7, 8</td>
<td>7</td>
<td>[1 – 99]</td>
<td>20</td>
</tr>
</tbody>
</table>
Figure 7.1: Comparison of results for ACO heuristic
Figure 7.2: Comparison of percentage deviation from optimal for CDS heuristic
7.1.3.3 CGA

Table 7.2 gives the details of the sample problems used by Chen et al. (15). 40 random problems were generated using the above specifications and the heuristic was executed on these problems. Figures 16 and 17 depict a machine-wise graphical comparison of the values obtained by Chen et al. in (15) with the values obtained by my implementation of the same heuristic for 10 and 15 machines respectively. The values in the graph are computed as given below.

\[ \delta = \left[ \mu_{CHEN} \right] / \left[ \mu_{SPIRIT} \right] \]

where \( \mu_{CHEN} \) is the makespan obtained using the CGA algorithm and \( \mu_{SPIRIT} \) is the makespan obtained using the SPIRIT heuristic proposed by Widmer et al. (101). It can be seen from the figures below that the results obtained are comparable within limits taking into account the variation in the processing times for the sample problems generated. Similar results were obtained for other sample problems.

7.1.3.4 HAMC

The authors of this heuristic compared the performance of their heuristic using the benchmark problems given by Taillard (91). Since the authors used the CR heuristic to generate the initial solution, the results for that heuristic was also compared.
Figure 7.3: Comparison of relative makespan for 10 machines for CGA heuristics
Figure 7.4: Comparison of relative makespan for 15 machines for CGA heuristic
Figures 7.5 and 7.6 show the comparison of average makespan values for the two heuristics. It can be seen from the above figures that the implementation of these heuristics produces results similar to those produced by the original authors of these heuristics.

7.1.3.5 ISA/TS

Ishibuchi et al. (39) give the average results of their heuristic for a set of 100 problems. Table 7.3 below gives the details of the sample problems used by Ishibuchi for a comparative analysis. The authors also implement the TS heuristic and give the corresponding results for TS heuristic executed on the same set of sample problems. This was used to validate the implementation of TS heuristic since the original author of the TS heuristic does not give sufficient data to make an effective comparison. It is important to note that due to limitations on computation times, the results given here for my implementation of these heuristics is only for 1000 evaluations as opposed to 10,000, 50,000 and 200,000 evaluations used by Ishibuchi et al. Since the authors also executed the OSA heuristic for the same set of problems and gave their results for ISA heuristics in terms of OSA heuristics, the validation of OSA heuristic was carried out using the results obtained by Ishibuchi et al. Figure 7.7 shows a comparison of the relative makespan values obtained by Ishibuchi and those obtained by our implementation. Figure 7.8 compares the relative makespan values obtained using

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Processing Time</th>
<th>Sample Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10, 20</td>
<td>[1 − 99]</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>10, 20</td>
<td>[1 − 99]</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>10, 20</td>
<td>[1 − 99]</td>
<td>100</td>
</tr>
</tbody>
</table>
Figure 7.5: Comparison of average makespan for HAMC heuristic
Figure 7.6: Comparison of average makespan for CR heuristic
Figure 7.7: Comparison of relative makespan for ISA heuristic
Figure 7.8: Comparison of relative makespans for TS heuristic
Taillard’s TS heuristic. The values in the graph are computed as follows.

\[
\text{relative makespan} = \left[ \frac{\mu_{\text{sample}}}{\mu_{\text{osman}}} \right] \times 100
\]

for first-move strategy with \( K = N/2, c = 5 \) where \( K \) and \( c \) are parameters used in the SA-based heuristic. In order to reduce the computation times for these comparisons, only 1000 evaluations were used. From the above figures, it can be concluded that my implementation of TS heuristic and ISA heuristic is valid.

7.1.3.6 WGA/NEH

The sample problems on which Wang et al. (99) based their results consists of 29 problems contributed to OR Library by Mattfeld, D. C., and Vaessens, R. J. M., of which 21 can be found in Reeves and 8 in Carlier (99). These problems can be downloaded from (6). Figure 7.9 displays the relative errors obtained by Wang et al. and my implementation for NEH heuristic. Figure 7.10 displays the corresponding relative errors obtained Wang et al. and my implementation for Wang’s GA-based heuristic. From the above results, it can be safely concluded that my implementation of the various algorithms produces similar results as those produced by the original authors of the algorithms. This validation is important for the effective comparison of the heuristic algorithms proposed with the existing algorithms.
Figure 7.9: Comparison of relative errors for NEH heuristic
Figure 7.10: Comparison of relative errors for WGA heuristic
Chapter 8: Results

In this chapter, the results of the extensive experimental analysis carried out to compare the relative performances of the various heuristics with respect to the actual quality of the solution and computation time involved have been presented. In the first section, the performance of the various heuristic algorithms on small problems for which the optimal solutions are known have been evaluated. In the next section, the relative performance of the various heuristic algorithms on large problems, the optimal solutions of which are not known, are examined. In Section 8.3, the individual iterations executed by each heuristic are analyzed in order to find a stopping/terminating criterion. Section 8.4 deals with the impact of batch size on the quality of the solution.

Maximum load is the maximum of the sum of processing times of all parts over all machines and is computed as follows.

$$\pi_l = \max_j \sum_{i=1}^{n} p_{j,i} = \sum_{i=1}^{n} p_{m_c,i}$$

where $p_{j,i}$ is the processing time of part $i$ on machine $j$. The corresponding machine is often called the bottleneck machine or critical machine ($m_c$) in scheduling literature. In a cyclic system, this is theoretically the lowest possible period that can be obtained in a system without buffers. The percentage deviation from maximum load is computed as follows.

$$\delta_d = \frac{\pi_h - \pi_l}{\pi_l} * 100$$

where $\delta_d$ is the percentage deviation, $\pi_h$ is the period obtained using the various heuristic approaches and $\pi_l$ denotes the maximum load. The relative performance of the various heuristics have been evaluated based on their $\delta_d$ values. Each category
of problem consists of 10 problem instances and the processing times vary uniformly between [1-100]. The notation used to identify each problem category is $n$-$m$ where $n$ is the number of parts and $m$ is the number of machines.

8.1 Results for Small Problems

In this section, the performance of the various heuristic algorithms have been compared on problems of comparatively smaller size. The optimal solutions can be obtained for these problems using an exhaustive search method.

8.1.1 Deviation from Maximum Load

8.1.1.1 Construction Heuristics

Figure 8.1 depicts the percentage deviation of the period from the maximum load obtained by the various construction heuristics. The last two categories in Figure 8.1 represent two sets of problems with $n$ and $m$ uniformly distributed between [6-9] and [4-8] respectively. Unlike the previous problem categories, the values of $n$ and $m$ need not be the same for these two categories. The values associated with the figures in this chapter have been given in Appendix B. Figure 8.2 shows the range of average $\delta_d$ values obtained for the random heuristic which explores $m$ randomly generated sequences where $m$ is the number of machines for each problem. The $\delta_d$ value of the optimal solution has also been included for illustrative purposes. The heuristics have been arranged in the decreasing order of their $\delta_d$ values for the Rand4-8 problem category. It can be seen from the above figure that NEH and MP_C heuristics yield the best results among the construction heuristics with an average $\delta_d$ value of 3%-4%.
Figure 8.1: Comparison of Construction heuristics for small problems
Figure 8.2: Results for the Random heuristic for small problems
8.1.1.2 Improvement Heuristics

Figure 8.3 shows the results for the improvement heuristics being compared. The heuristics have been arranged in the decreasing order of their $\delta_d$ values for the Rand4-8 problem category. It is evident from the figure that the developed heuristics yield results within less than 50% of the $\delta_d$ values produced by all the other improvement heuristics compared except WGA. It can also be observed that the results obtained are comparable to those obtained by WGA heuristic.

8.1.2 Computation Time

8.1.2.1 Construction Heuristics

Figure 8.4 shows the average computation time required by the construction heuristics for the small problems. PF and MP_C heuristics take negligible amount of time when compared to NEH heuristic. The computation times in all of the following charts are in milliseconds. It shows that MP_C and PF heuristic perform better than NEH in terms of computation time over small problems.

8.1.2.2 Improvement Heuristics

Figures 8.5 through 8.10 depict the average computation time required by the various improvement heuristics for small problems. It can be seen that the developed heuristics take far less time to converge and as seen earlier (Figure 8.3) yield better results in terms of percentage deviation. Although the average computation times of CGA, TS and HAMC heuristics are comparable to those of the developed heuristics, the quality of the solution obtained by these heuristics is not comparable as seen in Figure 8.3. It may also be noted that although WGA heuristic yielded results
Figure 8.3: Comparison of improvement heuristics for small problems
Figure 8.4: Comparison of computation times for small problems
comparable to those obtained by the developed heuristics, the computation time required by WGA heuristic is significantly greater (almost 7-8 times greater) than that of the developed heuristics as is evident from Figures 8.5 through Figure 8.10. The computation time required to obtain the optimal solution have also been included in Figure 8.5 and 8.6. It can be seen that the computation times for the developed heuristics are significantly lower than that required to find the optimal solution.

The average computation time required to obtain the optimal solution for the 8p-8m and the 8p-4m problem categories are 4176.57ms and 4083.20ms respectively. It can be seen that as the problem size increases, the difference between the computation times to obtain the optimal solution and the computation times using the heuristics becomes significant making it worthwhile to use heuristics to obtain solutions that are close to the optimal solution in terms of the period. Thus, it can be concluded that the developed heuristics yield significantly better results at a much lower computational expense. Hence, even though the optimal solutions for these problems can be obtained through an exhaustive search, the developed heuristics can be used to obtain near-optimal solutions in much less time. Among the construction heuristics, MP_ C yields results comparable to those obtained by NEH heuristic at a significantly lower computational expense. The improvement heuristics developed also fare better than the existing improvement heuristics both in terms of the quality of the solution and the computational expense.

8.2 Results for Taillard’s Benchmark Problems

In this section, the performance of the various heuristic algorithms have been examined over problems of larger size. These problems are considered benchmark problems for scheduling problems and were proposed by Taillard (91). Due to the
Figure 8.5: Comparison of computation time for 6p, 6m
Figure 8.6: Comparison of computation time for 7p, 7m
Figure 8.7: Comparison of computation time for 8p, 8m
Figure 8.8: Comparison of computation time for 8p, 4m
Figure 8.9: Comparison of computation time for Rand6-8
Figure 8.10: Comparison of computation time for Rand4-8
high computational complexities involved, the optimal solution to these problems have not yet been found.

8.2.1 Deviation from Maximum Load

8.2.1.1 Construction Heuristics

Figure 8.11 shows the $\delta_d$ values for the construction heuristics over Taillard’s benchmark problems. The heuristics have been arranged based on the descending order of the respective $\delta_d$ values for the 100-20 problem category. It can be seen that MP_C and NEH heuristics yield comparable results. Among all construction heuristics, the PF heuristic has the lowest $\delta_d$ value for most of the problem categories. It can also be noted that Random, MWKR and LWKR yield very poor results in comparison to the other construction heuristics. Figure 8.12 shows the range of average $\delta_d$ values obtained for the random heuristic which explores $m$ randomly generated sequences where $m$ is the number of machines for each problem.

8.2.1.2 Improvement Heuristics

Figure 8.13 compares the developed heuristics with the other existing improvement heuristics. The developed heuristics are represented by vertical bars that are hatched for each problem category and are placed at the end of the other heuristics. The heuristics have been arranged in the decreasing order of the respective $\delta_d$ values for the 100-20 problem category. It can be observed that the developed heuristics perform better than the other improvement heuristics in terms of the quality of the solution obtained. Among the developed heuristics, the MP_PFOSoM heuristic yields the best results with the lowest percentage deviation from the maximum load. It can
Figure 8.11: Comparison of Construction heuristics based on period for Taillard’s problems
Figure 8.12: Results for the Random heuristic for Taillard’s problems
be seen from Figure 8.13 that this heuristic performs the best consistently in all problem categories. Thus, it can be concluded that the developed construction and improvement heuristics outperform the existing heuristics in terms of obtaining a solution that is closest to the theoretical lower bound of the period.

8.2.2 Computation Time

8.2.2.1 Construction Heuristics

Figure 8.14 shows the computation times required by the various construction heuristics for the benchmark problems proposed by Taillard. It can be observed that the computation times for PF heuristic is significantly lower than the other two heuristics. Random, MWKR and LWKR heuristics have been omitted from the figure since the percentage deviations obtained by these heuristics was significantly higher than those obtained by PF, NEH, and MP_C heuristics.

8.2.2.2 Improvement Heuristics

The computation times involved for the improvement heuristics are shown in Figure 8.15 through Figure 8.23. Each figure shows the computation time required by the heuristics for a problem category. It can be seen that MP_COSoM and MP_PFOSoM require significantly lower computational expense when compared to other heuristics. It can also be seen that even though the percentage deviations obtained by MP_OSoM heuristic were comparable to those obtained by MP_COSoM and MP_PFOSoM heuristics, the computation times are higher. This can be accounted for by the fact that MP_OSoM heuristic begins with a randomly generated solution whereas MP_COSoM and MP_PFOSoM heuristics use solutions constructed...
Figure 8.13: Comparison of Improvement heuristics based on period for Taillard’s Problems
Figure 8.14: Comparison of computation time for Construction heuristics.
using MP.C and PF heuristics respectively. As in the case of small problems, although the computation times for CGA, HAMC, and TS heuristics are lower than most of the other heuristics, the percentage deviations obtained by these heuristics are significantly higher than those obtained by the other heuristics as seen in Figure 8.13. It can be seen from the above figures that the computation times of MP_COSoM, MP_OSoM and MP_PFOSoM heuristics are lower than MP_CSoM and MP_PFSoM for all the problem categories. This shows that the concept of opportunities is a more intelligent mechanism to search for better solutions. Thus, it can be concluded that for large problems, PF heuristics is the best construction heuristics among the construction heuristics compared. It can also be concluded that the developed heuristics yield solutions that are much closer to the theoretical lower bound than the existing improvement heuristics with comparable if not lower computational times. Among the developed heuristics, MP_PFOSoM yielded best results achieving the lowest percentage deviations with considerably lower computation times.

### 8.2.3 Number of Swaps and Moves

Figure 8.24 and Figure 8.25 show the number of swaps and moves, respectively, made by the developed improvement heuristics before converging to the final solution. It can be observed that when the initial solution is constructed using either MP.C or PF heuristics, the number of swaps or moves required before convergence is significantly lower when compared to the heuristics that use a randomly generated sequence as the initial solution. It can also be seen that the MP_COSoM and MP_PFOSoM heuristics also require lower number of steps to converge to the final solution which validates the claim that opportunities help in identifying potential areas for reducing the period. Among these two heuristics, MP_PFOSoM requires lesser number of steps
Figure 8.15: Comparison of computation time of Improvement heuristics for 20p, 5m
Figure 8.16: Comparison of computation time of Improvement heuristics for 50p, 5m
Figure 8.17: Comparison of computation time of Improvement heuristics for 100p, 5m
Figure 8.18: Comparison of computation time of Improvement heuristics for 20p, 10m
Figure 8.19: Comparison of computation time of Improvement heuristics for 50p, 10m
Figure 8.20: Comparison of computation time of Improvement heuristics for 100p, 10m
Figure 8.21: Comparison of computation time of Improvement heuristics for 20p, 20m
Figure 8.22: Comparison of computation time of Improvement heuristics for 50p, 20m
Figure 8.23: Comparison of computation time of Improvement heuristics for 100p, 20m
to converge which confirms the results obtained in Figure 8.11 that the PF heuristic generates a better initial solution than MP_C heuristic.

8.3 Early Termination

In this section, the individual iterations of heuristics have been examined and analyzed in order to explore the possibility of terminating the developed heuristics after a prescribed number of iterations beyond which no significant improvement can be achieved in the solution. Figure 8.26 shows the progress of period obtained using MP_COSoM heuristic for each of the 10 problems in the 20p-20m problem category. Figure 8.27 shows the percentage reduction beginning with the second iteration using the MP_COSoM heuristic for each of the 10 problems in the 20p-20m problem category. It can be seen from Figures 8.26 and 8.27 that the improvement from the solution obtained using MP_C heuristic is considerable in the first iteration and thereafter the improvements are within a comparable range of 0.2% to 1.3%. However, no distinct termination point could be identified since the improvements in each iteration were independent of the improvements in the previous iteration.

Figures 8.28 and 8.29 show the progress of period using MP_PFOSoM heuristics for the 20p, 20m problem category. It may be observed that the number of iterations in these figures is lower than those in Figures 8.26 and 8.27 which further validates the results in the earlier sections that the PF heuristics yields a better initial solution than MP_C heuristic. It can also be seen that the improvement in the first iteration (0.2% - 3%) is not as drastic as in the case of MP_COSoM heuristics (5% - 9%) which again validates the claims in earlier sections that the PF heuristic yields better solutions than MP_C heuristic. From the above figures, it is evident that the percentage improvement in each iteration fluctuates randomly and is independent of
Figure 8.24: Comparison of number of swaps for developed improvement heuristics
Figure 8.25: Comparison of number of moves for developed improvement heuristics
Figure 8.26: Progress of period over iterations using MP\_COSoM for 20p, 20m
Figure 8.27: Percentage reduction in period over iterations from 2\textsuperscript{nd} iteration using MP\_COSoM for 20p, 20m
Figure 8.28: Progress of period over iterations using MP_PFOSoM for 20p, 20m
Figure 8.29: Percentage reduction in period over iterations from 2\textsuperscript{nd} iteration using MP_PFOSoM for 20p, 20m
the iteration. Similar results were obtained for the other problem categories. Thus, it can be safely concluded that there does not exist any cut-off point beyond which it can be said that there is not much improvement in the quality of solution.

8.4 Batch Size-based Comparison

So far in this analysis, the batch size was always assumed to be one i.e. only a single part of each part type is considered while building the sequence of parts. Another area of possible improvement in the quality of solutions obtained is to consider a batch size of more than one i.e. multiple parts of each part type are considered while building the sequence of parts. To illustrate the impact of batch size on the computationally complexity, consider a system of 10 parts. The total number of sequences involved for a batch size of one is $9! = 362,880$ sequences. The total number of sequences involved when the batch size is three for example, is $(3\times9)! = 1.1\times10^{28}$ sequences. This shows that the total number of sequences increases exponentially when the batch size increases.

8.4.1 Deviation from Maximum Load

Figure 8.30 shows the percentage deviations from maximum load obtained using the developed heuristics with batch sizes of 1 and 3 for the 20p-20m problem category. Not all of the problem categories analyzed so far have been studied in this case due to the computational complexity involved in solving for larger problems with batch size 3. However similar results were obtained for the other problem categories that were evaluated and from the results, it is evident that there is not any significant improvement in the quality of the solution obtained by considering a higher batch size. This shows that by increasing the batch size, we do not achieve any improvement
Figure 8.30: Comparison of Max-Plus based heuristics over batch size 1 and 3 for 20p, 20m
in the quality of the solution for any of the developed improvement heuristics. The comparison of computation times for these cases are shown in the next section.

8.4.2 Computation Times and Number of Sequences

Figure 8.31 and Figure 8.32 show the computation time incurred and the number of sequences explored respectively by the various max-plus based improvement heuristics for 20p, 20m problem category. It is important to note that the scale on the y-axis of these graphs is a logarithmic scale. This was used in order to make the graph readable since the computation times incurred and the number of sequences explored for batch size 3 were significantly larger than those for batch size 1. From the above figures, it is evident that the increase in computation time does not result in a corresponding improvement in the quality of the solution. Hence, it can be concluded that increasing the batch size does not improve the quality of the solution albeit there is a significant increase in the computation times.
Figure 8.31: Comparison of computation times of Max-Plus based heuristics over batch sizes 1 and 3 for 20p, 20m
Figure 8.32: Comparison of number of sequences examined for Max-Plus based heuristics over batch sizes 1 and 3 for 20p, 20m
Chapter 9: Conclusions and Future Work

The main focus of this work has been cyclic permutation flow-shop scheduling problem. In today’s world of multitude customer choices, manufacturers often have to produce multiple types of components in large quantities. With strong emphasis on lean manufacturing, it becomes imperative to implement mixed-model scheduling so that the in-process inventory is minimized and the lead times are reduced. This is achieved through repeated production of a minimum set of parts thus reducing the lot sizes and the associated lead times. The objective was to design a mathematical formulation for the problem and develop heuristic algorithms to obtain the optimal or near-optimal solution which maximizes the throughput of the system. It is evident from the extensive literature review in Chapter 2 that this problem has received very little attention among researchers. The problem has been further restricted by eliminating buffers from the system in order to minimize the work-in-process for the system. A detailed explanation of the problem and the various assumptions was given in Chapter 3. The mathematical form of scheduling equations for this problem was the motivation behind using max-plus algebra for the mathematical formulation. The properties of max-plus algebra described in Chapter 4 allow us to apply linear algebra concepts like eigenvalue and eigenvectors to obtain a solution to the manufacturing system being modeled. The eigenvalue of the system of equations in max-plus algebra is the period of the corresponding cyclic flow-shop as shown by (3).

In this work, a mathematical formulation based on max–plus algebra has been developed for cyclic permutation flow-shops which was elaborated in Chapter 5. A construction heuristic based on NEH heuristic has also been developed using the max-plus formulation. The fact that NEH heuristic is considered one of the best
available construction heuristics was the motivation behind modeling my construction heurist on the NEH heuristic. A new concept of opportunities has been introduced in Chapter 6 that incorporates the system information of the existing schedule into the scheduling process. Five improvement heuristics also have been developed using the max-plus formulation viz. MP_CSoM, MP_COSoM, MP_OSoM, MP_PFSoM and MP_PFOSoM. These heuristics differ in the methodology employed to generate the initial solution and to identify the perturbation points. These heuristic algorithms were elaborated in Chapter 6. The commonly used perturbation mechanisms of swap and move have been used to search the solution space.

In order to compare the performances of the developed heuristic algorithm, some of the existing heuristics based on GA, SA, TS and ACO were implemented. The implementations were validated by producing results comparable to those produced by the original authors of these heuristics. Although these heuristics were originally developed for non-cyclic systems, these heuristics were applied to cyclic systems which in itself was an interesting exercise since it provided insights into how these heuristics fared when applied to cyclic systems. These heuristics have been described in Chapter 6.

All the heuristics were applied to two main categories of problems, viz. small problems for which the optimal solutions can be computed, and large problems for which the optimal solutions are not known. The samples for the small problems were randomly generated while Taillard’s benchmark problems were used as the samples for large problems. Since the optimal solutions to these problems are not known and to maintain consistency in the comparison, the percentage deviation from the maximum load which is the theoretical lower bound for the period in cyclic systems has been
used as the performance measure for both categories of problems. The results of the experimental analysis were presented in Chapter 7.

Based on the results for small problems, it was concluded that among the construction heuristics, MP\_C heuristic produces results comparable to that obtained by NEH heuristic at significantly lower computational times. The PF heuristic failed to produce results better than both NEH and MP\_C heuristic. Among the improvement heuristics, with the exception of WGA heuristic, the developed heuristics produced solutions closest to the optimal solution at much lower computational expense compared to WGA. From the results, it was concluded that although MP\_CSoM heuristic produced the best results for small problems, MP\_COSoM produced comparable results with lower computation times.

From the results obtained by executing the heuristics on Taillard’s benchmark problems, it was concluded that the PF heuristic yielded the best results with the lowest computation time among all construction heuristics. It was also concluded that among the improvement heuristics, MP\_PFOSoM heuristic produced the best results over all problem categories with minimum computation time per sequence explored. Even though the computation times for some of the existing heuristics were lower than those of MP\_PFOSoM heuristic, these heuristics produced solutions with higher percentage deviations from the maximum load.

Thus, it can be concluded that on an average, the heuristics based on the concept of opportunities perform better than the other heuristics. It can also be concluded that using a better initial solution results in a lower convergence time and an overall better solution. From the analysis, it is evident that the MP\_C heuristic and the PF heuristic are the best choices for producing the initial solution for small problems and large problems respectively.
The periods obtained in successive iterations using the developed improvement heuristics for each sample problem were analyzed to ascertain if there was any pattern to the improvement in the quality of the solution obtained over the iterations. As a result of the analysis, no such pattern was observed and thus it was concluded that a termination criterion in terms of the percentage improvement in the quality of the solution is not feasible. The impact of batch size on the quality of solution obtained was also analyzed by comparing the results obtained for batch sizes one and three. No perceptible improvements were observed to commensurate the increase in computation times for increased batch sizes.

The mathematical formulation presented in this work can be extended to other manufacturing systems like job-shops, and open-shops and the developed improvement heuristics can be employed to obtain solutions to the scheduling problems in these systems. The developed mathematical formulation can also be used to find the optimal buffer allocation for a given sequence of parts using the inter-operational dependencies. Another area to explore is the behavior of the model for stochastic processing times and sequence-dependent set-up times.
Bibliography


Appendix A: Algorithms

A.1 Algorithm to find the best perturbation using opportunities

BestPerturb\((O, S_c)\) identifies the best perturbation by exploring each opportunity and examining the reduction in period achieved. Steps 2 through 6 identify the best possible perturbation (the one that achieves maximum reduction in the period) for the pivot part in \(S_c\) by exploring two perturbation mechanisms viz. moving the part to another location and swapping the part with another part. Since there can be atmost \(n - 1\) opportunities, Steps 2 through 6 in the algorithm execute atmost \(n - 1\) times. The calculations of the \(F_{ij}\) matrices require \((3n - 10)m^2\) operations, the \texttt{Move}(\(S_c, p\)) and \texttt{Swap}(\(S_c, p\)) algorithms require \(3(n - 1)(m^2 + m^3)\) operations and finally the trace operation requires \(m\) operations. Therefore, the total number of operations required by BestPerturb\((O, S_c)\) is \((n - 1)[3(n - 1)m^3 + (6n - 13)m^2 + m]\) for a total complexity of \(O(n^2 m^3)\).
**BestPerturb**\((O, S_c)\)

Input-

\(O\) = set of opportunities

\(S_c\) = current sequence

Output-

\(S_b\) = best possible sequence

\(\pi_b\) = period for \(S_b\)

Let \(S_b = S_c\) be the current best sequence

Let \(\pi_c\) = be the period for \(S_c\)

Let \(\pi_b = \pi_c\)

--Examine all opportunities

Step 1

\(\forall o \in O\)

Step 2

Let \(i_p\) = pivot part for \(o\)

--Find the new location for pivot part either by swapping with another part or

--by moving the pivot part to another location

Step 3

--Find and store all \(F_n, j = n - 1, \cdots, 3, j \neq i_p\) and

\(F_{j,1}, j = n, \cdots, 2, j \neq i_p\) using the charts in Figures 6.2 and 6.3

Step 4

--Find the best location to move \(i_p^{th}\) part in the current sequence \(S_c\)

\((S_v, F_v) = \text{Move}(S_c, i_p)\)

--Obtain period of new sequence

\(\pi_v = \text{trace}(F_v)\)
Step 5
--Find the best candidate pair to swap \(i_p\)th part with in the current sequence \(S_c\)

\((S_w, F_w) = \text{Swap}(S_c, i_p)\)

--Obtain period of new sequence

\(\pi_w = \text{trace}(F_w)\)

Step 6
--Examine periods obtained by swap and move

\(\text{If}(\pi_v < \pi_b) \& (\pi_w < \pi_b)\)

--Both periods are better than current period. Choose the one with lower period

\(\pi_b = \min(\pi_v, \pi_w)\)

\(S_b = \text{corresponding sequence}\)

\(\text{Else If } (\pi_v < \pi_b)\)

--Period obtained using move is better than current period

\(\pi_b = \pi_v, S_b = S_v\)

\(\text{Else If } (\pi_w < \pi_b)\)

--Period obtained using swap is better than current period

\(\pi_b = \pi_w, S_b = S_w\)

Step 7
--Continue evaluating other opportunities

goto Step 2

end
A.2 Algorithm to find the best perturbation by exploring all possible alternatives

EvaluateAllPerturb(O, S_c) finds the best perturbation by exploring each of the available locations. Steps 4 and 5 evaluate the best possible moves and swaps possible for a given part respectively. Step 6 updates the current best solution and the algorithm repeats for all parts (Step 2). However, just as in BestPerturb(O, S_c), the intermediate product matrices can be efficiently computed and reused thereby eliminating the necessity to recompute them for every single iteration. The complexity of this algorithm is also given by $O(n^2m^3)$ as computed in Appendix.
EvaluateAllPerturb\((O, S_c)\)

Input-
\(S_c = \text{current sequence}\)

Output-
\(S_b = \text{best possible sequence}\)
\(\pi_b = \text{period for } S_b\)

Let \(S_b = S_c\) be the current best sequence
Let \(\pi_c = \) be the period for \(S_c\)
Let \(\pi_b = \pi_c\)

--Examine all opportunities

Step 1
\(\forall i = 1, 2, \ldots, n\)

Step 2
--Find the new location for the \(i^{th}\) part either by swapping with another part or
--by moving the pivot part to another location

Step 3
--Find and store all \(F_{n,j}, j = n - 1, \ldots, 3, j \neq i\) and
\(F_{j,1}, j = n, \ldots, 2, j \neq i\) using the charts in Figures 6.2 and 6.3

Step 4
--Find the best location to move \(i^{th}\) part in the current sequence \(S_c\)
\((S_v, F_v) = \text{Move}(S_c, i)\)

--Obtain period of new sequence
\(\pi_v = \text{trace}(F_v)\)
Step 5

--Find the best candidate pair to swap $i^{th}$ part with in the current sequence $S_c$

$(S_w, F_w) = \text{Swap}(S_c, i)$

--Obtain period of new sequence

$\pi_w = \text{trace}(F_w)$

Step 6

--Examine periods obtained by swap and move

If $(\pi_v < \pi_b)$ & $(\pi_w < \pi_b)$

--Both periods are better than current period. Choose the one with lower period

$\pi_b = \min(\pi_v, \pi_w)$

$S_b =$ corresponding sequence

Else If $(\pi_v < \pi_b)$

--Period obtained using move is better than current period

$\pi_b = \pi_v, S_b = S_v$

Else If $(\pi_w < \pi_b)$

--Period obtained using swap is better than current period

$\pi_b = \pi_w, S_b = S_w$

Step 7

--Continue evaluating other parts

goto Step 2

end
A.3 Algorithm to compute the best possible location to move a part

Move($S_c, p$) identifies the best possible location to move the $p^{th}$ part in the sequence $S_c$ by evaluating each location $i$ in the sequence $S_c$ (Step 1). The new sequence after moving the $p^{th}$ part to location $i$ corresponds to the $i^{th}$ row in Figure 6.2. The matrix $F$ requires one matrix multiplication and one # operation. Since $(n - 1)$ $F$ matrices are calculated, the total number of operations required by this algorithm is $(n - 1)(m^3 + m^2)$ with a total complexity of $O(nm^3)$. 
Move($S_c, p$)

Input-

$S_c =$ current sequence

$p =$ location of the part to be moved to a new location

Output-

new sequence and its period

Let $S_b$ denote the best sequence

$\pi_c =$ period of the sequence $S_c$

Step 1

--Examine all possible locations to move $p^{th}$ part

$\forall i = 1, 2, \ldots, n, \ i \neq p$

Step 2

--Construct a new sequence after moving $p^{th}$ part to $i^{th}$ location

Let $S_{ci}$ represent the $i^{th}$ part in $S_c$

$F = (F_{1,i-1} \# p_{Scp}) \otimes F_{i,n}$

--Obtain period of new sequence

$\pi_t = \text{trace}(F)$

If $\pi_t < \pi_c$

--Period is better than current period

$\pi_c = \pi_t \ S_b = S_c$ with $S_{cp}$ removed and reinserted in the $i^{th}$ position

If $i = n$ goto end

Else goto Step 2

end
A.4 Algorithm for MP_CSoM heuristic

\textbf{MPCOSoM()}

Input-

\( n = \) number of parts
\( m = \) number of machines
\( P = \) processing times for all operations

Output-

\( S_f = \) best possible sequence
\( \pi_f = \) period for \( S_f \)

Let \( m_c \) denote the critical machine

Let \( S_f = S_c \)

Step 0

--\textit{Construct the initial solution using MP\_C heuristic}

Let \( S_c = \) sequence obtained using MP\_C algorithm

Let \( F = \) matrix for \( S_c \) using equation 5.14

Step 1

--\textit{Obtain the period for the initial solution}

Let \( \pi_c = \) period using \text{trace}(F)
Step 2

\((S_c, \pi_c) = \text{EvaluateAllPerturb}(S_c)\)

If \(S_c = S_f\) goto end

Else

\(\pi_f = \pi_c; S_f = S_c\)

goto Step 2

end
A.5 Algorithm for MP_OSoM heuristic

\texttt{MPOSOM()}

\textbf{Input-}

\begin{itemize}
  \item \texttt{n} = number of parts
  \item \texttt{m} = number of machines
  \item \texttt{P} = processing times for all operations
\end{itemize}

\textbf{Output-}

\begin{itemize}
  \item \texttt{S}_f = best possible sequence
  \item \texttt{\pi}_f = period for \texttt{S}_f
\end{itemize}

Let \texttt{m}_c denote the critical machine

Let \texttt{S}_f = \texttt{S}_c

\textbf{Step 0}

---Construct the initial solution using a random sequence

Let \texttt{S}_c = sequence obtained using the random sequence

Let \texttt{F} = matrix for \texttt{S}_c using equation 5.14

Let \texttt{C} = completion times for all operations

Let \texttt{B} = block times for all operations

Let \texttt{D} = slack times for all operations

\textbf{Step 1}

---Obtain the period for the initial solution

Let \texttt{\pi}_c = period using trace(\texttt{F})
Step 2

--Obtain the list of opportunities for the sequence

\[ O = \text{Ops}(S_c, m_c, \mathbf{C}, \mathbf{P}, \mathbf{B}, \mathbf{D}) \]

If \( O = \emptyset \)

--There are no opportunities to reduce period.
--Current sequence is the best sequence

goto end

Else

Step 3

\((S_c, \pi_c) = \text{BestPerturb}(O, S_c)\)

If \( S_c = S_f \) goto end

Else

\( \pi_f = \pi_c; S_f = S_c \)

goto Step 2

end
A.6 Algorithm for MP_PFOSoM heuristic

\textbf{MPPFOSoM()}

Input-
\begin{itemize}
\item n = number of parts
\item m = number of machines
\item P = processing times for all operations
\end{itemize}

Output-
\begin{itemize}
\item $S_f$ = best possible sequence
\item $\pi_f$ = period for $S_f$
\end{itemize}

Let $m_c$ denote the critical machine

Let $S_f = S_c$

Step 0

\textit{Construct the initial solution using PF heuristic}

Let $S_c$ = sequence obtained using PF algorithm

Let $F$ = matrix for $S_c$ using equation 5.14

Let $C$ = completion times for all operations

Let $B$ = block times for all operations

Let $D$ = slack times for all operations

Step 1

\textit{Obtain the period for the initial solution}

Let $\pi_c$ = period using trace($F$)
Step 2

--Obtain the list of opportunities for the sequence

\(O = \text{Ops}(S_c, m_c, C, P, B, D)\)

If \(O = \emptyset\)

--There are no opportunities to reduce period.

--Current sequence is the best sequence

goto end

Else

Step 3

\((S_c, \pi_c) = \text{BestPerturb}(O, S_c)\)

If \(S_c = S_f\) goto end

Else

\(\pi_f = \pi_c; S_f = S_c\)

goto Step 2

end

A.7 Algorithm to find the opportunites and the corresponding pivot parts

In \(\text{Ops}(S_e, m_e, C, P, B, D)\), all \(n\) operations on the critical machine \(m_e\) are examined to identify opportunities for reducing the period. If an operation on \(m_e\) is blocked by a downstream machine, all downstream machines and operations are examined beginning with the \((m_e+1)^{th}\) machine to identify the pivot part \(i_p\) that causes the blocking. It can be seen that in the worst case, if \(m_e = 1\), \(i_p\) could be on the \(m^{th}\) machine. If \(m_e\) is idle after an operation, the \((m_e-1)^{th}\) machine and the \((i+1)^{th}\) part
is examined to identify $i_p$ that causes $m_c$ to be idle. Thus the complexity to identify all opportunities can be given by $O(nm)$.

\textbf{Ops($S_c, m_c, C, P, B, D$)}

\textbf{Input-}

$S_c =$ current sequence

$m_c =$ critical machine

$C =$ completion times for all operations

$P =$ processing times for all operations

$B =$ block times for all operations

$D =$ slack times for all operations

\textbf{Output-}

$O =$ list of opportunities

Let $O = \{ \}$

\textbf{Step 1}

--Examine all operations on critical machine for potential opportunities

\forall i = 1, 2, \ldots, n

\textbf{Step 2}

--Critical machine is blocked after $i^{th}$ part

If $b_{m_c,i} \neq 0$

--Find the part that causes the critical machine to be blocked by examining

--downstream machines and parts.
find \( i_p = i - 1, i - 2, \ldots, j = m_c + 1, m_c + 2, \ldots, m \) such that
\[ c_{j,i_p} = c_{m_c,i} + b_{m_c,i} \]
--If \( i_p \leq 0 \) then \( i_p = i_p + n \)
--Add the \((i_p)\)th part, the critical machine \( m_c \), the \( i \)th part to \( O \)
\[ \text{add}(m_c, i, i_p) \text{ to } O \]
--Critical machine is idle after \( i \)th part
Else If \( d_{m_c,i} \neq 0 \)
--Slack time is always due to the previous part in the sequence.
\[ i_p = i + 1, j = m_c - 1 \text{ such that} \]
\[ c_{j,i_p} = s_{m_c,i_p} \& b_{j,i_p} = 0 \]
--If \( i_p \geq n \) then \( i_p = i_p - n \)
--Add the \((i_p)\)th part, the critical machine \( m_c \), the \( i \)th part to \( O \)
\[ \text{add}(m_c, i, i_p) \text{ to } O \]
If \( i \neq n \), goto Step 2
Else goto end
\end

A.8 Algorithm to find the best possible swap pair

\( \text{Swap}(S_c, p) \) identifies the best candidate part to swap the \( p \)th part with in the sequence \( S_c \) by evaluating all possible candidate parts. There are two cases in this algorithm viz. when the candidate part is at a location that occurs before \( p \) in \( S_c \) (Step 1) and when the candidate part is at a location that occurs after \( p \) in \( S_c \) (Step 3). The new sequence after swapping the \( i \)th part in location with the \( p \)th part corresponds to the \( i \)th row in Figure 6.3. The \( F \) requires two matrix multiplications and two #
operations. Since \((n - 1)\) \(F\) matrices are calculated, the total operations required by this algorithm is \((n - 1) (m^3 + m^2)\) with a total complexity of \(O(nm^3)\).

**Swap** \((S_c, p)\)

**Input-**

\(S_c = \text{current sequence}\)

\(p = \text{location of the part to be swapped with candidate part}\)

**Output-**

new sequence and its period

Let \(S_b\) denote the best sequence

\(\pi_c = \text{period of } S_c\)

**Step 1**

--Examine all parts in locations before the \(p^{th}\) part to find a candidate part to swap

\(\forall i = 1, 2, \ldots, p - 1\)

**Step 2**

--Construct the matrix after swapping \(p^{th}\) part with \(i^{th}\) part

Let \(S_{ci}\) represent the \(i^{th}\) part in \(S_c\)

\(F = (F_{1,i-1} \# p_{S_{cp}}) \otimes (F_{i+1,p-1} \# p_{S_{ci}}) \otimes F_{p+1,n}\)

--Obtain period of new sequence

\(\pi_t = \text{trace}(F)\)

If \(\pi_t < \pi_c\)

\(\pi_c = \pi_t, \ S_b = S_c \text{ with } S_{ci} \text{ and } S_{cp} \text{ swapped}\)

If \(i = p - 1\) goto Step 3

Else goto Step 1
Step 3

--Examine all parts in locations after the $p^{th}$ part to find a candidate part to swap

\[ \forall i = p + 1, p + 2, \ldots, n \]

Step 4

--Construct the matrix after swapping $p^{th}$ part with $i^{th}$ part

\[ F = (F_{1,p-1} \# p_{S_{c_i}}) \otimes (F_{p+1,i-1} \# p_{S_{c_p}}) \otimes F_{i+1,n} \]

--Obtain period of new sequence

\[ \pi_t = \text{trace}(F) \]

If $\pi_t < \pi_c$

--Period is better than current period

\[ \pi_c = \pi_t, \quad S_b = S_c \text{ with } S_{c_i} \text{ and } S_{c_p} \text{ swapped} \]

If $i = n$ goto end

Else goto Step 3

end
Table B.1: Percentage Deviation from Maximum Load for Construction Heuristics on Small Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>6-6</th>
<th>7-7</th>
<th>8-8</th>
<th>8-4</th>
<th>Rand6-8</th>
<th>Rand4-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANDOM</td>
<td>30.56</td>
<td>23.98</td>
<td>22.23</td>
<td>25.75</td>
<td>22.55</td>
<td>28.08</td>
</tr>
<tr>
<td>MWKR</td>
<td>32.50</td>
<td>23.04</td>
<td>28.58</td>
<td>26.52</td>
<td>23.22</td>
<td>27.99</td>
</tr>
<tr>
<td>LWKR</td>
<td>30.46</td>
<td>25.63</td>
<td>25.08</td>
<td>27.69</td>
<td>17.86</td>
<td>27.70</td>
</tr>
<tr>
<td>PF</td>
<td>27.97</td>
<td>19.65</td>
<td>19.98</td>
<td>16.39</td>
<td>14.79</td>
<td>23.87</td>
</tr>
<tr>
<td>MP_C</td>
<td>23.51</td>
<td>15.79</td>
<td>16.39</td>
<td>14.96</td>
<td>13.86</td>
<td>18.37</td>
</tr>
<tr>
<td>NEH</td>
<td>23.51</td>
<td>15.79</td>
<td>16.39</td>
<td>14.96</td>
<td>13.86</td>
<td>18.37</td>
</tr>
</tbody>
</table>
Table B.2: Percentage Deviation from Maximum Load for Improvements Heuristics on Small Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Sample Data Category</th>
<th>6-6</th>
<th>7-7</th>
<th>8-8</th>
<th>8-4</th>
<th>Rand6-8</th>
<th>Rand4-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS</td>
<td></td>
<td>29.71</td>
<td>23.34</td>
<td>23.83</td>
<td>22.39</td>
<td>19.91</td>
<td>28.72</td>
</tr>
<tr>
<td>RANDOM</td>
<td></td>
<td>30.56</td>
<td>23.98</td>
<td>22.23</td>
<td>25.75</td>
<td>22.55</td>
<td>28.08</td>
</tr>
<tr>
<td>ISA</td>
<td></td>
<td>31.54</td>
<td>25.26</td>
<td>24.08</td>
<td>24.43</td>
<td>21.27</td>
<td>27.06</td>
</tr>
<tr>
<td>OSA</td>
<td></td>
<td>29.80</td>
<td>24.54</td>
<td>25.77</td>
<td>28.98</td>
<td>20.91</td>
<td>26.36</td>
</tr>
<tr>
<td>ACOI</td>
<td></td>
<td>25.11</td>
<td>21.21</td>
<td>19.74</td>
<td>20.84</td>
<td>15.72</td>
<td>24.40</td>
</tr>
<tr>
<td>CGA</td>
<td></td>
<td>30.43</td>
<td>24.98</td>
<td>22.49</td>
<td>28.31</td>
<td>20.81</td>
<td>23.30</td>
</tr>
<tr>
<td>HAMC</td>
<td></td>
<td>24.46</td>
<td>18.81</td>
<td>19.83</td>
<td>19.54</td>
<td>14.68</td>
<td>21.49</td>
</tr>
<tr>
<td>MP_OSoM</td>
<td></td>
<td>19.97</td>
<td>14.66</td>
<td>14.18</td>
<td>12.02</td>
<td>10.67</td>
<td>16.50</td>
</tr>
<tr>
<td>MP_COSoM</td>
<td></td>
<td>20.35</td>
<td>14.87</td>
<td>14.00</td>
<td>11.69</td>
<td>11.49</td>
<td>16.46</td>
</tr>
<tr>
<td>MP_PFOSoM</td>
<td></td>
<td>20.03</td>
<td>13.97</td>
<td>14.26</td>
<td>12.05</td>
<td>11.40</td>
<td>16.09</td>
</tr>
<tr>
<td>MP_CSoM</td>
<td></td>
<td>20.08</td>
<td>13.13</td>
<td>13.92</td>
<td>11.12</td>
<td>10.27</td>
<td>15.22</td>
</tr>
<tr>
<td>WGA</td>
<td></td>
<td>19.78</td>
<td>15.53</td>
<td>14.06</td>
<td>13.35</td>
<td>10.45</td>
<td>14.83</td>
</tr>
<tr>
<td>MP_PFSoM</td>
<td></td>
<td>19.78</td>
<td>13.80</td>
<td>13.15</td>
<td>11.78</td>
<td>11.86</td>
<td>14.73</td>
</tr>
<tr>
<td>Exhaustive Search</td>
<td></td>
<td>19.78</td>
<td>13.11</td>
<td>12.72</td>
<td>11.02</td>
<td>9.06</td>
<td>14.73</td>
</tr>
</tbody>
</table>

Table B.3: Execution Time (ms) of Construction Heuristics on Small Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Sample Data Category</th>
<th>6-6</th>
<th>7-7</th>
<th>8-8</th>
<th>8-4</th>
<th>Rand6-8</th>
<th>Rand4-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANDOM</td>
<td></td>
<td>4.70</td>
<td>0.00</td>
<td>15.67</td>
<td>10.90</td>
<td>9.20</td>
<td>3.10</td>
</tr>
<tr>
<td>NEH</td>
<td></td>
<td>1.60</td>
<td>4.60</td>
<td>2.50</td>
<td>0.00</td>
<td>0.00</td>
<td>1.60</td>
</tr>
<tr>
<td>MP_C</td>
<td></td>
<td>10.90</td>
<td>17.30</td>
<td>26.00</td>
<td>1.60</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>MWKR</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.60</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LWKR</td>
<td></td>
<td>0.00</td>
<td>1.50</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>PF</td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Table B.4: Execution Time (ms) of Improvement Heuristics on Small Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Sample Data Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6-6</td>
</tr>
<tr>
<td>Exhaustive Search</td>
<td>12.3</td>
</tr>
<tr>
<td>WGA</td>
<td>78.10</td>
</tr>
<tr>
<td>ACO1</td>
<td>51.70</td>
</tr>
<tr>
<td>ISA</td>
<td>51.70</td>
</tr>
<tr>
<td>OSA</td>
<td>45.40</td>
</tr>
<tr>
<td>HAMC</td>
<td>6.30</td>
</tr>
<tr>
<td>MP_CSoM</td>
<td>6.20</td>
</tr>
<tr>
<td>MP_PFOSoM</td>
<td>4.70</td>
</tr>
<tr>
<td>MP_PFSoM</td>
<td>7.70</td>
</tr>
<tr>
<td>MP_COSoM</td>
<td>6.40</td>
</tr>
<tr>
<td>CGA</td>
<td>9.40</td>
</tr>
<tr>
<td>MP_OSoM</td>
<td>4.60</td>
</tr>
<tr>
<td>TS</td>
<td>3.00</td>
</tr>
</tbody>
</table>
Table B.5: Percentage Deviation from Maximum Load for Construction Heuristics on Large Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>20-5</th>
<th>50-5</th>
<th>100-5</th>
<th>20-10</th>
<th>50-10</th>
<th>100-10</th>
<th>20-20</th>
<th>50-20</th>
<th>100-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANDOM</td>
<td>41.72</td>
<td>46.62</td>
<td>51.01</td>
<td>40.51</td>
<td>50.30</td>
<td>51.67</td>
<td>40.54</td>
<td>46.57</td>
<td>51.23</td>
</tr>
<tr>
<td>LWKR</td>
<td>37.93</td>
<td>40.36</td>
<td>43.08</td>
<td>40.20</td>
<td>47.94</td>
<td>47.44</td>
<td>40.64</td>
<td>45.84</td>
<td>50.53</td>
</tr>
<tr>
<td>MWKR</td>
<td>39.68</td>
<td>39.86</td>
<td>42.74</td>
<td>39.59</td>
<td>49.24</td>
<td>48.42</td>
<td>42.03</td>
<td>47.52</td>
<td>50.06</td>
</tr>
<tr>
<td>MP_C</td>
<td>23.49</td>
<td>22.54</td>
<td>24.45</td>
<td>30.47</td>
<td>34.66</td>
<td>33.66</td>
<td>32.48</td>
<td>36.81</td>
<td>39.42</td>
</tr>
<tr>
<td>NEH</td>
<td>23.49</td>
<td>22.54</td>
<td>24.45</td>
<td>30.47</td>
<td>34.66</td>
<td>33.66</td>
<td>32.48</td>
<td>36.81</td>
<td>39.42</td>
</tr>
<tr>
<td>PF</td>
<td>21.88</td>
<td>19.86</td>
<td>18.10</td>
<td>27.49</td>
<td>29.72</td>
<td>27.37</td>
<td>33.66</td>
<td>34.53</td>
<td>35.17</td>
</tr>
</tbody>
</table>
Table B.6: Percentage Deviation from Maximum Load for Improvement Heuristics on Large Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Sample Data Category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20-5</td>
</tr>
<tr>
<td>ISA</td>
<td>40.36</td>
</tr>
<tr>
<td>CGA</td>
<td>40.61</td>
</tr>
<tr>
<td>RANDOM</td>
<td>41.72</td>
</tr>
<tr>
<td>OSA</td>
<td>41.34</td>
</tr>
<tr>
<td>TS</td>
<td>35.72</td>
</tr>
<tr>
<td>WGA</td>
<td>31.28</td>
</tr>
<tr>
<td>ACO1</td>
<td>35.97</td>
</tr>
<tr>
<td>HAMC</td>
<td>32.44</td>
</tr>
<tr>
<td>MP_OSoM</td>
<td>19.11</td>
</tr>
<tr>
<td>MP_COSoM</td>
<td>17.98</td>
</tr>
<tr>
<td>MP_CSoM</td>
<td>17.47</td>
</tr>
<tr>
<td>MP_PFSoM</td>
<td>17.62</td>
</tr>
<tr>
<td>MP_PFSoM</td>
<td>16.69</td>
</tr>
</tbody>
</table>
Table B.7: Execution Time (ms) for Construction Heuristics on Large Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Sample Data Category</th>
<th>20-5</th>
<th>50-5</th>
<th>100-5</th>
<th>20-10</th>
<th>50-10</th>
<th>100-10</th>
<th>20-20</th>
<th>50-20</th>
<th>100-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_C</td>
<td></td>
<td>32</td>
<td>117</td>
<td>1938</td>
<td>53</td>
<td>348</td>
<td>5339</td>
<td>90</td>
<td>1167</td>
<td>17711</td>
</tr>
<tr>
<td>NEH</td>
<td></td>
<td>38</td>
<td>349</td>
<td>2763</td>
<td>81</td>
<td>748</td>
<td>5349</td>
<td>482</td>
<td>3419</td>
<td>14852</td>
</tr>
<tr>
<td>PF</td>
<td></td>
<td>4</td>
<td>43</td>
<td>286</td>
<td>8</td>
<td>83</td>
<td>672</td>
<td>16</td>
<td>154</td>
<td>1082</td>
</tr>
</tbody>
</table>
Table B.8: Execution Time (ms) for Improvement Heuristics on Large Problems

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>20-5</th>
<th>50-5</th>
<th>100-5</th>
<th>20-10</th>
<th>50-10</th>
<th>100-10</th>
<th>20-20</th>
<th>50-20</th>
<th>100-20</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_CSoM</td>
<td>224</td>
<td>4810</td>
<td>256860</td>
<td>767</td>
<td>15411</td>
<td>437683</td>
<td>3590</td>
<td>61039</td>
<td>1160722</td>
</tr>
<tr>
<td>WGA</td>
<td>983</td>
<td>8774</td>
<td>167000</td>
<td>3482</td>
<td>22193</td>
<td>218843</td>
<td>19127</td>
<td>85968</td>
<td>458994</td>
</tr>
<tr>
<td>MP_PFSoM</td>
<td>159</td>
<td>2948</td>
<td>92681</td>
<td>603</td>
<td>7445</td>
<td>153884</td>
<td>3454</td>
<td>38642</td>
<td>446106</td>
</tr>
<tr>
<td>MP_OSoM</td>
<td>103</td>
<td>1373</td>
<td>37582</td>
<td>311</td>
<td>3336</td>
<td>88953</td>
<td>1342</td>
<td>21153</td>
<td>212541</td>
</tr>
<tr>
<td>MP_COSoM</td>
<td>90</td>
<td>1329</td>
<td>30231</td>
<td>340</td>
<td>3490</td>
<td>79408</td>
<td>1303</td>
<td>16970</td>
<td>203450</td>
</tr>
<tr>
<td>CGA</td>
<td>77</td>
<td>16620</td>
<td>130582</td>
<td>132</td>
<td>1792</td>
<td>31697</td>
<td>548</td>
<td>20270</td>
<td>173013</td>
</tr>
<tr>
<td>MP_PFSoM</td>
<td>71</td>
<td>960</td>
<td>17992</td>
<td>249</td>
<td>2271</td>
<td>38573</td>
<td>1504</td>
<td>15304</td>
<td>112141</td>
</tr>
<tr>
<td>HAMC</td>
<td>120</td>
<td>2328</td>
<td>72438</td>
<td>195</td>
<td>1915</td>
<td>67789</td>
<td>532</td>
<td>4618</td>
<td>81029</td>
</tr>
<tr>
<td>ACO1</td>
<td>713</td>
<td>7127</td>
<td>24237</td>
<td>4169</td>
<td>8373</td>
<td>47923</td>
<td>4995</td>
<td>29455</td>
<td>77243</td>
</tr>
<tr>
<td>TS</td>
<td>28</td>
<td>368</td>
<td>2756</td>
<td>85</td>
<td>777</td>
<td>5545</td>
<td>483</td>
<td>3464</td>
<td>14958</td>
</tr>
<tr>
<td>OSA</td>
<td>198</td>
<td>466</td>
<td>1015</td>
<td>418</td>
<td>867</td>
<td>2074</td>
<td>1200</td>
<td>1959</td>
<td>4061</td>
</tr>
<tr>
<td>ISA</td>
<td>209</td>
<td>463</td>
<td>952</td>
<td>429</td>
<td>889</td>
<td>1666</td>
<td>1284</td>
<td>1927</td>
<td>3346</td>
</tr>
</tbody>
</table>
Table B.9: Percentage Reduction in Period at Each Iteration for MP_PFOSoM Heuristic

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Sample Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1.41</td>
</tr>
<tr>
<td>2</td>
<td>0.31</td>
</tr>
<tr>
<td>3</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>0.81</td>
</tr>
<tr>
<td>5</td>
<td>0.06</td>
</tr>
<tr>
<td>6</td>
<td>0.44</td>
</tr>
<tr>
<td>7</td>
<td>0.57</td>
</tr>
<tr>
<td>8</td>
<td>0.13</td>
</tr>
<tr>
<td>9</td>
<td>0.13</td>
</tr>
<tr>
<td>10</td>
<td>0.32</td>
</tr>
<tr>
<td>11</td>
<td>0.06</td>
</tr>
<tr>
<td>12</td>
<td>0.19</td>
</tr>
<tr>
<td>13</td>
<td>0.06</td>
</tr>
<tr>
<td>14</td>
<td>0.06</td>
</tr>
<tr>
<td>15</td>
<td>0.39</td>
</tr>
<tr>
<td>16</td>
<td>0.32</td>
</tr>
<tr>
<td>17</td>
<td>0.26</td>
</tr>
<tr>
<td>18</td>
<td>0.26</td>
</tr>
<tr>
<td>19</td>
<td>0.46</td>
</tr>
<tr>
<td>20</td>
<td>0.20</td>
</tr>
</tbody>
</table>


Table B.10: Percentage Reduction in Period at Each Iteration for MP_COSoM

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Sample Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5.70</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.50</td>
</tr>
<tr>
<td>4</td>
<td>0.38</td>
</tr>
<tr>
<td>5</td>
<td>0.13</td>
</tr>
<tr>
<td>6</td>
<td>0.13</td>
</tr>
<tr>
<td>7</td>
<td>0.13</td>
</tr>
<tr>
<td>8</td>
<td>0.44</td>
</tr>
<tr>
<td>9</td>
<td>0.38</td>
</tr>
<tr>
<td>10</td>
<td>0.26</td>
</tr>
<tr>
<td>11</td>
<td>0.13</td>
</tr>
<tr>
<td>12</td>
<td>0.26</td>
</tr>
<tr>
<td>13</td>
<td>0.19</td>
</tr>
<tr>
<td>14</td>
<td>0.13</td>
</tr>
</tbody>
</table>