SPACE SEARCH BASED ALGORITHM FOR CELL FORMATION WITH
ALTERNATIVE PROCESS PLANS

A thesis presented to
the faculty of the
Fritz J. and Dolores H. Russ
College of Engineering and Technology
of
Ohio University

In partial fulfillment
of the requirement for the degree
Master of Science

by
Srinivas N. Rajaraman

November 2003
This thesis entitled

SPACE SEARCH BASED ALGORITHM FOR CELL FORMATION WITH
ALTERNATIVE PROCESS PLANS

by

Srinivas N. Rajaraman

has been approved

for the Department of Industrial and Manufacturing Systems Engineering
and the Russ College of Engineering and Technology

Dušan Šormaz
Associate Professor of Industrial and
Manufacturing Systems Engineering

R. Dennis Irwin
Dean Fritz J. and Dolores H. Russ
College of Engineering and Technology
Cell formation is the process of identifying independent part families and machine cells in a manufacturing system. Space search is an artificial intelligence technique for problem solving. In this thesis, an algorithm to perform space search for cell formation in the presence of alternative process plans has been developed. Simultaneous decision making for cell formation and process plan selection is performed by the algorithm. The algorithm starts by sorting machines in the manufacturing system according to SICGE coding and usage frequencies. Machines are arranged into the sorted list and the first one is selected as a key machine for a state. Parts with activities on the key machine are selected as candidate parts for family formation. Incremental part families are formed from the candidate parts and they become the root family of a child state. Space states are evaluated based on the count of the number of exceptional machine operations (operations outside the cell) in each state. Space states are explored based on the best search technique. The algorithm finds the best solution based on the number of exceptions. The algorithm can also handle multiple machine units assigning one machine unit to a cell if there is a demand for that machine in
that cell. Two software prototypes called Factory Model Generator and Space Searcher were developed to perform collection of factory model information and space search on the cell formation problem. The algorithm has been tested on several examples from research journals for solution quality as well as to study the efficiency of the space search algorithm. Results obtained from the testing in terms of the number of exceptional machine operations show that algorithm performs better or on the same level as other previous algorithms.

Approved:

Dušan Šormaz

Associate Professor of Industrial and Manufacturing Systems Engineering
ACKNOWLEDGMENTS

I would like to dedicate this thesis to my father, N. Rajaraman, my mother, R. Vidhya, my sister R. Rohini and my fiancée A. Ashwini, whose motivation and continuous support has enabled me to reach this pinnacle of academic achievement. I would like to thank Dr. Dušan Šormaz for his continuous guidance and motivation towards the completion of this thesis. The knowledge he has provided me in the course of this thesis work is going to be my guiding light throughout my professional career. I would like to thank my committee members Dr. Gürsel A. Süer and Dr. Constantinos Vassiliadis for their valuable suggestions and insights for my thesis work. I would like to thank my fellow researchers and friends in the LabIMP research group, Harihara Ramachandra Sharma, Sam S. Kolli and Vikesh B. Jain for their useful suggestions in the completion of my thesis work. I would also like to thank my friends Prabhu, Akhtar Ali, Swaminathan, Leela Krishnan and Gayathri for their continuous support.
# TABLE OF CONTENTS

## 1 INTRODUCTION

- 1.1 Cellular Manufacturing and Alternative Process Plans ........................................... 13
- 1.2 Artificial Intelligence and Space Search ................................................................. 15
- 1.3 Motivation .................................................................................................................. 17
- 1.4 Research Objective and Goals .................................................................................... 19
- 1.5 Application Development for Cellular Manufacturing ................................................. 22
- 1.6 Structure of Thesis ...................................................................................................... 24

## 2 BACKGROUND AND RELATED WORK

- 2.1 Cellular Manufacturing Without Alternative Process Plans ........................................ 27
  - 2.1.1 Operations research .................................................................................................. 27
  - 2.1.2 Artificial Intelligence ............................................................................................... 33
    - 2.1.2.1 Genetic Algorithms .............................................................................................. 33
    - 2.1.2.2 Neural Networks .................................................................................................. 35
    - 2.1.2.3 Space Search ........................................................................................................ 36
  - 2.1.3 Industrial Applications ............................................................................................. 37
- 2.2 Cellular Manufacturing With Alternative Process Plans ........................................... 42
  - 2.2.1 Phased Approach for Cellular Manufacturing ........................................................ 43
  - 2.2.2 Simultaneous Process Planning and Cellular Manufacturing .................................... 44
- 2.3 Knowledge-Based Intelligent Manufacturing ............................................................. 45
  - 2.3.1 Manufacturing Activity Modeling ............................................................................ 46
  - 2.3.2 Knowledge-Based Group Technology Systems ....................................................... 49
  - 2.3.3 Generation of Alternative Process Plans ................................................................. 51
- 2.4 Overview of Java and XML Technologies ................................................................. 53
  - 2.4.1 Java Technologies .................................................................................................... 53
  - 2.4.2 XML Technologies .................................................................................................. 55
  - 2.4.3 Components of an XML Document ........................................................................ 56
  - 2.4.4 XML Transformation with XSLT ........................................................................... 58

## 3 METHODOLOGY

- 3.1 Cellular Manufacturing Without Alternative Process Plans ........................................ 61
  - 3.1.1 Problem Formulation ............................................................................................... 61
  - 3.1.2 State Space Representation ...................................................................................... 64
  - 3.1.3 State Transition Mechanism ..................................................................................... 66
  - 3.1.4 State Ordering and Filtering Mechanism ................................................................. 70
    - 3.1.4.1 Heuristic for State Evaluation .............................................................................. 71
    - 3.1.4.2 Computation of Total Traffic Reduction for a State $S$ ...................................... 72
  - 3.1.5 Space Search with Best Search Algorithm .............................................................. 73
  - 3.1.6 Search Termination ................................................................................................. 74
3.2 CELL FORMATION WITH ALTERNATIVE PROCESS PLANS ................................................................. 74
  3.2.1 Problem formulation .................................................................................................................. 74
  3.2.2 Preparation of machines for the space search algorithm ....................................................... 78
    3.2.2.1 Calculation of machine usage frequency ............................................................................. 78
    3.2.2.2 Sorting the machines in the manufacturing system .......................................................... 80
  3.2.3 State space representation ........................................................................................................ 81
  3.2.4 State transition mechanism ..................................................................................................... 84
  3.2.5 State ordering and filtering mechanism .................................................................................... 88
    3.2.5.1 Heuristic for state ordering ................................................................................................. 89
    3.2.5.2 Heuristic for state elimination ............................................................................................ 89
  3.2.6 Space search with best search technique .................................................................................. 91
  3.2.7 Search termination ................................................................................................................... 94

4 SYSTEM ARCHITECTURE AND IMPLEMENTATION ................................................................. 95
  4.1 PROCESS PLAN REPRESENTATION AND DATA GENERATION .................................................. 95
    4.1.1 Process plan representation model ......................................................................................... 95
    4.1.2 Definition of Java classes for models ...................................................................................... 98
    4.1.3 Factory model generator ....................................................................................................... 100
  4.2 XML DATA GENERATION AND PARSING ................................................................................ 105
    4.2.1 XML schema for PPRM ......................................................................................................... 105
    4.2.2 XML data generation mechanism .......................................................................................... 106
    4.2.3 XML data parsing .................................................................................................................. 109
  4.3 SPACE SEARCHER ......................................................................................................................... 113
    4.3.1 Searchable interface .............................................................................................................. 113
    4.3.2 Default space state ................................................................................................................. 118
    4.3.3 Space searcher tool ............................................................................................................... 118
  4.4 IMPLEMENTATION OF SPACE SEARCH ALGORITHM FOR CELL FORMATION
      WITHOUT ALTERNATIVE PROCESS PLANS ................................................................................ 123
    4.4.1 Implementation of Java classes ............................................................................................... 123
    4.4.2 State transition mechanism ................................................................................................... 126
    4.4.3 State comparison, ordering and goal switching ....................................................................... 127
  4.5 IMPLEMENTATION OF SPACE SEARCH ALGORITHM FOR CELL FORMATION WITH
      ALTERNATIVE PROCESS PLANS ................................................................................................. 129
    4.5.1 Implementation of Java classes ............................................................................................... 129
    4.5.2 State transition mechanism ................................................................................................... 131
    4.5.3 State comparison, ordering and goal switching ....................................................................... 135

5 TESTING AND EXAMPLES ........................................................................................................... 137
  5.1 JAE EXAMPLE .............................................................................................................................. 140
    5.1.1 Data generation ....................................................................................................................... 140
    5.1.2 Performance results from testing ............................................................................................ 142
    5.1.3 Comparison and evaluation of cell configurations ................................................................. 143
    5.1.4 Evaluation of presence of alternatives on quality of solution ............................................... 146
5.2 LEE 2 ................................................................................................................................. 149
   5.2.1 Data generation ........................................................................................................... 149
   5.2.2 Performance results from testing ................................................................................ 152
   5.2.3 Comparison and evaluation of cell configurations ..................................................... 154
5.3 OTHER EXAMPLES ........................................................................................................... 156

6 CONCLUSIONS AND FUTURE WORK ................................................................. 159
   6.1 RESEARCH SUMMARY ................................................................................................. 159
   6.2 CONTRIBUTIONS .......................................................................................................... 159
   6.3 LIMITATIONS ............................................................................................................... 162
   6.4 FUTURE EXTENSION .................................................................................................. 163

REFERENCES ..................................................................................................................... 166
LIST OF TABLES

Table 1-1 Initial part machine matrix for the Cell formation problem ....................... 14
Table 1-2: Manufacturing cells formed - Solution 1 .................................................. 20
Table 1-3 Manufacturing cells formed – Solution 2 .................................................. 20
Table 3-1: Part machine matrix with machine frequency values ............................. 79
Table 3-2: Machines sorted according to SICGE and usage frequency ................. 80
Table 3-3 Unassigned part machine matrix ............................................................... 84
Table 4-1: Icons on the Factory model generator ..................................................... 102
Table 5-1: Summary of test cases ............................................................................. 137
Table 5-2: Testing parameters for the cell formation with alternatives .................... 138
Table 5-3: Performance testing with 512 MB VM allocation .................................. 142
Table 5-4: Best solution for Jae from space search (Configuration A) ................. 144
Table 5-5: Solution from Jae ................................................................................... 146
Table 5-6: Performance testing on Jae with elimination of alternatives ............... 147
Table 5-7: Performance testing with 512 MB memory allocation ......................... 153
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Space search for a problem domain</td>
<td>16</td>
</tr>
<tr>
<td>2-1</td>
<td>Part machine incidence matrix from Burbidge [2]</td>
<td>38</td>
</tr>
<tr>
<td>2-2</td>
<td>Module summary from Burbidge [2]</td>
<td>40</td>
</tr>
<tr>
<td>2-3</td>
<td>Resolution of part-machine incidence matrix from Burbidge [2]</td>
<td>41</td>
</tr>
<tr>
<td>2-4</td>
<td>Product planning functions from Sormaz [23]</td>
<td>46</td>
</tr>
<tr>
<td>2-5</td>
<td>Manufacturing process model hierarchy</td>
<td>47</td>
</tr>
<tr>
<td>2-6</td>
<td>An XML document</td>
<td>57</td>
</tr>
<tr>
<td>2-7</td>
<td>A XSLT rendered XML document</td>
<td>59</td>
</tr>
<tr>
<td>3-1</td>
<td>Initial state representation</td>
<td>65</td>
</tr>
<tr>
<td>3-2</td>
<td>Intercellular traffic distribution for the initial state</td>
<td>66</td>
</tr>
<tr>
<td>3-3</td>
<td>Child 1 of initial state</td>
<td>67</td>
</tr>
<tr>
<td>3-4</td>
<td>Child 2 of initial state</td>
<td>67</td>
</tr>
<tr>
<td>3-5</td>
<td>Child 3 of initial state</td>
<td>67</td>
</tr>
<tr>
<td>3-6</td>
<td>Child 4 of initial state</td>
<td>68</td>
</tr>
<tr>
<td>3-7</td>
<td>Child 5 of initial state</td>
<td>68</td>
</tr>
<tr>
<td>3-8</td>
<td>Part families generated from state transition</td>
<td>86</td>
</tr>
<tr>
<td>4-1</td>
<td>Process plan representation model for manufacturing</td>
<td>96</td>
</tr>
<tr>
<td>4-2</td>
<td>Cellular manufacturing representation model</td>
<td>98</td>
</tr>
<tr>
<td>4-3</td>
<td>Java class for Manufacturing Activity</td>
<td>100</td>
</tr>
<tr>
<td>4-4</td>
<td>Factory model generator with parts and machines</td>
<td>101</td>
</tr>
<tr>
<td>4-5</td>
<td>Special plant list on Factory model generator</td>
<td>103</td>
</tr>
<tr>
<td>4-6</td>
<td>XML data generation panel</td>
<td>104</td>
</tr>
<tr>
<td>4-7</td>
<td>XML schema for PPRM from XML Spy Suite</td>
<td>106</td>
</tr>
<tr>
<td>4-8</td>
<td>Machine Java class showing XML data generation mechanism</td>
<td>107</td>
</tr>
<tr>
<td>4-9</td>
<td>XML Data for a Machine</td>
<td>108</td>
</tr>
<tr>
<td>4-10</td>
<td>Generating an XML content handler</td>
<td>110</td>
</tr>
<tr>
<td>4-11</td>
<td>Parsing machine XML data with a content handler</td>
<td>111</td>
</tr>
<tr>
<td>4-12</td>
<td>Searchable interface to perform space search</td>
<td>114</td>
</tr>
<tr>
<td>4-13</td>
<td>Component based description of Space Searcher</td>
<td>119</td>
</tr>
<tr>
<td>4-14</td>
<td>State inspection panel for the traveling salesman</td>
<td>121</td>
</tr>
<tr>
<td>4-15</td>
<td>Interface between Space Searcher and space search problem</td>
<td>122</td>
</tr>
<tr>
<td>4-16</td>
<td>Interface for space search from FMG</td>
<td>122</td>
</tr>
<tr>
<td>4-17</td>
<td>Space State class for cell formation without alternatives</td>
<td>124</td>
</tr>
<tr>
<td>4-18</td>
<td>Java class for manufacturing cell holding machines</td>
<td>125</td>
</tr>
<tr>
<td>4-19</td>
<td>State Transition mechanism for cell formation without alternatives</td>
<td>126</td>
</tr>
<tr>
<td>4-20</td>
<td>State comparison mechanism for CF without alternatives</td>
<td>127</td>
</tr>
<tr>
<td>4-21</td>
<td>Goal switching mechanism for cell formation without alternatives</td>
<td>129</td>
</tr>
<tr>
<td>4-22</td>
<td>Space State Java class for CF with alternatives</td>
<td>130</td>
</tr>
<tr>
<td>4-23</td>
<td>State transition mechanism for CF with alternatives</td>
<td>132</td>
</tr>
<tr>
<td>4-24</td>
<td>Helper method to perform state transformation</td>
<td>133</td>
</tr>
<tr>
<td>4-25</td>
<td>Construction of child states for the current state in space search</td>
<td>134</td>
</tr>
</tbody>
</table>
Figure 4-26: State comparison mechanism for cell formation with alternatives.  135
Figure 4-27: Goal switching mechanism for cell formation with alternatives.  ....  136
Figure 5-1: Configuration of space search from FMG.  ........................................  139
Figure 5-2: Part-machine incidence matrix for the 15 parts.  .............................  141
Figure 5-3: Special plant list for Jae. .................................................................  141
Figure 5-4: Possible cell configurations (A, C) obtained. .....................................  145
Figure 5-5 Solution A-1 and A-2 from Table 5-6. ...............................................  148
Figure 5-6: Part-machine incidence matrix for Lee 2 (P1-P22). .........................  150
Figure 5-7: Part-machine incidence matrix for Lee 2 (P23-P40). .......................  151
Figure 5-8: Special plant list for Lee 2. ...............................................................  152
Figure 5-9: Cell configuration A. .......................................................................  154
Figure 5-10: Cell configuration B. ......................................................................  155
Figure 5-11: Best solution for Lee 1 by space search. ........................................  156
Figure 5-12: Best solution for Burbidge by space search. .................................  157
Figure 5-13: Best solution for Kasilingam by space search. .........................  158
1 Introduction

The advent of information technology has lead to an increased interaction between manufacturing industries and computers through the development of enterprise applications. The speed at which manufacturing philosophies like Flexible Manufacturing Systems (FMS) and Group Technology (GT) reach industry depends on how an extensible and user-friendly application for these concepts is made. With the development of manufacturing application software, the amount of time devoted to improving research philosophies can be increased tremendously. The speed with which an organization can adapt to changing market demands at minimum cost determines the extent of its survival. Improving productivity at minimum cost is the primary objective of every organization. As part of this drive, manufacturing systems are moving from mass production systems to traditional group-based systems. In a manufacturing context, GT can be defined as a method for determining similar parts and group them into part families to take advantage of the inherent similarities in design and manufacturing of these parts. The concepts of GT can be applied to men, machines, parts and tool management. The application of GT can result in a complete changeover of the organizational structure and layout of a manufacturing enterprise. One of the primary focuses of GT is Cellular Manufacturing (CM), which can be defined as decomposing a manufacturing system into independent manufacturing cells. The primary focus of this thesis is
directed towards developing an algorithm for cell formation with consideration of alternative process plans in a manufacturing system.

1.1 Cellular manufacturing and Alternative Process Plans

Cellular manufacturing is defined as the process of determining independent manufacturing cells dedicated to producing part families. In order to perform CM, it is important to decide which parts and machines have to be considered for CM. CM may be implemented with existing machines or may be based on the procurement of new machines. One of the important aspects of CM is Cell Formation (CF). CF problem may be defined as: “If the number, types and capacities of production machines, the number and types of parts to be manufactured, and the routing plans and machine standards for each part are known, which machines and their associated parts should be grouped together to form cells?” [22]. The primary advantage achieved out of CM is reduced material handling within the plant floor. The other benefits of CM include reduced throughput time, work-in-progress inventory, shorter lead times, reduced material handling and set-up time [27] with simplified planning and scheduling.

Process planning is the function within a manufacturing facility that establishes which processes and parameters are to be used (as well as those machines capable of performing these processes) to convert a part from its initial form to a final form predetermined in an engineering drawing [5]. In a factory, which produces hundreds of parts and has as many machines, it is common for
parts to have alternative routings of machines. This routing may be through a completely different set of machines or there may be some common machines between the different routings. Modern machines (workstations) are capable of performing many operations. A specific operation can be carried out on different machines with varying levels of efficiency and processing time. These facts lead to the consideration of alternative routings for parts in both the design and operation of cells [23]. Alternative Process Plans play a significant role in cell formation and produce better machine utilization. Alternative process plans are necessary for rapid change and reduced planning and manufacturing cost. Most CF algorithms start with an initial part-machine incidence matrix. An example part-machine machine matrix is shown in Table 1-1 and has 4 parts and 5 machines.

<table>
<thead>
<tr>
<th>Part</th>
<th>Machine</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td>A1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Part 2</td>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Part 3</td>
<td>A1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Part 4</td>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1-1 Initial part machine matrix for the Cell formation problem
Each part has several alternative process plans, marked appropriately with the alphabet ‘A’ in sequence. A value of 1 in the routing for a part indicates that the part has an activity on the corresponding machine. A value of 0 indicates no activity on that machine. For example, Part 1 has three alternative plans, namely A1, A2 and A3. A 1 corresponding to Part 1 A1 and M1 indicates that Part 1 has an activity on M1 if alternative A1 is used. Activities for a part on the same machine in two different routings may signify different work content and these two operations may not be interchangeable. For example, Part 1 has an activity on M1 in routings A1 and A2, but these may signify different work content.

1.2 Artificial Intelligence and Space Search

Artificial Intelligence (AI) is the enterprise of constructing a physical-symbol system that can reliably pass the Turing test [7]. In other terms, AI is the process of creating an intelligent system that can work in the desired way in the domain it was built to work with. A lot of domain specific knowledge has to be built into the system. The important aspects of AI include search, knowledge representation, logic, reasoning and application of these to the domain under consideration. AI has developed several problem-solving techniques. One of these is space search in the solution domain of the problem under consideration. In this approach, decision procedure is seen as selection of a solution that is better by searching the space states of the problem domain. The steps involved
in this include state or knowledge representation, search initialization, state transition for search progression, search filtering and search termination for producing the best solution. Figure 1-1 shows space search of the solution domain for a problem with domain knowledge included in each state.

Figure 1-1 Space search for a problem domain
To perform intelligent search, generally an evaluation function weight is included for each space state. Each state is evaluated on the basis of this weight and the best state (with the least weight) is expanded for search progression. Each circle is a result of the symbol-based knowledge representation of the problem domain and represents a space state. Each state is evaluated by the result of a heuristic evaluation function and the result is called weight of the state. The best state (some decision based on weight) is selected for state progression. Travel in state space is achieved by applying the state transition function on the selected state. If no child states are generated, then the next state to be expanded is selected for expansion. States that do not conform to the decision criteria based on weight are not considered for expansion. This method allows for backtracking. If, during the search, a better state is found, then the goal is switched and further expansion is decided based on the weight of the new goal.

1.3 Motivation

One of the most important issues in Cellular Manufacturing (CM) is Cell Formation (CF). Cell formation is the process of selecting machines and the parts that can be manufactured with these machines from a list of available parts and machines. This problem of finding partitions among a set of machines to completely manufacture a part family that can be completely manufactured with these machines is NP-hard. Most of the existing algorithms for CM don’t consider the effect of alternative process plans on CF. Process Plans for parts in
production represent input data for CF algorithms. However, very often these process plans may have to be modified because of CF. The motivation of this thesis is to study the effect of simultaneous decision making on process plan selection from alternative plans for parts and forming part families and machine cells. Fixed process plans may have to be changed because of the presence of several exceptional parts and machines. A quick look at Table 1-1 does not reveal how alternative plans for parts and machines interact. Application of knowledge based solution to understand the problem of simultaneous decision-making for selecting process plans for parts and forming manufacturing cells is considered to be a good approach. A machine may be needed for a part in one alternative but may not be needed for the same part in a different alternative. Consideration of these two alternatives can lead to two different solutions. Therefore, it is necessary to verify selection of alternatives at each and every step. This kind of step-by-step evaluation of states is well suited with applications of space search algorithms [20]. Because of design and technical constraints, there may be a limit on the size of a manufacturing cell, determined by the number of machines in each cell. Also, the manufacturing cells formed should have no more than a desired number of machines in any cell, at minimum total inter cell traffic. If an efficient heuristic can be built to evaluate the states, then a significant reduction in the amount of solution space explored can be achieved. The importance of designing an efficient heuristic is stated by the significance that it not only helps reduce the amount of space searched but also helps to
direct the search towards an optimal solution. As explained earlier, the result of the heuristic evaluation of a space state is called weight. Knowledge-based representation for a state in the cell formation domain with alternative plans should show the manufacturing cells formed with their parts and machines, the remaining parts and machines, which are yet to be assigned to manufacturing cells and the weight, factor for the state. The weight factor can be a count of the inter-cell traffic for the current state.

1.4 Research objective and goals

From the discussion above, the primary objective of this thesis is to develop a space search algorithm for solving the problem of manufacturing cell formation with the consideration of alternative process plans. The effect of the presence of alternative process plans on the cell formation is studied. The algorithm generates different solutions for manufacturing cell formation and is able to compare and evaluate different solutions. For example, two possible solutions of the cellular manufacturing problem with alternative process plans from Table 1-1 are shown in Table 1-2 and Table 1-3. These states are possible goal states that the space search algorithm should produce. These goal states include information about the manufacturing cells formed with their machines and part families. Ideally, the result will be cells where complete manufacturing of a part can be done. A part has an exception if it has some operation on a machine
that is not in the machine cell corresponding to the part family in which the part is available.

In Table 1-2, part family PF1 has P2 A1, which has an operation on M5. But M5 is not in Cell 1, which is the machine cell corresponding to PF1. This is an exception. This space state has one exception (inter cell traffic), while the space state in Table 1-3 has three exceptions. The space search algorithm
should return the space state shown in Table 1-2 as the better of the two solutions. The decision making on selection of the appropriate process plan and forming the manufacturing cell simultaneously has to be handled by the space search algorithm. The space search algorithm can select different alternative process plans for the same part. The heuristic function can be designed based on available information like quantity of parts to be handled, cost of material handling, manufacturing time, number of exceptional operations etc. In this research, the evaluation function is limited to number of exceptional operations, while other performance criteria mentioned in Section 1.1 are not considered.

A prototype will be implemented for the purpose of testing the space search algorithm. Since the concept of CF deals with many parts and machines, an automated data collection mechanism is needed to create the necessary data, feed it to the space search application and produce the appropriate result. Since space search involves generation of many child states, proper visualization tools are needed so that the user can know at a glance whether the cell configuration produced as a result of the space search algorithm is good or not. The Java programming language Platform 2 was recognized as an appropriate medium for software development. Java Swing and Java Util API’s provide appropriate classes for the purpose of space search and heuristic evaluation. The data that need to be collected are the parts in the manufacturing system, the machines in the manufacturing system, the routing of parts through machines and operation specific information like number of machine units available, the quantity of each
part to be produced, the manufacturing time for each part etc. The collected data needs to be stored so that any modifications in the algorithm can be tested on the same data set without having to create the data all over again. XML was decided upon as an appropriate medium for the storage of part and machine data.

1.5 Application development for Cellular Manufacturing

In the past decade, there has been an exponential increase in the amount of interaction between information technology and manufacturing. Software tools have been developed for a variety of engineering concepts. These automation tools help in the rapid development of algorithms and also evaluation of the same to study efficiency and effectiveness. The process of algorithm development for engineering concepts is followed by generation of appropriate data for the purpose of testing the algorithm. If the time spent on data collection, storage and retrieval can be reduced, then the amount of time spent on algorithm development can be significantly reduced. Also, these tools can be used to evaluate the quality of solutions produced by these algorithms. A lot of algorithms have been developed using a variety of techniques like Operations Research, Genetic Algorithms, Neural Networks and Space Search. Space Search involves searching the solution domain of the knowledge-based problem to find a good or optimal solution. An application to perform space search will help the user better understand the direction of search. Good visual display of the space search tool
will also act as a means of testing the quality of the solution produced by the algorithm. A tool called Space Searcher (SS) was built with Java 2 Foundation classes and the Java Swing API [26] that could perform space search on any given knowledge based problem, provided the problem provides implementation for state initialization, state transition and state ordering mechanism. The dynamic nature of the Java Swing API makes it possible to show different visuals for different types of problems. Also, the input to most Cellular Manufacturing algorithms is details of parts and machines in a manufacturing system. For parts, the information includes routing data, quantity, manufacturing time, manufacturing cost etc. For machines, the machine data could include machine code (SICGE), machine usage frequency, maximum machine hours usable, load etc. This data has to be prepared in a computer understandable format so that it can be fed into the programmed algorithm. A tool called Factory Model Generator (FMG) was built with the Java Swing API to create the data for parts and machines needed for Cellular Manufacturing. The generated data is stored in Extensible Markup Language (XML) format. The reason for this is, once an example factory model is created, the same model can be retrieved for testing different Cellular Manufacturing algorithms. XML is emerging as the best way of performing data exchange between software applications. The advantages of using XML are well discussed in several XML technology books. Some advantages associated with XML are customized markup, self-describing data, and structured and integrated data associated with easy data exchange. Plenty
of XML specific parsers are available in the market to parse XML documents. The FMG in conjunction with the SS create application software for solving the problem of Cellular Manufacturing.

1.6 Structure of Thesis

The research performed as part of this thesis is presented in 6 chapters. Chapter 1 is an introduction to the problem, motivation and goals of this thesis. Chapter 2 discusses the background and related research in Cellular Manufacturing, Space Search and Intelligent Manufacturing. Algorithms developed for CF with and without alternative processes are discussed according to the technique they use to solve this problem. Chapter 3 contains the methodology of this thesis. It contains two sections, first for discussing the CF algorithm without alternative process plans and then for CF algorithm with alternative process plans. Chapter 4 presents the architecture and implementation of the software prototypes developed for solving the problem of CF with consideration of alternative process plans. It also discusses the implementation process of space search algorithm for CF and integration into the software prototypes. Chapter 5 demonstrates the use of the developed prototypes for example representation, saving and retrieval of the collected data and testing of the CF algorithm with alternative process plans. Two test cases are presented from research literature. The results generated by the tools are presented with the needed visuals. Chapter 6 presents a summary of the
research conducted as part of this thesis. It also discusses the future extensions possible for this research work.
2 Background and related work

Many research efforts have been directed towards developing algorithms for solving the CM problem. These algorithms differ in the methodologies used for CM. Different techniques include Operations Research, Genetic Algorithms, Artificial Intelligence and Industrial Applications. A section has also been included on Intelligent Manufacturing and Manufacturing Activity Modeling. Discussion on the Java and XML technologies used as part of this research is also presented in this chapter. A detailed review of research in the field of CM and classification of algorithms based on the solution methodology can be found in [22]. Classification is based on methodologies like Descriptive Procedures, Mathematical Programming, Cluster Analysis, Graph Partitioning and Artificial Intelligence. These are further classified into several specialized techniques. Descriptive Procedures include three parts namely part family identification (PFI), machine group identification (MGI) and part family/machine grouping (PF/MG) which identifies the part families and machine groups simultaneously. Mathematical Programming techniques include Linear Programming (LP), Linear and Quadratic Integer Programming (LQP), Dynamic Programming (DP) and Goal Programming (GP). Artificial Intelligence techniques include Space Search, Genetic Algorithms and Neural Networks. The literature review carried out as part of this research is presented in two sections. The first section includes discussion of CM algorithms, which do not consider alternative process plans for
parts in the manufacturing system. The algorithms are further described according to the different methodologies used. The second section discusses research on CM with consideration of alternative process plans for parts. This section is divided into two sub-sections. The first sub-section describes the algorithms that perform cell formation and selection of alternative process plans as two phases. The second sub-section describes the algorithms available for simultaneous cell formation and process plan selection for parts in the manufacturing system.

2.1 Cellular manufacturing without alternative process plans

This section presents an overview of some of the algorithms that have been developed for solving the problem of CM without considering alternative process plans for parts.

2.1.1 Operations research

Mathematical Programming and Operations Research have been by far the most popular technique used to generate mathematical models to identify the manufacturing cells in a manufacturing system. These techniques also form the basis for several advanced techniques like Genetic Algorithms and Space Search, which work upon these models to generate manufacturing cells.

Kusiak et al [15] developed a branch and bound IP algorithm for solving the problem of Group Technology with consideration of bottleneck parts and
machines. A bottleneck part is a part with operations in more than one cluster or manufacturing cell. A bottleneck machine is a machine that processes a part that has manufacturing activities in more than one cluster or manufacturing cell. Two types of clusters or manufacturing cells called Mutually Separable and Partially Separable clusters are defined. The partially separable clusters give rise to bottleneck parts and machines. The branch and bound algorithm developed here is an extension of the Cluster Identification algorithm (CI) presented by Kusiak et al in [16]. The CI algorithm solves a special case of the GT problem where the machine part incidence matrix is decomposable into mutually separable sub matrices. CI finds optimal machine cells and part families provided that the machine part incidence matrix has the block diagonal structure embedded. Observations are made about the feasibility rules for using the CI algorithm to solve the GT problem. The branching is started with the root node that contains only one matrix. The CI algorithm is applied on a sub matrix of a node. If the sub matrix is decomposable, one single new node is generated. The branching scheme involves definition of a bottleneck measure for each part in a non-decomposable matrix. This bottleneck measure (BM) estimates the likelihood of a part being a bottleneck part, which prevents decomposition on the incidence matrix. It is acknowledged that this may be an approximate solution approach to this problem.

Kusiak [14] introduced an integer programming formulation called as the p-median model to solve the problems involved in visual analysis of clusters for a
part-machine incidence matrix with large number of parts and machines. In the p-median formulation, the number of part families is decided prior to the matrix resolution procedure. The objective function maximized the total sum of similarities between all the parts. The constraints ensure that each part belongs exactly to one family and that the specified number of families is formed. Wang et al [27] introduced a new integer linear programming formulation for cell formation, which has additional desirable features over the original p-median model. The additional features include control on the maximum number of machines in a machine cell or the number of parts in a part family by specifying an upper bound. This reduces the number of constraints in the model formulation. This significantly increased the computational efficiency of the part-family formation procedure.

Harhalakis et al [9] proposed a two-step heuristic algorithm capable of minimizing inter-cell traffic movement. The first step is a bottom-up aggregation procedure to minimize “Normalized Inter-cell traffic”. The second step involves validation of machine assignment to machine cells. This algorithm starts by placing each machine in a cell. The normalized inter-cell traffic between each cell is calculated. The normalized inter-cell traffic between two cells is a measure based on the repetition of machine patterns for the machines in these cells in the routing of all the parts in the manufacturing system. This count for all machines between two cells, multiplied by some factor like the material handling cost between these machines or the quantity of parts that hold these machine
patterns, constitutes the normalized inter-cell traffic. Two cells between which, the normalized traffic is maximum, are aggregated into a single cell, provided the cell size rule in terms of the number of machines is satisfied. The remaining cells are left untouched at this stage. This procedure attempts to minimize inter-cell traffic and maximize intra-cell traffic. Merging of cells is carried out until no further merging can take place or the traffic between existing cells is zero. The input to the algorithm is a lower triangular matrix showing the traffic between machines. In the second step, as part of the local refinement procedure, each machine is considered as a separate entity and its traffic with each of the existing cells is evaluated. The machine is then assigned to the cell with which it has the maximum traffic.

Nagi et al [21] proposed a linear programming model for solving the cellular manufacturing problem with consideration of multiple part routings and multiple functionally similar work centers. The projected production requirements are also taken into account when forming manufacturing cells with the demand being distributed evenly between multiple routings for better cell design. The problem of routing selection and cell formation is solved simultaneously with this model and the common goal is to minimize total inter-cell traffic. The total inter-cell traffic between two cells is defined as the total exchange of the parts between the cells factored by the weight and quantity of each part. The weight of each part may be a factor based on the combination of material handling and part costs. This value is normalized with the cardinalities of the two cells under
consideration. An initial partition of cells is created with each work-center in one cell, the vector of production volumes using appropriate routings to minimize traffic is found and this vector is used to create a new partition of cells to minimize the traffic again. Validation of work-center assignment to cells is performed with an enhancement to the ICTMM algorithm from [9]. Two cells, between which the inter-cell traffic is a maximum, are aggregated to form a single cell in phase one. Each aggregation is followed by a validation step that ascertains the significance of the assignment of a work-center to a cell. This aggregation and validation procedure is continued until no further cells can be formed or the existing traffic between the cells is zero.

Lee et al [19] developed a multi-criteria weighed approach to form machine cells and part families. The parameters considered include cycle demand, batch size, pallet size, routing sequences, processing time, machine capacities, and work load status of machines. A weighted approach is designed to measure both part movement between machine cells and balancing workload between duplicated machines. Each part is assumed to have one unique routing sequence. Machines are partitioned into unique and duplicated set. The unique set has only one unit of each machine type while the duplicated set contains machines that include two or more identical units for each machine type. The mathematical formulation has an objective function for minimizing normalized inter-cell traffic. The reasoning for including normalized inter-cell traffic is to reduce inter-cell and intra-cell traffic. Constraints for this objective function
include rules for cell size limit, workload capacity for each machine and workload balancing on each identical machine unit type. A criterion for measuring cell formation is the weighted sum of two measures called Group Technology (GT) measure and Workload Balance (WB) measure. A higher weighted value of GT measure indicates that the cell formation is better in terms of reduced inter-cell traffic. A higher weighted value of WB measure indicates more workload balance among duplicated machines than a weighted measure with a lower value. A three-phased approach for cell formation was proposed. During the cell formation process, the inter-cell movement and workload balance are optimized based on the weighted criteria. The first phase developed estimation criteria to determine workload balances (WB) for duplicated machines. In the second phase, machine cells and part families were constructed based on minimal inter-cell traffic and estimation of workload balance. The last phase was a heuristic procedure to improve cell formation with an objective to maximize the weighted measure. The considered techniques include machine reassignment, machine exchange, part reassignment and part exchange.

The knowledge about mathematical models for the cell formation problem gained from these papers formed the basis of the mathematical model for the space search algorithm for cell formation with alternative plans discussed in Section 3.2.1. Specifically, the modeling aspects for introduction of alternative process plans into cell formation model, Calculation of intercellular traffic from exceptional operations, introduction of constraints for machine cell size and
design of optimization function were built from the experience gained from these papers.

2.1.2 Artificial Intelligence

Different techniques from Artificial Intelligence have been used to solve the problem of cellular manufacturing. Three popular techniques, Genetic Algorithms, Neural Networks and Space Search are discussed in this section.

2.1.2.1 Genetic Algorithms

Genetic algorithms (GA) are based on the biological principle of natural selection of the fittest. GA maintains a family of solutions when searching for an optimal solution. A solution is represented as a chromosome. The gene or a chromosome is a string, which is a code for machine group and/or part family. This string can hold a group of binary numbers, real numbers, integers, symbols etc. A fitness function is designed to evaluate each individual or chromosome. A series of generations is carried out on the population with the fitness function acting as the evaluation and the filtering criteria. Reproduction is carried out by the application of genetic operators on the population. The most commonly used operators are mutation and crossover. Mutation operators alter one parent by changing one or more gene. Crossover operators combine information from two parents and incorporate them into the child. After a series of generations, the genes are decoded to arrive at the part families and machine cells formed. The six fundamental issues that need attention for developing a genetic algorithm are
chromosome representation, population initialization, selection strategy, genetic operators, termination criteria and evaluation measures [12]. Joines et al [12] proposed an integer-programming model for cellular manufacturing solved using genetic algorithms. Each gene in a chromosome is an alphabet consisting of the integers between the variable’s upper and lower bound. The bound is the range of part families/machine cells that the system should have. Random seeding was used to initialize the population. Normalized geometric ranking scheme was used to select chromosomes for further generation based on the values of their fitness functions. Mutation and crossover operators were used for reproduction. Grouping Efficacy that seeks to minimize number of exceptional elements and number of voids in the block diagonal structure was used as the evaluation criterion.

Chan et al [4] developed an integer-programming model that took the number of exceptional elements outside the diagonal blocks and the number of voids inside the diagonal block as a measure of fitness. They also introduced a genetic algorithm to solve this mathematical model. The objective function was to minimize the number of voids inside each diagonal block and the number of exceptional elements outside the diagonal blocks. An integer number that indicates its sequence position represents each gene in the chromosome. This identifies the corresponding machine or part. The value of the gene indicates the manufacturing cell to which the machine or part is assigned. Fitness is evaluated as a ratio between the objective function value of each chromosome and the best
(lowest) objective function value for each generation. The two-point crossover operator was used for reproduction. One cross point was selected randomly within the genes representing machines and parts. The generation is continued until there is very little change in the best value of the objective function for a specified number of generations.

2.1.2.2 Neural Networks

Nallan Suresh [25] discussed the different kinds of Artificial Neural Networks (ANN) and their application to solve the problem of Cellular Manufacturing. The pattern recognition capability of Artificial Neural Networks is used to solve the cellular manufacturing problem. Based on the direction of signal flow, two types of neural networks called feedforward and feedback network are presented. In the feedforward method, signal flow is unidirectional from the input layers to the intermediate layers to the output layers. In the feedback network, signal can flow from the output of any neuron to the input of any neuron. Based on the type of learning adopted, neural networks are classified into supervised and unsupervised learning. In the former type, the network is trained with the input as well as correct output is presented to the network. In the latter, the network has no knowledge of the correct output. The network clusters similar input vectors into distinct classes depending on similarities. An exemplar vector is used to represent each class. The exemplar vector is updated in response to a new input that is found to be similar. Three application areas for using artificial neural networks for cellular manufacturing
were identified as 1) Clustering based on design features, 2) Clustering based on routings data and 3) Other classification approaches for part and tool grouping. The paper mainly focuses on artificial neural networks for cellular manufacturing based on clustering of routing data. The paper classifies the unsupervised methods for cellular manufacturing for clustering based on routing data as 1) competitive learning model, 2) interactive activation and competitive learning, 3) adaptive resonance theory (ART), 4) self organizing feature map, and 5) fuzzy ART. The ART model is an improvement over competitive learning and interactive activation models, which uses vigilance measure and stability properties. Fuzzy ART model is an improvement over ART. It can handle analog and binary inputs. This is also based on unsupervised learning. Two application areas for ANNs in context of cellular manufacturing are also presented. Capacitated cell formation with consideration of part demand volumes, machine capacities, multiple machine units, alternative process plans etc. It is acknowledged that neural networks alone cannot be used to solve all of them, but they can be used to form families and then goal programming can be used to optimize the result. Sequence dependent clustering is also proposed as another prospective research area for consideration of ANNs for cellular manufacturing.

2.1.2.3 Space Search

Ghosh et al [6] presented a branch and bound space search algorithm for cellular manufacturing. It is acknowledged that the problem of identifying an optimum partition of machines in a plant floor with minimum inter cell traffic is
NP-hard. This algorithm starts with an upper bound on the solution and tries to improve the solution by state-space search of the solution domain of the problem. The upper bound is calculated with a fast heuristic called Inter-cell traffic minimization method (ICTMM) [9]. The lower bound is found based on relaxation of merging technique and is used for state ordering and state pruning. Search is started with one machine in one cell. Cells are merged based on the minimization of inter-cell traffic and the best state is selected for search progression. This algorithm was used for testing purposes as part of this research and is discussed in further detail in Section 3.1.

2.1.3 Industrial Applications

Burbidge [2] introduced Production Flow Analysis (PFA) as a technique to plan the change of a factory from process layout to product layout. In context of jobbing and batch production, he defined PFA as a technique for identifying part families and machine cells for Group Technology (GT). PFA consists of a sequence of steps that have to be performed for a factory to move from process organization to product organization. PFA starts with Company Flow Analysis (CFA), which tries to simplify the flow between factories. Then, the best division of each factory into divisions is found with Factory Flow Analysis (FFA). FFA simplifies the material flow between different departments by finding the best division of departments for the factory to move towards product-based organization. PFA then plans the division of departments into groups with Group
Analysis (GA). The flow of materials between key workstations in each group is found with Line Analysis (LA). Finally, Tool Analysis (TA) is performed to find tool families. The output of PFA is a list of groups or manufacturing cells, lists of machines and other facilities that have to be included in each group, and the list of parts or part family that have to be manufactured in each group.

Group Analysis is a sub-technique of PFA that divides the major groups and major families found by FFA into subsets known as groups and families. GA performs resolution of a part machine incidence matrix as shown in Figure 2-1 into groups.

![Figure 2-1: Part machine incidence matrix from Burbidge [2]](image)
A part of the procedure is the classification of machines by type and ranking them in accordance with their importance to group formation, combining parts to form modules so that majority of parts in each module will fit as a group and planning the method of combining the modules to form groups in a number of independent steps. The data needed for group analysis include data on machines (type, name, number of units, SICGE code), data on parts (number, name, material, material form), data about the operational methods etc. Burbidge devised an algorithm for machine classification based on operation transferability and number of units of each machine available and called it SICGE coding. The letter ‘S’ in SICGE stands for special category machines. They are used to perform special operations that cannot be transferred to other machines. Also, only one unit of this machine is available. The letter ‘I’ in SICGE stands for intermediate machines. These are also similar to ‘S’ type machines but there are more than one unit of this machine type. These machines are also used for operations that cannot be transferred to other machines easily. The letter ‘C’ in SICGE stands for common machines. There is normally more than one of each of these machines. They are used on many different parts and are generally needed in more than two or more groups. The letter ‘G’ in SICGE stands for General category machines. They are used for operations on a large number of manufactured parts or they may be dangerous requiring special precautions for use. These machines are very few in number. The letter ‘E’ in SICGE is used for Equipment type machines like benches, surface plates, and some equipment for
manual operations. GA forms small modules based on key machines and builds the groups around these machines. Each module consists of a set of parts, together with the machines needed to manufacture the parts. Figure 2-2 shows a module summary of the different modules formed as part of group analysis.

Figure 2-2: Module summary from Burbidge [2]
The modules are later combined to form groups. The key machines are found from a plant list called Special Plant List (SPL), which keeps the machines in a plant floor, sorted in order of increasing frequency of usage of machines. After the modules have been found, groups are formed. Emphasis is laid on finding part families that can be completed using only its complementary set of machines. Group analysis takes advantage of the presence of multiple units of a machine to distribute load between machines. Figure 2-3 shows the result of group analysis on the part-machine incidence matrix shown in Figure 2-1.

Figure 2-3: Resolution of part-machine incidence matrix from Burbidge [2].
Group selection may result in different groups like heavy machining groups, heavy rotational groups, heavy shaft group, gear group, hardened and ground steel shaft group, steel non-rotational groups, CI non-rotational groups, bar lathe group etc.

### 2.2 Cellular manufacturing with alternative process plans

Research efforts in the direction of cellular manufacturing with consideration of alternative process plans have been limited. It is difficult to achieve a clear demarcation between the different techniques that have been used to solve this problem. Two techniques used to solve the problem of cellular manufacturing with consideration of alternative process plans are the two-phased approach and the simultaneous approach. In the phased approach, the problem of route selection for parts is done in one step and the problem of forming cells is done in another step. In simultaneous approach, the selection of process plans for parts from alternatives and the formation of manufacturing cells are performed simultaneously. Some of the research effort in this direction is presented in this section. Simultaneous process planning and manufacturing cell formation makes the problem very complicated and generation of good results depend on the efficiency of the algorithm. Phased approach may not utilize the advantage offered by alternative process plans in cell formation.
2.2.1 Phased approach for cellular manufacturing

Hwang et. al. [11] proposed a two-phased cell formation algorithm with consideration of alternative process plans for parts. In the first stage, the problem of route selection is solved with the objective of maximizing the sum of compatibility coefficients among the selected process plans. The compatibility coefficients are designed in such a way that highly similar and highly dissimilar process plans get high coefficient values. They suggested the use of two types of compatibility coefficients called Type 1 and Type 2 coefficients. Type 1 coefficients are developed based on Jaccard’s measure for the similarity coefficients. Type 2 coefficients are developed based on the initial part-machine incidence matrix. The objective function is to maximize the sum of compatibility coefficients of all pairs of process plans in a covering set (one part-one routing principle). A covering set with high values of these compatibility coefficients is expected to form a completely block diagonal structure. In the second stage, part families are formed based on the generated covering set and machines are assigned. Part families are formed based on p-median model with the objective of maximizing the sum of similarity coefficients of part pairs in the same family. Once the part families are formed, machines are assigned to families so as to reduce the number of exceptional elements and the number of in-block blanks.

Sundaram et. al. [24] proposed a three-step algorithm for forming part families for parts having alternative process plans. The three steps in the
algorithm include 1) Formation of part families, 2) Assignment of remaining parts, and 3) Assignment of residual remaining parts. Two factors called ADDOP and LIMIT are defined to help in the formation of part families. ADDOP is the maximum number of additional operations a key part in a family can have to accommodate additional parts in a family. LIMIT value is used to avoid the formation of a super family where there exist parts requiring all or most of the operations. Parts that are not assigned to any family are called unassigned parts. Parts that cannot be assigned to any family because of violation of ADDOP, are called remaining parts. In the first step, initial families are formed based on key parts. The parts are arranged in descending order of the number of operations required. The first part in the list forms the first family. The remaining parts in the list are compared to the current part and based on the value of ADDOP, are assigned to the current family or to the remaining parts list. At the end of the three steps, part families are formed.

2.2.2 Simultaneous process planning and cellular manufacturing

Caux et. al. [3] proposed a combined approach for solving the problem of alternative process planning and cellular manufacturing simultaneously. A simulated annealing approach is used to solve the problem of route selection for parts from alternative plans and a branch and bound algorithm is used for solving the problem of minimizing intercellular traffic. The objective function is to minimize intercellular traffic subject to the constraints of machine cell size, one
routing for one part principle and machine loading. The simulated annealing algorithm produces machine partitions based on the cell size constraint. For a given partition, the branch and bound algorithm attempts to select one process plan for each part from a list of alternative plans such that the intercellular traffic is minimized. A solution from the simulated annealing algorithm is represented as a vector $v$ where $v_i$ indicates the cell number to which machine $i$ belongs. A neighbor for a solution is obtained by applying insertion or permutation techniques. Insertion involves moving a machine from one cell to another, while permutation involves swapping two machines between two cells. The branch and bound algorithm assigns one routing for each part for a given partition of machines. This assignment must satisfy the machine capacity constraint and also minimize the intercellular traffic. The solution is enumerated in the form of a tree structure. Each level of the tree represents a part and each node represents an alternative routing for that part. This enumeration tree is further pruned with constraints for machine capacity and a minimum lower bound on the intercellular traffic value. The lower bound is computed by relaxing the machine capacity constraint.

2.3 Knowledge-based intelligent manufacturing

This section describes knowledge-based systems developed for manufacturing. It focuses on manufacturing activity modeling, knowledge-based group technology systems and generation of alternative process plans.
2.3.1 Manufacturing activity modeling

Sormaz [23] developed a manufacturing model for intelligent information integration. Figure 2-4 shows the basic product planning functions.

![Diagram showing product planning functions](image)

Figure 2-4: Product planning functions from Sormaz [23].

Various production management activities involved in product development and manufacturing are discussed in detail with emphasis on their contribution to concurrent engineering. The need for integration of various production activities during a product development cycle is also discussed. Product development cycle is seen as a series of answers to simple questions: Why? Where? When? How? Who? and What? These questions relate to the
different activities involved in product development. Each of the tasks represented by these questions involves a lot of back end engineering tasks, which have to be completed for each question to be answered correctly. These activities cannot be solved independently and involve a certain degree of feedback or overlapping. Overlapping functions share certain common process planning tasks between them and are viewed as necessarily leading to integration between these tasks. A manufacturing activity model that was developed that demonstrated the interaction between process planning and other activities involved in product development like design, scheduling, feature recognition and resource planning to name a few tasks. The proposed manufacturing process model consists of three dimensions, time, variety and aggregation as shown in Figure 2-5.

Figure 2-5: Manufacturing process model hierarchy
The manufacturing process model allowed for consideration of alternative process plans. The basic entity of the manufacturing process model is a process or activity. Each of the activities defined is associated with its attributes, which have to be defined before the activity is executed. Each manufacturing process is related to other processes with these three layers. The time dimension is used to establish temporal relation between several processes of the same type. This representation may be used in a feature precedence network. The variety dimension is used to describe different processes generated within a certain level. This variety may be because of alternative process plans for parts or a set of operations for machining a set of features. The aggregation dimension is related to the various scopes of the planned processes. It helps to distribute planning tasks among different functions and the functions can concentrate just on the relevant attributes. These dimensions cannot be implemented independently. The underlying basis for the manufacturing process model is the process plan network. This network has four layers namely, feature layer, process layer, tool orientation layer and machine layer. The feature layer is a network of machining features called Feature Precedence Network (FPN). The process layer is made up of process instances of different features. The tool orientation is a set of cutting processes that can be performed with similar tool orientation and machine setup. The machine layer is used to represent different sets of processes traversed through these machines. A salient feature of this model is its ability to generate process plans dynamically corresponding to
changes in design, machine status etc. The changes can be reflected through the network. The architecture of Process Plan Representation Model (PPRM) developed as part of this research effort is based on this manufacturing model.

2.3.2 Knowledge-based group technology systems

A product development cycle involves a lot of complex manufacturing activities, which have to be synchronized in order for the different objectives and demands of the industry to be met. The knowledge of domain experts need to be captured into a system which can help in the different activities involved in a product development cycle. Such a system is called as knowledge-based system. The core of a knowledge-based system consists of a set of rules, which define the problem domain under consideration. The knowledge of domain experts is captured and stored in the database. This data interacts with the rules through an inference engine to assert new data or facts.

Kusiak [17] developed a knowledge-based system for group technology called as KBGT. The important components in the KBGT are database, knowledge base, inference engine, request processor and the clustering algorithms used to form machine cells and part families. KBGT was implemented in common LISP. The input data for KBGT are the machine data and the part data along with optional data about the number of machines in a cell or the maximum number of trips that could be handled by a material-handling carrier. When the KBGT is started up, the data about parts and machines is initialized in
the database as objects. Then, the KBGT system forms machine cells and part families. Each machine is analyzed by the knowledge-based subsystem (KBS) before it is included into a machine cell. The knowledge-based subsystem tests whether constraints for maximum machine loading, upper limit on the material handler frequency and the machine cell size and depending on the result decides upon including a machine in a cell. The knowledge-based subsystem has three components: knowledge base, inference engine and request processor. The knowledge base is a repository for system knowledge obtained from domain experts. The knowledge base in KBGT has three types of rules: preprocessing rules that deals with the initialization of objects for parts and machines, current machine rules which evaluate the assignment of each machine to a cell and machine cell rules which check for cell level constraints. The inference engine attempts to infer or assert facts after a set of rules are fired in the considered context. The inference engine of KBGT employs a forward chaining control strategy. The request processor is used to handle interaction between the clustering algorithm and the knowledge-based system. Based on the requests from the clustering algorithm, the request processor selects the set of rules to be fired by the inference engine. Finally, the clustering algorithm is a modification of the cluster identification algorithm of Kusiak [16]. As an overview, data about the parts and machines in the system is initialized in the database as objects. The knowledge base defines the rules (preprocessing, current machine and machine cell rules) that are fired during different phases of the cell formation process. The
request processor acts as an interface between the clustering algorithm and the inference engine helping decide the class of rules that have to be fired for a given context. The knowledge base, the inference engine and the request processor together form the knowledge-based subsystem. KBGT outputs the machine cells formed, part waiting list, unused machines, bottleneck machines and cell size.

2.3.3 Generation of alternative process plans

Sormaz [23] proposed a process plan network hierarchy for generation of process plan data. The proposed network has four layers: feature layer, process layer, tool orientation layer and machine layer. The feature layer is used to represent a network of machining features. The nodes in the layer are the machining features and the arcs implement the precedence constraints between the different features. This layer is called the Feature Precedence network (FPN) and its generation is the first step in process planning procedure. The next higher layer is the process layer. Process layer consists of manufacturing process instances with which the features from the feature layer are to be machined. The precedence constraints in the feature layer are also carried over to the process layer so that a manufacturing process for a feature that has to be machined before another manufacturing process for a feature does not interchange positions. The next level of classification is the tool orientation layer, which has cutting processes, which can be carried out with the same tool orientation and the same machine type. This layer also inherits precedence constraints from its
parent layer. The final layer of the process plan network is the machine layer, which has sets of process activities from the tool orientation layer for its nodes. The feature precedence constraints are again inherited and implemented in the form of arcs between nodes. The generation of the process plans network starts from the stock and only the features that can be processed on the stock are considered. Process candidates for the selected features are clustered through the process plan network. The result of these steps is presence of alternative operations on machines. For each of the selected operations, remaining feature candidates for the next stage are selected from the feature precedence network and the unused features from the current operation. This procedure is repeated until all the features from the feature precedence network are used up. After the completion of the process plan network, any node from the beginning of the network to the end of the network represents a complete process plan for the part under consideration. Different paths indicate alternative process plans for manufacturing the same part. Two algorithms based on state space search and network algorithm were presented to select the best process plan in terms of the minimum processing time or minimum cost for manufacturing the part. For space search, each space state in the search space represented a path in the process plan network from the start node to the current node represented by the state. Evaluation was based on A* approach where the state space included the manufacturing cost from start state to the current state for manufacturing the part and an estimate of the manufacturing cost from the current state to the end state.
2.4 Overview of Java and XML technologies

This section summarizes the technologies used for the purpose of developing the software prototypes used to solve the problem of cell formation with alternative process plans for parts.

2.4.1 Java technologies

Object-oriented programming is a programming methodology that concentrates on design and analysis of real world objects. Three important concepts in object-oriented programming are inheritance, polymorphism and encapsulation.

- Inheritance

Inheritance is the mechanism by which a child inherits attributes from its parent. For example, a child can inherit behavioral characteristics like way of walking, speaking etc from its parents. Inheritance can help in code reuse for defining same functionality or performing minor changes in functionality.

- Polymorphism

The same object can behave differently under different circumstances. For example, a generic animal class can be defined to model all animals in the universe. Specialized subclasses can inherit from the animal parent to model different species of animals like dogs, cats etc.
• Encapsulation

Encapsulation is the mechanism by which data about an object is protected from outside world. Each Java class hides its information from outside world to prevent data corruption and enhance security. Data of a class should be modified only through accessor and modifier methods as defined by the Java beans style of programming.

Java is an object-oriented programming language that is robust, secure, and portable and platform independent [29]. Java language code gets compiled into bytecodes. The bytecodes are the machine language equivalent which tells the Java virtual machine (JVM) what to do. Java classes are platform independent. This means that an application that is built on one platform is expected to function equally well on a different platform. This is because java code gets compiled in platform independent bytecode. In Java, classes interact through interfaces. An interface defines a behavioral protocol to be implemented by any implementing class in the following circumstances [29].

• Capturing common functionalities between unrelated classes without forcing unnecessary class relationship.

• Revealing an object’s programming behavior without exposing the object class.

• Declaring behavioral methods that one or more classes implement.
2.4.2 XML technologies

Extensible markup language (XML) is a simple and structured text format for data storage and exchange on the web and between software applications [28]. XML originated from Standard generalized markup language (SGML). Its simple text data make it very useful data transfer mechanism for the web and it is emerging as the “de facto” mechanism for data generation, storage and exchange between distributed software applications. XML stores not only the data but also information about the data in the form of markup. The advantages of XML are widely discussed in several books and research articles. Some of the advantages of XML are summarized as follows [10]:

- Easy data exchange

The text data in XML format makes it the most suitable mechanism for data generation and storage. The availability of standard parsers for parsing XML documents make it all the more easy to reuse the parsed XML data.

- Customized markup language

XML is a markup language that can be customized by the people who want to use XML. Unlike HTML, which is a fixed format markup language, XML can have user-defined tags which have information about the data. This allows for generation of custom data.
• Self-describing data

XML data is self-describing because it has information about the data stored in the XML document. Generally, the tags of an XML document can contain attributes and additional tags that can hold a lot of information about the data being presented.

• Structured and integrated data

XML data is structured because it can maintain a hierarchical parent-child relationship in the XML document. XML data is integrated or complete because it has all the features needed to completely represent data.

• Separation of data and presentation layers

XML data can be rendered with style sheets. Different style sheets can be used to render different parts of the XML document. The separation of data (XML) and presentation (Style sheets) makes it a very flexible language for data presentation.

2.4.3 Components of an XML document

An XML document consists of custom elements with each element having its own attributes and sub elements. The XML document can conform to rules set in a DTD (Document type definition) or a Schema definition. Figure 2-6 shows an XML document developed with XML Spy Software.
This document holds customer information in the form of an XML document. The first line in the file indicates the current version of the XML document. It is also called as a processing instruction, since it comes outside the root element. The attribute stand-alone in the first line indicates that the XML document is not based on a DTD or a schema. A piece of information enclosed by opening and closing markup tags is called Element [28]. A root element is the first element in the XML document and all the other elements in the XML document come under the root element. In this example, the root element is <customer> enclosed in markup. Elements can have their properties defined in the form of Attributes. The <address> element has an attribute called “type”
which is used to identify if the address is home address or office address. This name-value combination makes XML more understandable to programmers as well as easier processing by computers. The root element <customer> has three child elements namely, <name>, <address> and <phone>. Each opening child element tag has to be matched by a corresponding closed element tag and these tags should appear in the reverse order of the opening tags. This syntax has to be followed through out the XML document for it to be well formed.

2.4.4 XML transformation with XSLT

XSL stands for XML style language. W3C defines XSL as a language for expressing style sheets. It has three parts, XSL Transformations, a language for transforming XML documents, XML Path (XPath), a language used by XSLT to refer to parts of XML documents, and XML Formatting Objects (XML FO), an XML language for specifying formatting semantics [28]. With XSLT, the same XML file can be viewed with different styles and formats. Also, different parts of the XML document alone can be displayed with XPath. An XSLT file defines rules for transforming a source tree into a solution tree. This is achieved by associating patterns in XML documents with templates in XSLT. The templates tell the XML document how the elements have to be displayed. This feature of XML combined with the object-oriented nature of Java, form a very powerful tool for the development of data generation, and storage and exchange mechanism
for software applications built in Java. Figure 2-7 shows the transformed XML document of Figure 2-6 as rendered by a web browser.

Srinivas Narayan Rajaraman

B5, Mill Street
Athens, Ohio, 45701

Ph: 740-594-4394

Figure 2-7: A XSLT rendered XML document.
3 Methodology

This section describes the problem formulation and methodology of the cell formation algorithms with and without alternative process plans. It is divided into two main subsections; the first sub-section discusses a simplified form of the space search algorithm for cell formation proposed by Ghosh et. al. [6], the second section discusses the space search algorithm developed for cell formation with consideration of alternative process plans developed for this thesis. The reasoning behind the implementation of the space search algorithm without alternative process plans was to get a clear understanding of the different aspects of space search for cell formation. The different aspects of space search for cell formation were identified as problem formulation, state-space representation, state ordering and transition, state filtering, heuristic evaluation function for the states, best search technique, and search termination. The problem-formulation describes the mathematical model of the given problem as a basis for space search algorithm. The state-space provides representation of the data necessary for each state in the problem domain. In context of space search for cell formation, this representation includes details of the cells currently formed, the remaining parts and machines which need to be assigned to manufacturing cells, current value of intercellular traffic between the different cells etc. The section on state ordering and transition discusses the ordering techniques used to sort the space states in order to produce good results in a
reasonable amount of computational time. Ordering is based on the weight of each state. The state transition mechanism discusses the transition mechanism for search progressions. The transition function produces new and valid child states that need to be evaluated for quality of the solution. The state filtering mechanism is used to evaluate the child states produced by the state transition function. Only those states that pass the heuristic evaluation test are considered for further search. The best search technique selects the best available state based on the heuristic evaluation function for further expansion. Finally, the best search has to be terminated to produce a result from the space search performed.

3.1 Cellular manufacturing without alternative process plans

This section discusses the space search algorithm for cell formation proposed by Ghosh et. al. [6] without considering alternative process plans for parts.

3.1.1 Problem formulation

The problem formulation presented here is not a new formulation and exists in a lot of research papers that attempt to solve the problem of cell formation. A manufacturing system with a set $M = \{m_1, m_2, m_3, \ldots, m_m\}$ of $m$ machines and a set $P = \{p_1, p_2, p_3, \ldots, p_p\}$ of $p$ parts is considered. Each machine is considered to be a unique identity. Let $q_i$ represent the desired quantity for
part type \( p_i \). Let \( c_i \) represent the cost associated with the production of one unit of part type \( p_i \). Each part can be manufactured with one routing. A routing for a part is defined as a sequence of \( v \) operations on a subset of \( M \) through, which a part has to pass to be completely manufactured. The routing associated with part \( p_i \) is represented as \( r_i \). 3-1 shows the routing \( r_i \) for a part \( p_i \).

\[
r_i = \{m_i^1, m_i^2, ..., m_i^v\}
\]

where

\[
v \text{ is the number of operations for part } p_i \text{ and each } m_i \in M.
\]

Processing times and setup times are not considered in this algorithm. Because of practical considerations, the number of machines in a cell is usually restricted to a certain value, say \( N \). For each \( p_i, m_i, m_k \in P \times M \times M \), \( d_{ijk} \) is defined as the number of times \( m_i \) follows \( m_k \) or \( m_k \) follows \( m_i \) in the routing \( r_i \). Then, for each pair \((m_i, m_k) \in M \times M \), the traffic \( t_{jk} \) between \( m_i \) and \( m_k \) is defined by 3-2, 3-3 and 3-4 as follows.

\[
t_{jk} = t_{ij} \quad (3-2)
\]

\[
t_{ij} = 0 \quad (3-3)
\]

\[
t_{jk} = \sum_{i=1}^{n} c_{q_i} d_{ijk}, \quad \text{for } j \in \{1,2,\ldots,m\}, k \in \{1,2,\ldots,m\} \quad (3-4)
\]
3-2 states that the traffic between machines \( m_j \) and \( m_k \) is the same as the traffic between machines \( m_k \) and \( m_j \). 3-3 states that the traffic between the same machines is zero. 3-4 gives the value of the traffic between machines \( m_k \) and \( m_j \) in terms of the cost and quantity of all the parts that have operations on \( m_k \) and \( m_j \). This traffic computation between the machines forms the core of the traffic computation between the different manufacturing cells in the system. A partition of \( M \) is any set \( C = \{C_1, C_2, C_3, ..., C_w\} \) of \( w \) cells such that

\[
C_i \cap C_j = \emptyset, \text{ for } i, j \in \{1, 2, ..., w\}, |C_i| \leq N, |C_j| \leq N, i \neq j.
\]

If \( C_i \) and \( C_j \) are two cells in a partition \( C \), then the traffic between \( C_i \) and \( C_j \) can be calculated based on 3-4 as shown in 3-5.

\[
F(C_i, C_j) = \sum_{M_r \in C_i, M_s \in C_j} t_{rs}
\]

(3-5)

where

\[
F(C_i, C_j) \text{ is the traffic function for cells } C_i \text{ and } C_j.
\]

3-5 says that the total traffic between two cells is the summation of the traffic between all the machines present in the two cells. Since \( t_{ik} = t_{kj} \) from 3-2, the traffic between states \( C_i \) and \( C_j \) is equal to the traffic between cells \( C_j \) and \( C_i \) as shown in 3-6.

\[
F(C_i, C_j) = F(C_j, C_i)
\]

(3-6)

The total inter cell traffic for \( C \) is shown in 3-7.
\[
F(C) = \sum_{i \neq j} F(C_i, C_j)
\]  

(3-7)

3-7 states that the total traffic for a cellular manufacturing system represented by \( C \) is the sum of the intercellular traffic between the different manufacturing cells in the system. The combination of all the different partitions of machines for the manufacturing cell problem is represented as \( U = \{A, B, C, D, \ldots\} \). The manufacturing cell optimization problem is the problem of finding the optimum cell configuration \( C^* \) from \( U \) such that the condition specified by 3-8 is satisfied.

\[
F(C^*) = \min F(C), \text{ for all } C \in U
\]  

(3-8)

subject to, \( \forall C, \) cardinality of cells \( (C) \leq N \).

3.1.2 State space representation

A state in the search space is a collection of machine cells, each cell containing a maximum of N machines. Each state is accompanied by a matrix of the values of the intercellular traffic between the different cells in the current state. Each cell is marked with a label saying “Open” or Closed. If a cell is marked “Open”, it means that the cell can be merged with other cells or machines can be added to the cell without violating the cell capacity constraint. If a cell is marked “Closed”, then the cell cannot be merged with other cells and machines cannot be added to the cell. The state-space representation for the
problem of cell formation without alternative plans for parts is shown in Figure 3-1.

The search is started with each machine being allocated to a single cell. The number of cells in this state is equal to \( m \), the total number of machines in the manufacturing system. Each cell is marked “Open” indicating that all cells in the initial state are mergeable with the other cells. Figure 3-1 lists the cells available in the initial state and the machines available in each state. The traffic matrix for each state-space is symmetric, since \( F(C, C_j) = F(C_i, C) \) from 3-6. Figure 3-2 shows the intercellular traffic distribution between the different manufacturing cells for the initial state of the search space.
The calculation of the intercellular traffic is based on assumed values for quantity of each part and the manufacturing cost of each part and is shown here for illustration purposes. The intercellular traffic matrix is symmetric and either the upper half or the lower half can be used for computation. Additional data like cardinality of each cell can be inferred from the state-space representation. A color-coding scheme is used to distinguish “Open” and “Closed” cells. “Open” cells are marked with green color and “Closed” cells are marked with red color.

### 3.1.3 State transition mechanism

The transition mechanism creates $m$ successor states for the initial state. These correspond to the merging of Cell1 of the initial state with all the other cells in the initial state, one at a time to produce $m-1$ successor states. In addition, if it is decided not to merge Cell1 with any other cell, it is marked as “Closed” and corresponds to the $m^{th}$ successor of the initial state. If after merging two cells, the new cell’s size is less than or equal to $N$, then the cell is marked...
“Open”. Otherwise the cell is marked as “Closed”. Figure 3-3, Figure 3-4, Figure 3-5, Figure 3-6 and Figure 3-7 represent the child states of the initial state.

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell 1</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 2</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 3</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cell 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Cell 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 3-3: Child 1 of initial state.

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell 1</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 2</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 3</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cell 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Cell 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 3-4: Child 2 of initial state.

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell 1</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cell 2</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Cell 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Cell 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 3-5: Child 3 of initial state.
Figure 3-6: Child 4 of initial state.

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Cell 1</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-7: Child 5 of initial state.

<table>
<thead>
<tr>
<th>Name</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 1</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Child</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Cell 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Figure 3-7 corresponds to the \( m^{th} \) child state where the current cell under consideration cannot be merged with other cells based on some decision criteria.

The criteria for deciding if a cell can be merged with another cell in a space state is decided as shown in the next procedure.
Let $C_i \{M_{k_1}, \ldots, M_{k_p}\}$ be the first cell in a state $S$ that is marked open. Let $C_j \{M_{l_1}, \ldots, M_{l_p}\}$ be the next open cell in the state $S$, such that $j > i$, and $C_i$'s largest machine index $kp$ is less than $C_j$'s smallest machine index $kl$. This condition is used to prevent the same partitions for repeating in the search space. If this condition is satisfied, then $S$ has a successor $S'$ corresponding to the merging of cells $C_i$ and $C_j$. This condition trims the number of states generated and so reduces the number of states to be searched. $S'$ contains one cell less than $S$ because cells $C_i$ and $C_j$ in $S$ are merged to form a new cell in state $S'$. For all cells other than $C_i \cup C_j$ in $S'$, the intercellular traffic remains the same. For the merged cell $C_i \cup C_j$ in $S'$, the traffic between all the other cells in $S'$ is the sum of $C_i$'s traffic and $C_j$'s traffic with the other cells in $S$. 3-9 shown below represents this statement.

$$F(C_i \cup C_j) = F(C_i, C_i \cup C_j) = F(C_i, C_j) + F(C_i, C_j) \quad \text{(3-9)}$$

In addition to the children produced in the above procedure, $S$ also has a child corresponding to the decision not to merge $C_i$ with any other cell. It can be understood from the above formulation that the maximum depth of the search tree is $m$, and that every possible partition of machines is a node in the defined search space.
3.1.4 State ordering and filtering mechanism

Knowledge about the space states can be used to introduce the concept of direction in the space search of the problem domain for a good solution. Heuristic functions can be designed for the purpose of guiding the search in the correct direction so that a good solution can be achieved in a reasonable amount of time. The state transition function discussed above can produce all possible child states for the current state. If no knowledge is included in this search, then the search ends up being a blind search in the state space of the problem. The disadvantages of blind search techniques have been discussed widely. The search for a good solution may get stuck in a bad patch and return very poor solution quality in the available time and memory resources. Best search technique introduces direction to the search in the state space, but all the generated states have to be remembered in memory and the search becomes memory intensive. If knowledge in the form of weight is included in each state, then states can be compared for quality and only those states that pass fitness tests based on weight are retained for further expansion. Even those states which clear the fitness tests are ordered based on the value of weight so that better states are expanded before states with an inferior value of weight. This technique ensures that the quality of the solution returned by the space search algorithm is not only good but also the time and memory resources spent on the problem can be controlled.
3.1.4.1 Heuristic for state evaluation

The size of the search space depends on heuristic evaluation function for visited states. The heuristic used to sort child states is based on the computation of normalized traffic reduction for a state. The technique is called relaxation of merging. Relaxation of merging relaxes the merging rules and allows for maximum merging of cells. The remaining intercellular traffic is taken as the lower bound for the traffic value. Let \( S \) be any arbitrary state and let \( C_1 \) to \( C_p \) be the cells in \( S \). 3-10 defines the total intercellular traffic for the state \( S \).

\[
\sum_{i=1}^{p-1} \sum_{j=i+1}^{p} F(C_i, C_j) \tag{3-10}
\]

For the traffic matrix shown in Figure 3-2, this total intercellular traffic value is the sum of the upper triangular (or lower triangular matrix). This traffic value is 700, the sum of traffic values of 100, 100, 100, 100, 100 and 200 between the five distinct cells of the state \( S \). A factor called maximum possible traffic reduction \( R(C_i) \) is computed for each cell in the current state. This reduction represents an upper bound on the maximum reduction in traffic that can be achieved by merging the current cell \( C_i \) with the other cells in the state. A lower bound on the intercellular traffic value for each state is found from the total intercellular traffic and the maximum reduction as shown in 3-11.

\[
LB(S) = \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} F(C_i, C_j) - \frac{1}{2} \sum_{i=1}^{p} R(C_i) \tag{3-11}
\]
This factor is computed for all the children of a particular state. Child states that have a higher value of the lower bound value than the current lower bound value can be eliminated from the space search as the risk of losing the optimal solution is minimized. The value $R(C_i)$ is divided by half because the traffic between cells $C_i$ and $C_j$ is included twice in the computation, once as $F(C_i, C_j)$ and then as $F(C_j, C_i)$. If an upper bound can be computed, then this range represents a tight range of traffic values, which can be used to filter child states. ICTMM [9] was used as an upper bound to decide the tolerance range for the traffic value.

### 3.1.4.2 Computation of total traffic reduction for a state $S$

Let $S$ be any state and let $C_1 - C_p$ be the cells in the state. Then, the reduction for the state $S$ is defined as $\sum_{i=1}^{p} R(C_i)$. The cells in state $S$ are considered in decreasing order of the ratio of cell traffic to cell cardinality. The cell cardinality is the count of the number of machines in the cell. 3-12, 3-13, 3-14 and 3-15 represent this iterative procedure of computation of total traffic reduction. Let $r$ be the reduction in traffic that can be achieved for the first cell $C_i$ under consideration. Let $c$ be the cardinality of the cell $C_i$.

```plaintext
//Initialize computation
r = 0; c = |C_i|  \hspace{1cm} (3-12)
```

Loop
If there is no merge able cell $C_j$ that satisfies the conditions for merging of two cells, then return $r$. Else let $C_j$ be the cell that maximizes $F(C_i, C_j) / |C_j|$. 

If $c + C_j \leq N$, then

$$r = r + F(C_i, C_j) \text{ // accumulated traffic.}$$  \hspace{1cm} (3-13)

$$c = c + C_j \text{ // Cardinality of new cell.}$$  \hspace{1cm} (3-14)

Eliminate $C_j$ from future consideration.

Else

$$r = r + F(C_i, C_j) \times (N - c) / |C_j|$$  \hspace{1cm} (3-15)

Repeat until there are no cells that need computation.

3.1.5 Space search with best search algorithm

The best search technique selects the best search for expansion from possible child states. Upon the imposition of ordering constraints, directional search of the state space can be achieved. If the heuristic evaluation function is good, then a good solution can be achieved in a reasonable amount of time and with limited memory resources. An ordering constraint ordered the child state of the current state based on the value of the lower bound of the intercellular traffic. Child states produced by the state transformation function are ordered in increasing order of their value of the lower bound function for intercellular traffic.
3.1.6 Search termination

A goal state is a state in which all the cells are marked “Closed”, no further merging and hence no further reduction in traffic can take place. This state is marked for evaluation as a possible solution.

3.2 Cell formation with alternative process plans

This section discusses the space search algorithm for cell formation with consideration of alternative process plans for parts. This algorithm considers the presence of alternative process plans for manufacturing parts and the emphasis is on simultaneous selection of process plans for parts and cell formation.

3.2.1 Problem formulation

Nomenclature:

\[ M \] - Set of machines in the manufacturing system.
\[ m \] - Total number of machines in the manufacturing system.
\[ P \] - Set of parts to be manufactured in the manufacturing system.
\[ p \] - Total number of parts to be manufactured in the manufacturing system.
\[ q_i \] - Number of units of part type \( P_i \in P \).
\[ c_i \] - Cost of one unit of part type \( P_i \in P \).
\[ r_{ik} \] - The \( k^{th} \) routing for part type \( P_i \in P \).
\[ l_i \] - Total number of possible routings for part type \( P_i \in P \).
\[ v \] - Total number of machine operations on a part.
\[ N \] - Maximum number of machines in a cell.
\[ nc \] - Number of machine cells in \( S \).
\[ nf \] - Number of part families in \( S \).
\[ S \] - A configuration of machine cells and part families in the problem domain.
\[ C \] - Collection of machine cells for the configuration \( S \).
\[ PF \] - Collection of part families for the configuration \( S \).
\( Ci \in C, i = 1 \to nc \) - Machine cell in \( C \), a subset of \( M \).

\( PF_i \in PF, i = 1 \to nf \) - Part family in \( PF \).

\( PE \) - Function that determines the number of exceptional machine operations in a part.

\( PFE \) - Function that determines the number of exceptional machine operations in a part family.

\( CE \) - Function that determines the number of exceptional machine operations in a configuration.

\( U \) - Set of all possible cell configurations for the cell formation problem.

A manufacturing system with a set \( M = \{M_1, M_2, M_3, \ldots, M_m\} \) of \( m \) machines and a set \( P = \{P_1, P_2, P_3, \ldots, P_p\} \) of \( p \) parts is considered. Let \( q_i \) represent the desired part quantity for part \( P_i, i = 1 \to p \). Let \( c_i \) represent the cost associated with the production of one unit of part type \( P_i \). Each part can be produced by alternative routings or process plans. A routing for a part is defined as a sequence of \( v \) operations on a subset of \( M \) through, which a part has to pass to be completely manufactured. The \( k^{th} \) routing associated with part \( P_i \) is represented as \( r_{ik} \). Each part can have different number of alternative routings, say \( l_i \). A routing \( r_{ik} \) for a part \( P_i \) can be defined by 3-16 as follows.

\[
r_{ik} = \{m_{ik1}, m_{ik2}, \ldots, m_{ikv}\}, k = 1 \to l_i.
\] (3-16)

where \( v \) is the number of operations for part \( P_i \) on the \( k^{th} \) routing \( r_{ik} \).

The value of \( v \) is the number of machine operations in an alternative route and varies from process plan to process plan. Processing times and setup times are not considered for this algorithm. Because of practical considerations, the number of machines in a cell is usually restricted to a certain value, say \( N \). A
configuration $S$ in this problem domain is defined by a collection of machine cells $C$ and the corresponding collection of part families $PF$. 3-17 shows the state space representation for each space state.

$$S = \{C, PF\} \quad \text{(3-17)}$$

where

$C$ is a collection of machine cells.

$PF$ is a collection of part families.

3-18 and 3-19 shows the collection of machine cells for the space state $S$.

$$C = \{C_1, C_2, \ldots, C_m\}, \quad \text{(3-18)}$$

Where $C_i \cap C_j = \emptyset$,

for $i, j \in \{1, \ldots, nc\}$, $|C_i| \leq N, |C_j| \leq N, i \neq j$.

$$PF = \{PF_1, PF_2, \ldots, PF_m\} \quad \text{(3-19)}$$

A machine cell $C_i \in C$ is a collection of $o$ machines from $M$ and is defined by 3-20.

$$C_i = \{M_{i1}, M_{i2}, \ldots, M_{io}\}, \quad \text{(3-20)}$$

Where each $M_{ij} \in M, i = 1 \text{ to } nc, j = 1 \text{ to } o, o \leq N$.

One machine in $C_i \in C$ is the key machine based on which part families are formed. A part family $PF_i$ corresponding to machine cell $C_i \in C$ is a collection of $n$ parts from $P$ and is defined by 3-21.

$$PF_i = \{P_{i1}, P_{i2}, \ldots, P_{in}\}, \quad \text{(3-21)}$$
Where each \( P_s \subset \{ r_{sk} \}, 1 \leq s \leq l_i, i = 1 \) to \( np, s = 1 \) to \( n \), subset of part activity on part \( P_s \) such that \( m_{ij} \in C_i, j = 1 \) to \( l_i \)

Each part in \( P \) can be manufactured through alternative process plans. Decision has to be made as to which alternative process plans for a part are to be selected to form a part family in a particular configuration. This decision is made during space search. Each part family has a count of its exceptional machine operations. This is the sum of all the exceptional machine operations on all the parts in the part family. An exceptional machine operation on a part \( P_s \in P, PF_i \) is a machine operation on a machine \( m_{ij} \in M, \notin C_i \). Assume the function \( PE(P_s) \) determines the number of exceptions for part \( P_s \in P, PF_i \) and the function \( PFE(PF_i) \) determines the number of exceptions of the family \( PF_i \in PF \). Then, these functions are correlated as shown by 3-22.

\[
PFE(PF_i) = \sum_{s=1}^{n} PE(P_s) \quad (3-22)
\]

Let \( CE(S) \) be the function that determines the number of exceptions for the configuration \( S \). The number of exceptions for state \( S \) is the sum of the number of each part family in the state. \( CE(S) \) is defined as shown by 3-23.

\[
CE(S) = \sum_{i=1}^{nf} PFE(PF_i) \quad (3-23)
\]
The combination of all the possible cell configurations for the cell formation problem with alternative process plans is defined as $U$. The cell formation problem with alternative process plans is the problem of finding the optimum cell configuration $S^* \in U$ such that the condition specified by 3-24 is satisfied.

$$CE(S^*) = \min_{S \in U} CE(S) \tag{3-24}$$

3.2.2 Preparation of machines for the space search algorithm

Before the beginning of the cell formation process, usage frequencies are calculated for all the machines in the manufacturing system. Machines are then sorted in increasing order of their usage frequency before the cell formation process is started.

3.2.2.1 Calculation of machine usage frequency

The usage frequency attempts to determine the amount of usage of each machine, in the presence of alternative process plans for parts. Every part in the manufacturing system contributes to the usage frequency calculation on a machine. Table 3-1 shows a manufacturing system with four parts Part1, Part2, Part3 and Part4 and five machines M1, M2, M3, M4 and M5. It also shows the routing information of each part and the usage frequency and SICGE code of each machine. This example is used purely for illustration purposes and does not represent real data.
Table 3-1: Part machine matrix with machine frequency values

<table>
<thead>
<tr>
<th>Part</th>
<th>Machine</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>S</td>
<td>I</td>
<td>I</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1.67)</td>
<td>(2.5)</td>
<td>(1.83)</td>
<td>(2.33)</td>
<td>(2.84)</td>
</tr>
<tr>
<td>Part 1</td>
<td>A1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Part 2</td>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Part 3</td>
<td>A1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Part 4</td>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

A ‘1’ in the routing information of an alternative process plan of a part indicates that the part has an activity on the machine corresponding to the column of the ‘1’ entry. A ‘0’ entry indicates that the part does not have an activity on that machine in the current alternative process plan. Frequency calculation is explained on machine M1. Every part in the manufacturing system that uses machine M1 contributes to the usage frequency of M1. Part 1 uses machine M1 in alternative process plans A1 and A2. So, the contribution of Part 1 to the usage frequency of M1 is $2/3 = 0.67$ (2 out of 3 alternative process plans of Part 1 use M1). Similarly, the contribution of Part 2 is $2/3 = 0.67$, Part 3’s contribution is zero (no alternative process plan of Part 3 uses M1). Part 4’s contribution is $1/3 = 0.33$. So, the total usage frequency of M1 is $(0.67+0.67+0.33 = 1.67)$. 
Similarly, the usage frequency of the remaining machines is also calculated in the initial part machine matrix.

3.2.2.2 Sorting the machines in the manufacturing system

A sorting scheme is introduced to order the machines in the manufacturing system. This sorting principle is based on the sorting principle for machines developed by Burbidge in Group Analysis of Production Flow Analysis [2]. Machines in a manufacturing system are sorted according to the SICGE classification. First S type machines are listed followed by I, C, G and E type machines in that order. Further, under each category, the machines are sorted in the increasing order of magnitude of the machine usage frequency. Table 3-2 shows the sorted machines from Table 3-1 have been sorted.

<table>
<thead>
<tr>
<th>Machine Part</th>
<th>M1 S (1.67)</th>
<th>M3 I (1.83)</th>
<th>M2 I (2.5)</th>
<th>M3 C (2.33)</th>
<th>M5 C (2.84)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Part 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Part 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Part 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
3.2.3 State space representation

A part family, a machine cell, a parent state, a machine part incidence matrix of the unassigned parts and unassigned machines and a count of the number of exceptions represent each state in the search space. The machine cell is based on a nucleus machine, which forms the core of the space-state. This machine is called as the “Key Machine”. The part family formed corresponding to this key machine is called the “Root Family” of the state. The root family includes all or some of the parts that have activity on this key machine. The machine cell consists of additional machines that may be needed to completely manufacture the parts in the part family. The part family in turn contains additional parts that can be completely manufactured with the machine cell of this state without requirement of additional machines. This set of additional parts together with the parts with activity on the key machine form the part family of this state. Each state also holds a count of its exceptional machine operations. Thus, each space state can be represented by 3-25.

\[ S_i = (KM_i, PF_i, C_i, S_{i-1}, MA_i, EX_i) \]  \hspace{1cm} (3-25)

where

- \( KM_i \) is the key machine for State \( S_i \).
- \( PF_i \) is the part family for State \( S_i \).
- \( C_i \) is the machine cell for State \( S_i \).
- \( S_{i-1} \) is the parent state of State \( S_i \).
\( MA_i \) is the unassigned part family for State \( S_i \).

\( EX_i \) is the number of exceptional operations for state \( S_i \).

The unassigned part machine matrix consists of a list of unassigned parts and unassigned machines defined by (3-26).

\[
MA_i = \{UP_i, UM_i\} \tag{3-26}
\]

where

\( UP_i \) is the collection of unassigned parts for this state.

\( UM_i \) is the collection of unassigned machines for this state.

The initial state \( S_0 \) from which the search is started is defined by (3-27).

\[
S_0 = \{null, null, null, null, MA_0, 0\} \tag{3-27}
\]

The initial state does not have a key machine, a part family, a machine cell and a parent state. Consequently, the count of exceptions is zero. The machine part incidence matrix of unassigned parts and unassigned machines is the same as the one shown in Table 3-2 as no part families and machine cells have been formed yet. Machines from the matrix are considered in the sorted order of SICGE and usage frequency for cell formation. From Table 3-2, M1 is an ‘S’ type machine and has the least frequency of all ‘S’ type machines (In this case, there is only one ‘S’ type machine). Thus, M1 is the key machine for the children of the state \( S_0 \). Part 1 A1, Part 1 A2, Part 2 A1, Part 2 A2 and Part 4 A1 have an activity on M1. The state transformation function to generate new states is explained in the next section. For now, assume that the candidate parts selected for family
formation are Part 1 A2 and Part 2 A1. Part 1 A2 has M1, M3 and M2 in its process plan. Part 2 A1 has M1 and M5 in its process plan. So, the candidate machines to completely manufacture the part family of Part 1 A2 and Part 2 A1 are M1, M3, M2 and M5. Also, assume no additional parts are selected for family formation. The remaining machine to be assigned is M4 and the remaining parts to be assigned are Part 3 and Part 4. This child state \( S_{\text{child}} \) is defined by 3-28).

\[
S_{\text{child}} = (KM_{\text{child}}, PF_{\text{child}}, C_{\text{child}}, S_0, MA_{\text{child}}, EX_{\text{child}})
\]  

where

\( KM_{\text{child}} = M1 \)

\( PF_{\text{child}} = \{\text{Part 1 A2, Part 2 A1}\} \)

\( C_{\text{child}} = \{M1, M3, M2, M5\} \)

\( S_0 \) is the parent state.

\( MA_{\text{child}} = \{UP_{\text{child}}, UM_{\text{child}}\} \)

\( UP_{\text{child}} = \{P3, P4\}, UM_{\text{child}} = \{M4\} \)

\( EX_{\text{child}} = 0 \)

Table 3-3 shows the part machine matrix of unassigned parts and unassigned machines for state \( S_{\text{child}} \). The number of exceptions of this state is zero as no exceptions are carried over from the parent state and no exceptions are formed in the child state.
Table 3-3 Unassigned part machine matrix

<table>
<thead>
<tr>
<th>Part</th>
<th>Machine</th>
<th>M4 C (2.33)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>A1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>A1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>1</td>
</tr>
</tbody>
</table>

The part family formed can be completely manufactured with the machine cell in this state. Additional information like number of units of each machine, count of exceptions that lead to intercellular traffic can also be included as part of the state information.

3.2.4 State transition mechanism

The state transition mechanism is the mechanism by which new space-states are generated from the current state. The newly generated space-states are called as child states. The child states hold information about its parents, and also some information of its own. Assume the current state in the search space is \( S_i \). The state transformation function works on \( S_i \) to produce child states. Let one child state \( S_{i+1} \) be defined by 3-29.

\[
S_{i+1} = \{KM_i + 1, PF_i + 1, Ci + 1, Si, MA_i + 1, EX_i + 1\}  
\]  

(3-29)
Part families are formed incrementally from the unassigned parts in the parent state having activity on key machine. Adding the key machine and any additional machines required to completely manufacture the part family of this state forms the machine cell, provided the machines are available in the unassigned machines of the part-machine matrix in the parent state. The unassigned parts are the parts that have not yet been assigned to a part family. It is the difference between the unassigned parts of the parent state and the parts in the part family of the child state. The unassigned machines are the machines that have not yet been assigned to a machine cell. It is the difference between the unassigned machines of the parent state and the machines in the machine cell of the child state. The number of exceptional machine operations in the child state \(S_{i+1}\) is the sum of the number of exceptional machine operations in the
parent state $S_i$ and the number of exceptional machine operations in the part family $PF_{i+1}$ of this state, as determined by the function $PFE$. For the cell formation algorithm with alternative process plans, a transformation function based on incremental part family sizes is designed. The function creates incremental size part families for selected candidate parts for the key machine in the current state. Each part family becomes the root family for a child state. The key machine for the states at one level of the search tree remains the same. For the example shown in Table 3-1, the first key machine is M1. Part 1 A1, Part 1 A2, Part 2 A1, Part 2A2 and Part 4A1 have activity on M1. These form the candidate parts list. Figure 3-8 shows the possible child states.

Figure 3-8: Part families generated from state transition.
Part families are formed incrementally to consider all possible space states. Also, a constraint is introduced that at this stage of family formation. Only one alternative process plan for a part is selected from selected alternatives in each family formed. This increases the complexity of the search. This means that each of the part families has only one alternative for each selected part. Each of the above generated states has the same key machine i.e. M1. Each of the root families generated by the state transformation mechanism becomes the root family of a child state. There are 17 child states for initial state corresponding to the 17 root families formed as a result of state transformation. For each state, after the identification of the root family, additional machines are included in the machine cell corresponding to this state from the unassigned machines list of the parent state to completely manufacture the root family depending upon availability. After the inclusion of additional machines in the machine cell, additional parts are included in the part family of this state from the unassigned parts list. Only those parts that can be completely manufactured with the machine cell of this state are included as additional parts. At this stage, all alternatives for the selected parts that can be completely manufactured with the machine cell of this state are included. This ensures that all possible advantages of alternative process plans are used when forming manufacturing cells. For the space state \( S_{child} \) defined by 3-28, only one child state is possible, as \( S_{child} \) has only one unassigned machine \( M_4 \). So, the child state for \( S_{child} \) is defined by 3-30.
\[
S_{\text{child}} - \text{child} = \{K_{\text{child}} - \text{child}, PF_{\text{child}} - \text{child}, C_{\text{child}} - \text{child}, S_{\text{child}}, MA_{\text{child}} - \text{child}, EX_{\text{child}} - \text{child}\}
\]

(3-30)

where

\[
K_{\text{child}} - \text{child} = M4.
\]

\[
PF_{\text{child}} - \text{child} = \{Part3A1, Part4A3\}
\]

\[
C_{\text{child}} - \text{child} = \{M4\}
\]

\[
S_{\text{child}} \text{ is the parent state.}
\]

\[
MA_{\text{child}} - \text{child} = \{UP_{\text{child}} - \text{child}, UM_{\text{child}} - \text{child}\}
\]

\[
UP_{\text{child}} - \text{child} = \{\}, UM_{\text{child}} - \text{child} = \{\}
\]

\[
EX_{\text{child}} - \text{child} = 2
\]

It is assumed above that the part family of Part 3 A1 and Part 4 A3 is selected for state progression. All parts have been assigned to part families and all machines are assigned to machine cells. Total number of exceptions is 2 as both Part 3 A1 and Part 4 A3 have an activity on M4, but M5 has already been assigned.

3.2.5 State ordering and filtering mechanism

In order to achieve a good solution as quickly as possible, it may be necessary to expand only certain child states generated from the state transformation. Also, if an ordering scheme is introduced into the space search so that child states are ordered based on some heuristic, there is a better chance of getting a good solution in reduced time. A heuristic based on the count of
exceptional machine activities in a state is defined. The result of this heuristic function is used to sort the child states generated from the state transformation in increasing order of the count of exceptional machine operations. Thus, best search based on minimum number of exceptional machine operations can be performed.

3.2.5.1 Heuristic for state ordering

Each state has a count of its exceptional elements. The total number of exceptional elements for the state is the sum of the number of exceptional elements in its root family and the number of exceptional elements in the parent state. After the computation of the number of exceptional elements for all the child states of the current state, the child states are sorted in increasing order of the count of exceptional elements. Thus, a child state with lesser number of exceptional machine operations will be expanded before a child state with more number of exceptional machine operations.

3.2.5.2 Heuristic for state elimination

If all the child states are considered for expansion in space search, then the search space becomes huge for even problems of small instances. So, an elimination strategy has to be introduced to eliminate some child states after they have been generated. An elimination strategy is introduced to eliminate states with smaller root families on a percentage basis. The value $p$ is a percentage
such that all states where number of parts in their root families satisfies the inequality

\[ |PF| < el = p \times n / 100 \]  (3-31)

where

- \( p \) is a user defined elimination value on a percentage basis.
- \( n \) is the number of initially selected candidate part models on the key machine for the current state.

All states that satisfy this inequality are eliminated from further analysis (deleted from open list). For the initial state shown in Figure 3-8, the candidate part routings selected are Part 1 A1, Part 1 A2, Part 2 A1, Part 2 A2 and Part 4, A1. For 70% elimination, the integer value of \( el \) is 0.7*3 = 2.1 = 2. This means that all families formed as a result of state transformation, which have a size less than 2 will not be considered for expansion. Thus, all the child states of the current state whose root family sizes are less than \( el \) are eliminated from the search space and are not considered for expansion. Since the attempt is to completely manufacture part families with all the machines that they need, bigger sized families have more probability of returning better solutions. Also, the eliminated families may be subsets of bigger families. This also reduces redundant search.
3.2.6 Space search with best search technique

The best search algorithm developed to solve the problem of cell formation in the presence of alternative process plans is explained in this section. It includes all the aspects needed to complete space search, i.e., state initialization, state transformation and state ordering and elimination mechanism. The algorithm is explained stepwise.

1. Machines in the manufacturing system are prepared for space search. SICGE classification and machine frequency is found.

2. Machines are sorted according to SICGE classification in that order. Within each type, machines are again sorted in increasing order of the machine usage frequency.

3. Space search is started. A machine with the least usage frequency from the unassigned machines list is selected as the key machine.

4. All the part models and their part activities (routings), which have a machine activity on the key machine, are selected. These part activities form the candidate parts for the root family of the child states. If the number of part activities selected is zero, then if the number of remaining machines to be assigned is not zero, then this machine is removed from the candidate machines list and search is continued with the next available machine.
5. The state transformation function is applied on the initial/current state and new child states are generated based on incremental part families.

6. Find all the machines needed to completely manufacture a part family from the unassigned machines list. These machines with the key machine form the machine cell of the state corresponding to the current part family.

7. Remove the assigned machines from the unassigned machines list.

8. Apply the cell size rule to the generated child states. Remove all states that violate the cell size rule.

9. Find additional part models and their part activities that can be completely manufactured with the machine cell of the current state. Include all part activities of a part model that can be completely manufactured with the machine cell of the state.

10. Include the additional part models and their activities on the root family of the current state.

11. Remove the assigned part models along with all its assigned and unassigned part activities from the unassigned parts list.

12. Include all the machine cells and part families formed in the parent states of the current state in the current state.
13. Eliminate states that violate the family size rule based on percent elimination.

14. Select the best state for expansion.

15. Select the next available machine with the least frequency from the unassigned machines list and continue from step 4.

16. If there are no more machines available to be assigned to machine cells, then, if there are some unassigned part models, these are assigned to a manufacturing cell in the current state in which a maximum number of machine operations for the part models can be carried out.

17. If all parts have been assigned to part families, then this state is a possible goal state. The number of exceptional elements for this state is found and is recorded as a goal state. In further expansion, if a better goal is found then the newly found state is made as the goal state.

18. If there are no more states to be expanded, then search is terminated and the current goal state is returned as the result of the space search algorithm.

19. The goal state has a list of all the machine cells and part families formed as a result of space search. It also has all details of the exceptional elements in the state for all the manufacturing cells formed.
3.2.7 Search termination

There are two main cases in which a search for a goal state can terminate in the course of space search. The first case is when space search has explored all possible states and the best state is returned as the goal state. The second case is when the search for a goal state is terminated because the space search has reached a dead end and is not able to proceed. In this algorithm, dead end does not happen because when all machines have been assigned to machine cells, the remaining parts are assigned to cells in such a way that they introduce minimum number of exceptions. The second case of termination can also be classified into two more categories depending on the availability of memory. In the course of space search, the computer may run out of memory. This case can be divided into two categories depending on whether a goal state was found before the computer ran out of memory.
4 System architecture and implementation

Software prototypes were constructed as part of this research to build process plan models and perform space search for cell formation algorithms. This section describes the prototypes constructed, the process plan model built, process plan data generation with XML and interface between a space search problem and the space searcher tool.

4.1 Process plan representation and data generation

This section describes the process plan representation model for representing process plans and cellular manufacturing model for the cell formation problem. Construction of Java classes and attributes for model entities and attributes is described with appropriate examples and code snippets. Model data generation with a tool called Factory model generator is also discussed.

4.1.1 Process plan representation model

Sormaz [23] proposed a manufacturing model for representation of process plans based on the different activities involved in manufacturing. A generic manufacturing model facilitates the development of better algorithms for manufacturing problems like sequencing, scheduling etc. by reducing the over-all algorithm development time. The model accommodates a variety of data that
may be needed in manufacturing algorithms. Figure 4-1 shows the object-oriented representation of the process plan representation model.

![Figure 4-1: Process plan representation model for manufacturing](image)

A manufacturing activity represents the core of the model. Any activity that contributes to the manufacturing of a part is called as a manufacturing activity. A manufacturing activity has attributes like manufacturing cost, manufacturing time, member process etc. A manufacturing activity can be a part activity, machine
activity or a tool direction activity. A part activity describes the process plan for a part. There is an association relationship between part activity and part. Each part activity is associated with a part. A process plan for a part is a collection of the machines through which a part has to pass through to be completely manufactured. The part in turn can have multiple alternative process plans. The machining process of a part on each machine in its process plan is represented by a machine activity. Each machine activity is associated with a machine object. There is an aggregation relationship between part activity and machine activity. Each part activity is a collection of machine activities. The part object has attributes like a collection of its alternative process plans, part material, features list, process list etc. The machine object has attributes like machine name, number of units, usage frequency, SICGE code etc. Each machine activity is a collection of tool direction activities. A tool direction activity holds directional information about a machining process. The member process attribute of manufacturing activity is used to store the aggregations of a manufacturing activity. Thus, the member process attribute of a part activity holds a collection of machine activity. The member process attribute of a machine activity holds a collection of tool direction activities. The representation model for the cellular manufacturing shown in Figure 4-2 is built on the basis of the process plan representation model.
The core of the cellular manufacturing model is the manufacturing system. A manufacturing system is a place where parts are manufactured on machines. The main task of cell formation is the partition of the parts in the system into part families and machines into machine cells to reduce intercellular traffic. The objective is to find a partition for machines into machine cells and parts into part families in such a way as to minimize intercellular traffic.

4.1.2 Definition of Java classes for models

The entities and attributes of the process plan representation model and cellular manufacturing model have to be captured in programmatic code for use in software applications. The object oriented nature of the Java programming language makes it extremely compatible and appropriate to create an object-
oriented data structure. The main properties that make Java suitable for object-oriented programming are encapsulation, polymorphism and inheritance [29]. The entities in the process plan representation model include manufacturing activity, part activity, machine activity, tool direction activity, part model and machine. These entities describe their state with the help of attributes. Each of the entities in the process plan representation model is converted into Java classes. The attributes of each entity are converted into Java variables and objects. These attributes can be accessed only through accessor and modifier methods of the Java classes. This is called as encapsulation. Variables common to all manufacturing activity like manufacturing cost, manufacturing time, etc are defined in manufacturing activity class. Since manufacturing activity can be a part activity, a machine activity, or a tool direction activity, these classes are defined as child classes of the manufacturing activity class. By inheritance, these child classes have access to their parent’s attributes. The different forms of the manufacturing activity are the result of polymorphism. The class variables are declared as private or protected to ensure encapsulation. Any modifications to these variables have to be performed through the accessor or modifier methods. Inheritance is incorporated by making a Java class extend another class by including the keyword extends in the class definition. Figure 4-3 shows the definition of the Manufacturing Activity java class.
public class ManufacturingActivity extends ImpObject {
    private double manufacturingCost;
    private double manufacturingTime;
    protected LinkedList memberProcess;
    // Constructor Definition
    public ManufacturingActivity () {
        // Initialize attributes
    }
    // Accessor methods
    public double getManufacturingCost () {
        return manufacturingCost;
    }
    // Modifier methods
    public void setManufacturingCost (double cost) {
        manufacturingCost = cost;
    }
}

Figure 4-3 Java class for Manufacturing Activity

Similar class files are built for all the entities in the process plan representation model and cellular manufacturing model. These include classes for MfgPartModel, PartActivity, MachineActivity, ToolDirectionActivity and Machine in process plan representation model and ManufacturingSystem, PartFamily and MachineCell in cellular manufacturing representation model.

4.1.3 Factory model generator

Factory models based on the process plan model can be used as input to the cellular manufacturing algorithm to form manufacturing cells. To automate the
The process of generating factory models based on the process plan representation model, a graphical user interface called as Factory Model Generator was designed and constructed as part of this thesis. FMG is a technique for rapid collection of factory models based on GUI based data collection. Figure 4-4 shows the factory model generator tool.

The Java Swing API [26] provides excellent functionality for the construction of graphical interfaces. The data structures in the form of Java
classes defined in Section 4.1.2 are used to construct instances of manufacturing activities, parts and machines. Table 4-1 provides a brief explanation of the functionalities available in the Factory model generator tool.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="icon1.png" alt="Icon" /></td>
<td>New Model</td>
<td>Creates a new factory model with zero parts and machines.</td>
</tr>
<tr>
<td><img src="icon2.png" alt="Icon" /></td>
<td>Open Model</td>
<td>Opens a previously saved factory model.</td>
</tr>
<tr>
<td><img src="icon3.png" alt="Icon" /></td>
<td>Save Model</td>
<td>Save the currently open factory model.</td>
</tr>
<tr>
<td><img src="icon4.png" alt="Icon" /></td>
<td>Add part model.</td>
<td>Adds a new part model to the factory model.</td>
</tr>
<tr>
<td><img src="icon5.png" alt="Icon" /></td>
<td>Add alternative</td>
<td>Adds an alternative process plan to the previously added part.</td>
</tr>
<tr>
<td><img src="icon6.png" alt="Icon" /></td>
<td>Add machine</td>
<td>Add a new machine to the factory model.</td>
</tr>
<tr>
<td><img src="icon7.png" alt="Icon" /></td>
<td>Delete part model.</td>
<td>Delete a selected part model.</td>
</tr>
<tr>
<td><img src="icon8.png" alt="Icon" /></td>
<td>Delete part activity.</td>
<td>Delete the selected alternative process plan from a part.</td>
</tr>
<tr>
<td><img src="icon9.png" alt="Icon" /></td>
<td>Delete Machine.</td>
<td>Delete selected machine from factory model.</td>
</tr>
<tr>
<td><img src="icon10.png" alt="Icon" /></td>
<td>Clear.</td>
<td>Clear factory model of all parts and machines.</td>
</tr>
<tr>
<td><img src="icon11.png" alt="Icon" /></td>
<td>Configure search.</td>
<td>Configure space search with necessary information.</td>
</tr>
<tr>
<td><img src="icon12.png" alt="Icon" /></td>
<td>Start search.</td>
<td>Start the process of space search.</td>
</tr>
</tbody>
</table>

The functionalities included in FMG are creation of a new factory model, opening an existing model and saving an open model, adding/deleting part models and machines to an existing factory model, generating the process plan
information for all part models etc. The tick marks indicate a machine activity in the process plan of a part model corresponding to the row of the tick mark. This machine activity is on a machine that corresponds to the column of the tick mark. All the tick marks in a row corresponding to a part activity on a part model correspond to the process plan of the part. Figure 4-5 shows the SPL of the Factory Model Generator for the example shown in Figure 4-4.

![Figure 4-5: Special plant list on Factory model generator.](image)

Option for sorting machines according to SICGE coding and usage frequency is also available. SPL is a list of all the machines in the factory model sorted according to SICGE coding and usage frequency. The data in Factory
model generator is converted into XML data for the purpose of saving and reuse. Figure 4-6 shows the XML data generation panel on FMG.

Figure 4-6: XML data generation panel.
4.2 XML data generation and parsing

This section discusses the data generation mechanism for the process plan representation model. It describes the XML schema based on which process plan data is generated, generation of XML data from Java and reconstruction of class instances from XML code.

4.2.1 XML schema for PPRM

A schema is a set of rules that have to be conformed to in an XML document. The root element is <MfgSystem> that holds all information about a factory model. This element holds two collection elements <partsList> and <machinesList>. The <partsList> element is a collection of part models represented by the <MfgPartModel> element. The <machinesList> is a collection of the available machines represented by <Machine> element. Each <MfgPartModel> element has a list element called <partActivityList>, which is a collection of alternative process plans, for the enclosing part element. These alternative process plans are represented as <PartActivity> elements on the part model. Each <PartActivity> element has a <machineActivityList> element. The <machineActivityList> element is a collection of all the <MachineActivity> elements in the enclosing process plan. Each <MachineActivity> element encloses a <Machine> element that holds information about the machine on which the machining operation is done. Each <MachineActivity> element holds a
<toolDirectionActivityList> element. This element holds the spatial information about the tools used in this machining activity. The hierarchical relationship between parts, machines, part activity, machine activity and tool activity is incorporated in the XML document. Figure 4-7 shows the XML Schema definition for the process plan representation model taken from XML Spy Suite.

4.2.2 XML data generation mechanism

An object-oriented data generation framework was constructed for the purpose of generating XML data for factory models based on the process plan
representation model. The different entities whose XML data needs to be generated for the factory model include \textit{MfgPartModel}, \textit{Machine}, \textit{PartActivity}, \textit{MachineActivity} and \textit{ToolDirectionActivity}. All the Java classes built for the process plan representation model should be capable of generating self-describing XML data. Figure 4-8 shows the \textit{Machine} java class with its XML data

```java
public class Machine extends ImpObject {
    private String machineName;
    private String machineCode;
    private int totalUnits;
    private double frequency;

    public void writeXML (StringBuffer xmlBuffer, String inTab) {
        String constTab = "    ";
        xmlBuffer.append(inTab);
        xmlBuffer.append("<Machine machineName="+
                        this.getMachine () +
                        "\" machineCode
                        =\""+this.getMachineCode () +\"\"
                        number=\""+
                        this.getNumberofUnits () +\"\"
                        frequency =\"" +
                        this.getFrequency ()+\"">\n"");
        xmlBuffer.append (inTab);
        xmlBuffer.append ("</Machine>\n");
    }
}
```

Figure 4-8 Machine Java class showing XML data generation mechanism
The *Machine* class holds information about the machine like name, number of units, usage frequency, SICGE code etc. It is this state data for each machine that needs to be saved into XML. Ignoring the specifics of the programming involved, the generated XML element data for a machine whose name is M1, whose units is 1, usage frequency is 0.33 and SICGE code is ‘S’ is shown in Figure 4-9.

```xml
<Machine machineName = "M1" machineCode = "S" number = "1"
 frequency = "0.33">
</Machine>
```

Figure 4-9 XML Data for a Machine

To establish uniformity, a method called `writeXML` with the following syntax is defined.

```java
writeXML (StringBuffer xmlTag, String indent)
```

This method is called on the Java class whenever data about the state of an instance of the class has to be generated in the form of XML. The argument `xmlTag` holds all the factory model information generated in XML form until the `Machine` class method was called. The generated XML data from the `Machine`
class is then included in the xmlTag variable. The indent argument is to generate properly indented XML code to improve readability

4.2.3 XML data parsing

XML data generated with Factory model generator holds all the information necessary to reconstruct the Java class instance, including data about all the parts in the factory model, the machines in the factory model, the routing information for all the parts etc. XML parsers can be used to reconstruct instances of Java classes from the XML data. There are two kinds of parsers available for XML data parsing: SAX parser (Simple API for XML parsing) and DOM parser (Document object model). DOM views an XML document as a tree structure and loads the entire document into memory. It builds parent-child relationship between nested elements. The DOM API provides standard methods for querying the XML document and reconstructing Java objects. SAX is an event-based parser that fires different events based on the element parsed. A SAX parser, unlike the DOM parser, does not maintain a default model for the parsed data. If a listener can be set on the SAX parser to listen to parsing of specific tags, then whenever the corresponding element is parsed, the listener can construct a Java instance of the class corresponding to the element. The SAX parser used in Factory Model Generator is an instance of the Java class (javax.xml.parsers.SAXParser). This parser is used to construct factory models from the saved XML files. The listener for this parser is called as content handler
and is an instance of the Java class (org.xml.sax.helpers.DefaultHandler). Every data structure in the process plan representation model should be capable of producing a content handler that can reconstruct instances of the Java class from the XML element parsed. Figure 4-10 shows the mechanism of generating a content handler for Machine object.

```java
public class Machine extends ImpObject {
    public static ContentHandler getSAXHandler(ImpXmlReader reader, ImpObject parent) {
        return getSAXHandler(reader, parent);
    }
}
```

Figure 4-10 Generating an XML content handler

The content handler class for parsing the machine XML data is shown in Figure 4-11. For the Machine class, the content handler is defined as an inner class and an instance of the content handler can be obtained by calling a method called `getSAXHandler()`. XML data can then be parsed with this content handler.
public static class SAXHandler extends ImpXmlHandler {

   //The XML Reader Object
   ImpXmlReader machineReader;

   //The machine object to be created
   Machine mac;

   //The parent of this Machine object
   ImpObject parent;

   //This method is fired when the start element <Machine> is
   //parsed.
   public void startElement (String namespaceURI, String
   localName, String qName, Attributes atts)
   {
      mac = new Machine (atts.getValue ("machineName"));
      mac.setMachineCode (atts.getValue ("machineCode"));
      mac.setNumberOfUnits (atts.getValue ("number"));
      mac.setFrequency (atts.getValue ("frequency"));
   }

   //This method is fired when character data is parsed between
   //the start and the end element.
   public void characters (char [] ch, int start, int length) {};

   //This method is fired when the end element </Machine> is
   //parsed.
   public void endElement (String namespaceURI, String localName,
   String qName)
   {
   }
 }

Figure 4-11 Parsing machine XML data with a content handler

Separate events are generated corresponding to the parsing of
<startElement>, </endElement> and character data. The startElement () method
is called when the SAXParser parses the <Machine> tag. Since all the
information in the <Machine> element is available as attributes of the element,
the values corresponding to the attributes can be obtained from the `atts` argument of the method, as shown in the code above. Thus, the Java object `mac` corresponding to the `<Machine>` element tag parsed is constructed. The `characters` method is fired when characters are parsed between the `<startElement>` and the `<endElement>`. Since the `<Machine></Machine>` tag shown above does not have any characters, this method never gets fired. The `endElement()` method gets fired when the `<Machine>` tag is parsed by the `SAXParser`. This method can be used to deal with the instance of the Machine class constructed in the `startElement()` method. For example, this instance can be added to the `machineList` variable of the `MfgSystem` object, which holds this machine. The content handler of the `MfgSystem` class has already constructed the `MfgSystem` object by the time the Machine element is parsed. This `MfgSystem` object is passed to the machine content handler as a variable called `parent`. As the XML file is being parsed, the `XMLReader` class is capable of switching content handlers according to the XML data being parsed. There is a method in the (`javax.xml.parsers.SAXParser`) Java class through which content handlers can be set. This method is called `setContentHandler` (`DefaultHandler handler`). Thus, the entire factory model instance is constructed from the XML data. The Factory Model Generator initiates this process of constructing an instance of the factory model from the XML data.
4.3 Space searcher

Space Searcher is a tool for performing space search on a given problem and obtains a good or optimal solution. This technique can be used for several problems like Farmer-Goat, Missionary-Cannibal, Traveling salesman and the Cellular Manufacturing problem. These problems are defined as Java classes using the techniques of object-oriented programming. A Java interface called Searchable is defined to represent problems that can be solved with the Space Searcher tool. The Searchable interface defines methods that are essential for performing space search. Problems that need to be solved with the Space Searcher tool have to define Java classes that implement the Searchable interface. By implementing the Searchable interface, problems provide the mechanism for performing search initialization, state transformation function, search termination etc.

4.3.1 Searchable interface

A Java interface is a generic interface that defines the behavior of a family of Java classes. Problems that have to be solved with the space search technique have generic behavior provided by Searchable interface and need to provide implementation for it. Figure 4-12 shows the definition of Searchable interface for space search.
package edu.ohiou.implanner.spacesearch;

public interface Searchable
{
    public Collection makeNewStates();
    public Searchable runSpaceSearch();
    public Searchable runSpaceSearch(Searchable s);
    public boolean memberInList(Collection l);
    public String printPath();
    public boolean equals(Searchable s);
    public Searchable getParent();
    public DefaultMutableTreeNode getNode();
    public int[] setSearchTypes();
    public boolean canBeGoal();
    public boolean isBetterThan(Searchable inState);
    public Searchable getClone();
    public Comparator getComparator();
}

Figure 4-12 Searchable interface to perform space search

All these problems have to implement mechanism for initializing space search, producing child states with the state transformation function, evaluation of states to produce goal etc, state filtering and state ordering. The most important methods in the Java code above that influence space search are explained in this section.

• makeNewStates()

This method provides the implementation for the state transformation function for search progression. This method generates child states for the current state and helps build the parent-child relationship in the space search.
Java classes for the problems implement this method and return the child states for the current state in the search space.

- **runSpaceSearch (Searchable s)**

  This method performs the actual process of space search. It generates the child states for the current state, evaluates these states for the goal states, performs state evaluation and filtering of repeated states in the search space and selects the next state for evaluation. This method performs the recursive process of finding the path to the goal state for problems with defined goal and optimal search problems where the goal is not defined and the best solution is returned. Java classes don't have to provide implementation for this method because this is taken care of by the Space Searcher tool. This alleviates the builder of the Java classes from worrying about the execution control part of the space search and just focuses on problem class.

- **equals (Searchable s)**

  This method helps in the filtering and elimination process during the space search. This method compares the argument state s with the state on which this method is called and returns a *boolean* value. If the two states are the same, then the method returns a *boolean true* else the method returns *boolean false*. All the Java classes that have to use the Space Searcher tool properly have to provide a correct.
• **getParent ()**

  This method returns an instance of the *Searchable* object that is the direct parent of the current state. This method helps in building the parent-child relationship between a state and its child states. Classes don’t have to provide an implementation but the class constructor for the Java problems have to include a *Searchable* element, which is going to be the parent of the state under construction.

• **setSearchTypes ()**

  This method returns an integer array, which helps in enabling the different search types on Space Searcher GUI. Static integer variables are defined in the Space Searcher Java classes for enabling the different search types on the Space Searcher GUI. These integers are ‘1’ for depth-first search, ‘2’ for breadth-first search, ‘3’ for best-first search, ‘4’ for A* and ‘5’ for User-Guided search. By returning a proper integer array for the problem’s Java class, different kinds of search can be performed on the problem with the Space Searcher tool.

• **canBeGoal ()**

  This method is used to distinguish between states in the search space that are eligible for being goal state. For example, in the cellular manufacturing problem, a state in which an assignment to manufacturing cells is not found for a part model may not be eligible to be a goal state. This method returns a *boolean true* value if a state is eligible to be a goal state, otherwise it returns *boolean*
false. Implementation for this method has to be provided by the Java class of the problem being solved.

- isBetterThan (Searchable inState)

  This method also returns a boolean value. It is used to compare space-states that are eligible to be goal states. This method is used for solving optimization problems in which there is no defined goal state and hence a goal state has to be decided during the space search. The Java classes for the problems have to implement this method, which ensures that the Space Searcher tool remains independent of the problem being solved. If the argument state is better than the state on which this method is called, then this method returns a boolean true else the method returns a boolean false.

- getComparator ()

  This method provides a mechanism for sorting child states generated by the state transformation function. This is again optional for the problems on which space search needs to be performed. The farmer-goat problem does not need to implement a state ordering mechanism, but the cellular manufacturing algorithms need a state ordering mechanism as defined in Sections 3.1.4 and 3.2.5. The Java classes for the cellular manufacturing problems implement this method. The Space Searcher tool uses this method to perform state ordering on the generated child states.
4.3.2 Default space state

Because of the way Java is built, all the classes that implement the `Searchable` interface have to provide implementation for all the methods in the interface. In such a situation, it becomes cumbersome for Java classes to provide implementation for all the methods in the interface. For this purpose, an abstract Java class called `DefaultSpaceState` is defined to provide default implementation for all the methods in the `Searchable` interface. Java classes for the problems that need to be solved have to just extend this class. In this way, the Java classes can implement the minimum number of methods that are needed for performing space search with the Space Searcher tool.

4.3.3 Space searcher tool

Space Searcher is a Java Swing API based tool to perform space search for problem solving. A panel called Search Space panel is used to represent the search space generated as a tree structure. This tool was constructed as part of this thesis to solve common AI examples like farmer-goat-cabbage problem, missionary-cannibal problem and traveling salesman problem. Apart from this, this tool is also used to solve the problem of cellular manufacturing. The Space Searcher tool can perform the process of space search on a given problem with the algorithms of breadth-first, depth-first and best-first, A* and User-guided searches. Figure 4-13 shows the component-based description of the Space Searcher tool.
The current implementation of the Space Searcher tool has implementation for breadth-first, depth-first, best-first and User-guided searches. The tool displays the details about the current state under expansion, the list of child states available, current goal state (if one has been found) etc. The current state in the search space can be viewed in the Space Searcher. Options are also available for controlling execution of the search space. A panel is also available
to display the path of the goal state from the initial state for the purpose of backtracking. The object-oriented nature of Java enables viewing of different panels in the same tool for different problems. The hierarchical relationship between parent and child states is represented in a tree structure. A panel called algorithm selection panel is used to select the space search technique to be used to solve the current problem. Different techniques that are available are depth-first, breadth-first, best-first, A* and user-guided search techniques. A panel called State Inspection panel is used to display additional data about each space-state. A panel called execution control panel is used to control the progression of space search. This panel provides control to the user as to the number of steps of execution to be performed. Different options available are running one step that performs one state transformation, running $n$ number of steps that performs transformation on $n$ number of states, and runAll that performs space search until the desired goal state is found. A solution panel is also available to determine the path of a goal state from the initial state. The display in the state inspection panel varies with each problem being solved. Depending on the problem being solved, users need to see different data. The visuals for the traveling salesman problem should show the cities currently covered and the remaining cities. It also shows the distance between different cities, the current list of covered cities, the list of cities to be covered, the distance between the different cities and the goal state Figure 4-14 shows the inspection panel for the traveling salesman problem.
Figure 4-14: State inspection panel for the traveling salesman problem

Figure 4-15 shows the interface between the Space Searcher tool and a space search problem that has to be solved with the Space Searcher tool.
Figure 4-15: Interface between Space Searcher and space search problem

Figure 4-16 shows the interface for performing space search for cellular manufacturing from Factory model generator.

Figure 4-16 Interface for space search from FMG.
Space search can be configured from FMG. The options included are options for sorting machines according to the defined frequency functions, generation of Special Plant List, setting of search options like cell size, percent elimination etc and starting the process of space search on the problem. Selecting the Searcher option starts the Space Searcher tool with the part and machine information populated in the Space Searcher and the search can be started at this stage.

4.4 Implementation of space search algorithm for cell formation without alternative process plans

This section discusses the implementation of the space search algorithm for cell formation without alternative process plans discussed in Section 3.1. Description for Java class implementation and space search implementation are provided.

4.4.1 Implementation of Java classes

Each space state for the algorithm is represented by an instance of the Java class \textit{GTState} that extends from \textit{DefaultSpaceState} parent class and hence has to provide implementation for all the methods to perform space search on Space Searcher. Figure 4-17 shows the java class for representing a space state in this algorithm.
public class GTState extends DefaultSpaceState {
    private LinkedList machineCellsList = null;
    private TreeSet gtStateChildren = null;
    private double totalInterCellTraffic;
    private double lowerBound;
    public GTState (Collection machinesList, Collection partActivityList, int capacity) {
        // Initialize state variables.
    }
    public boolean canBeGoal () {
        return true;
    }
    public boolean equals (Searchable s) {
        // Compare this state with the argument state.
    }
    public GTStateComparator getComparator () {
        return new GTStateComparator ();
    }
    public Collection makeNewStates () {
    }
    public void computeLowerBound () {
        this.lowerBound = // Result of computation of lower bound.
    }
    public void calculateTotalInterCellTraffic () {
        this.totalInterCellTraffic = // Result of computation of traffic.
    }
    public boolean isBetterThan (Searchable state) {
        boolean flag = false;
        if (this.totalInterCellTraffic < ((GTState) state).totalInterCellTraffic) {
            flag = true;
        }
        return flag;
    }
}

Figure 4-17: Space State class for cell formation without alternatives.
The `machineCellsList` variable in `GTState` holds all the machine cells in the current state. The `gtStateChildren` variable in `GTState` class holds all the children of this instance of `GTState` produced by the state transformation function `makeNewStates()`. Decision is based on the variable `totalInterCellTraffic`, which holds the traffic value of this state. The method `calculateTotalInterCellTraffic()` computes this value for the current instance of `GTState` and stores the value in `totalInterCellTraffic`. Each machine cell is an instance of the Java class `MfgCell` shown in Figure 4-18.

```java
public class MfgCell extends Resource {
    private String cellName;
    private LinkedList machinesList;
    private String cellStatus; //Either open or closed.
    private int cellCapacity;
}
```

Figure 4-18: Java class for manufacturing cell holding machines.

Each instance of `MfgCell` holds a collection of Machine instances stored in the variable `machinesList`. The status of each cell (either “Open” or “Closed”) is stored as a string.
4.4.2 State transition mechanism

The transformation function determines new cells in a state by identifying two mergeable machine cells from the `machineCellsList` variable. Figure 4-19 shows a summary of the state transformation function in `GTState` class.

```java
public Collection makeNewStates ()
{
    MfgCell cell1 = new MfgCell ();
    int index1 = 0;
    //This procedure identifies the first machine cell to merge.
    for (Iterator itr = this.machineCellsList.iterator();
         itr.hasNext ();)
    {
        cell1 = (MfgCell) itr.next ();
        if (cell1.getCellStatus ().equals (“Open”))
        {
            index1 = this.machineCellsList.indexOf(cell1);
            break;
        }
    }
    //This procedure identifies the second machine cell to merge.
    MfgCell cell2 = new MfgCell ();
    int index2 = index1+1;
    for (int i=index2;i<this.machineCellsList.size (); i++)
    {
        cell2 = (MfgCell) this.machineCellsList.get (i);
        if (canBeMergedWith (cell1, cell2))
        {
            //Construct a child state
            GTState childState = new GTState ();
            // Merge cell1 and cell2 to form new cell.
            MfgCell mergedCell = cell1.mergeCell (cell2);
            childState.machineCellsList.add (mergedCell);
            this.gtStateChildren.add (childState);
        }
    }
    //Form nonmergeable state and add to gtStateChildren.
    //Return child states
    return gtStateChildren;
}
```

Figure 4-19: State Transition mechanism for cell formation without alternatives.
After identifying two mergeable cells, the cells are merged into one cell, a new state is created with the newly formed cell and other cells pertinent to the child state. After this, the non-mergeable state is created and then added as a child state. The newly formed states are accumulated into a collection of objects and then returned from the method call to the caller of this method.

4.4.3 State comparison, ordering and goal switching

During space search, states are sorted in some order to perform efficient search. Sorting mechanism is introduced in the form of comparator class. Figure 4-20 shows the of the Comparator interface defined in `gtStateChildren`.

```java
gtStateChildren = new TreeSet<GTStateChildren>();
class GTStateComparator implements Comparator
{
    public int compare(GTStateChildren o1, GTStateChildren o2)
    {
        if (o1.equals(o2))
        {
            return 0;
        }
        if (((GTState) o1).lowerBound > ((GTState) o2).lowerBound
        {
            return 1;
        }
        else
        {
            return -1;
        }
    }
}
```

Figure 4-20: State comparison mechanism for CF without alternatives.
The states in the space search algorithm without alternative process plans are sorted according to the result of the computation of the lower bound on traffic value. Java’s mechanism to perform ordering on a set of related objects is with comparators. The variable `gtStateChildren` holds all children of the current state in the search. The method `makeNewStates()` populates `gtStateChildren` into the child states. When the transformation function generates new child states and adds them to `gtStateChildren`, the comparator calls the `compare()` method with two states that are to be compared. If state o1 has a lower bound greater than state o2, then o1 is added after o2. If state o1 has a lower bound less than state o2, then o1 is added before o2. If state o1 and o2 have equal lower bound, then one state is not added to the list. The child states are sorted according to the selected comparison function. The sorted child states are given to Space Searcher, which displays the child states under the current state as tree nodes. In the course of space search, if the Space Searcher has to make a decision as to whether the current state is better than the current goal state, then the `canBeGoal()` and `isBetterThan(Searchable state)` method is called on `GTState` with the argument state as the state to be compared with the goal state. For the cell formation algorithm without alternative process plans, all states can be considered as goal states and hence the `canBeGoal()` method returns true. A switch of goals is to be made if the current state can be a goal state and has total intercellular traffic value less than the current goal state. Figure 4-21 shows the goal switching mechanism for this algorithm.
4.5 Implementation of space search algorithm for cell formation with alternative process plans

This section discusses the implementation of the space search algorithm for cell formation with alternative process plans described in Section 3.2.

4.5.1 Implementation of Java classes

Each space state for the algorithm is represented by an instance of the Java class `GTStateAltPlans` that extends from `DefaultSpaceState` parent class. This class holds information about the space states in the solution domain of this problem. Each variable in this class represents a property of this state. Figure 4-22 shows the Java class to represent the space state for this algorithm.
public class GTStateAltPlans extends DefaultSpaceState {
    private Machine keyMachine;
    private PartFamily rootFamily;
    private Collection originalMachines;
    private Collection originalParts;
    private Collection families = new LinkedList();
    private static int cellSize;
    private int percentEliminate;
    private Collection childStates;
    public GTStateAltPlans(Collection partsList, Collection machinesList, Collection machinesUnitsList, GTStateAltPlans parent, PartFamily family, int cellSize, int percentEliminate, boolean allowTraffic, boolean allowSubFamilies) {
    }
    public boolean canBeGoal () {
        return this.numModels == MfgSystem.numModels;
    }
    public void findStateTrafficCount () {
        this.overallTrafficCount =
            this.rootFamily.getTrafficCount() +
            ((GTStateAltPlans)this.parent).rootFamily.
            getTrafficCount();
    }
    public boolean isBetterThan (Searchable state) {
        GTStateAltPlans instate = (GTStateAltPlans)state;
        return this.overallTrafficCount <
            instate.overallTrafficCount;
    }
    public static GTStateAltPlansComparator getComparator() {
        return new GTStateAltPlansComparator();
    }
    public Collection makeNewStates () {
    }
}

Figure 4-22: Space State Java class for CF with alternatives.
The \textit{keyMachine} variable is the key machine for the state being constructed. The \textit{rootFamily} variable is the part family of the state under consideration. The \textit{originalMachines} list is the unassigned machines list from the parent state. The \textit{originalParts} list is the unassigned part activity list from the parent state. The variable families holds the families formed as a result of the state transformation function being applied on the current state. The \textit{cellSize} variable holds the maximum cell size for the states being explored. The \textit{percentEliminate} variable holds the percentage of child states to be eliminated from the states being explored. The \textit{childStates} list holds all the child states generated by applying the state transformation algorithm on the current state in the search. The variable \textit{overallTrafficCount} holds the number of exceptional machine operations for the current state.

4.5.2 State transition mechanism

The state transformation function for this algorithm generates space states based on incremental sized part families. The \textit{makeNewStates()} method is the entry point for Space Searcher when state transformation function has to be applied on the current state in the search procedure. First, a key machine is identified from the list of unassigned machines. Then, based on the part activities on this key machine, incremental sized part families are formed by recursion. These families become the root families for the child states being formed. Assignment of parts to part families and machines to machine cells are to be
done in different ways depending on the number of machines and parts available. Figure 4-23 shows the implementation of this algorithm in `GTStateAltPlans` class.

```java
public Collection makeNewStates() {
    newStates = new LinkedList();
    if(this.originalMachines.size()==0) {
        newStates.addAll(this.handleEndAssignment());
    }
    if(this.originalParts.size()!=0 &&
        this.originalMachines.size()!=0) {
        newStates.addAll(
            this.handleUsualAssignment());
    }
    return newStates;
}
```

Figure 4-23: State transition mechanism for CF with alternatives

If all the machines have found assignment, then `handleEndAssignment()` method is called. This assigns the routings of unassigned parts to cells in such a way that the number of exceptional elements introduced into the cell configuration is minimal. Figure 4-24 shows the implementation of `handleUsualAssignment()`.
private LinkedList handleUsualAssignment()
{
    LinkedList test = new LinkedList();
    //This machine identifies the key machine for
    //this state
    this.setKeyMachine();
    //This method identifies all part activities on
    //key machine
    this.findPartsForKeyMachine();
    //These methods form incremental part families.
    this.formInitialPartFamilies();
    this.formPartFamilies(this.families,1);
    //This method eliminates child states based on
    //percent elimination.
    this.percentElimination();
    //This method removes sub families.
    if (!this.allowSubFamilies)
        this.removeSubFamilies();
    //Generate child states based on part families formed.
    this.constructChildStates();
    test.addAll(this.childStates);

    //Returns the list of available child states.
    return test;
}

Figure 4-24: Helper method to perform state transformation

If there are machines remaining that need assignment, then
handleUsualAssignment() method is called. This method performs the process
of identifying the next key machine, the part activities to be selected for the key
machine, elimination of families that do not conform to the minimum size criteria.
After the formation of families, this method constructs the child states to be
returned to space search for continued search. The child states include
information about the unassigned parts, machines and the number of exceptional
machine operations. The constructChildStates() method generates the
necessary child states and populates the childStates variable. Figure 4-25 shows
the construction of child states for the current state in space search.

```java
private void constructChildStates ()
{
    childStates = new TreeSet
        (GTStateAltPlans.getComparator ());
    this.findAdditionalPartsForFamilies ();
    LinkedList tempFamiliesList = new LinkedList ();
    for (Iterator itr = this.families.iterator ();
        itr.hasNext ();)
    {
        ParFamily family = (PartFamily) itr.next ();
        if (family.getPartsList ().size ()>minSize)
        {
            if (!tempFamilies.contains (family))
            {
                family.findTrafficCount ();
                GTStateAltPlans childState   = new
                    GTStateAltPlans(
                        family.getUnassignedParts (),
                        family.getUnassignedMachines (),
                        family.getMachineUnitsList (),
                        this,family,cellSize,
                        percentEliminate,allowTraffic,
                        allowSubFamilies);
                childState .parentFamilies.addAll
                    (this.parentFamilies);
                childState .setTrafficCount(
                    family.getTrafficCount () +
                    this.getTrafficCount ());
                tempFamiliesList.add(family);
                childStates.add(childState );
            }
        }
    }
}
```

Figure 4-25: Construction of child states for the current state in space search.
After each child state is constructed, the necessary data from parent state are populated into the child state. Thus, when Space Searcher calls `makeNewStates()` method of GTStateAltPlans, newly generated states are returned.

4.5.3 State comparison, ordering and goal switching

The newly generated child states are sorted in increasing order of the number of exceptional machine operations as discussed in Section 3.2.5. Child states are considered in the sorted order for expansion during space search. Figure 4-26 shows the comparator java class for this algorithm.

```java
static class GTStateAltPlansComparator implements Comparator {
    public int compare (Object o1, Object o2) {
        GTStateAltPlans ob1 = (GTStateAltPlans) o1;
        GTStateAltPlans ob2 = (GTStateAltPlans) o2;
        if (ob1.overallTrafficCount<=ob2.overallTrafficCount) return -1;
        else return 1;
    }
}
```

Figure 4-26: State comparison mechanism for cell formation with alternatives.
The `getComparator()` method returns an instance of `GTStateAltPlansComparator`. This class performs the actual process of comparison and sorting. The separation of the sorting mechanism from the process of space search helps in defining different comparators for different kinds of problems. For the cell formation algorithm with alternative process plans, a state can be a goal state if all part models has found assignment to part families. Space Searcher makes a switch of goal state by calling `isBetterThan()` method on all states in the search space that can be goal states. A switch of goals occurs if all parts have found assignment to part families and the number of exceptional machine operations for the current state is less than the current goal state. When the Space Searcher has to compare goal states, then it calls the `canBeGoal()` and `isBetterThan(Searchable state)` methods shown in Figure 4-27.

```java
public boolean canBeGoal() {
    return (this.numModels==MfgSystem.numModels);
}

public boolean isBetterThan (Searchable state) {
    GTStateAltPlans instate = (GTStateAltPlans)state;
    return this.overallTrafficCount <
           instate.overallTrafficCount;
}
```

Figure 4-27: Goal switching mechanism for cell formation with alternatives.
5 Testing and examples

This section covers the testing on examples from literature conducted to study the performance of the space search cell formation algorithm with alternative process plans. This algorithm has been tested on several examples from literature as summarized in Table 5-1.

<table>
<thead>
<tr>
<th>Example</th>
<th>No. of Parts</th>
<th>No. of Machines</th>
<th>Alternative plans</th>
<th>Multiple machine units</th>
<th>Improvement</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adil</td>
<td>10</td>
<td>10</td>
<td>Yes</td>
<td>No</td>
<td>No (2,1)</td>
<td>[1]</td>
</tr>
<tr>
<td>Burbidge</td>
<td>35</td>
<td>20</td>
<td>No</td>
<td>No</td>
<td>Same (2,2)</td>
<td>[12]</td>
</tr>
<tr>
<td>Jae</td>
<td>15</td>
<td>8</td>
<td>Yes</td>
<td>No</td>
<td>Same (1,1)</td>
<td>[8]</td>
</tr>
<tr>
<td>Lee 1</td>
<td>13</td>
<td>8</td>
<td>Yes</td>
<td>No</td>
<td>Same (1, 1)</td>
<td>[18]</td>
</tr>
<tr>
<td>Lee 2</td>
<td>40</td>
<td>30</td>
<td>Yes</td>
<td>No</td>
<td>Same (1,1)</td>
<td>[18]</td>
</tr>
<tr>
<td>Kasilingam</td>
<td>15</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes (2,6)</td>
<td>[13]</td>
</tr>
</tbody>
</table>

Each example problem was tested based on certain parameters like variation in percentage elimination of new states, virtual memory allocation for the Java run time environment, maximum cell size etc. Table 5-2 shows the testing parameters for the cell formation algorithm with alternative process plans based on which testing of the algorithm was performed.
Table 5-2: Testing parameters for the cell formation with alternatives.

<table>
<thead>
<tr>
<th>State name</th>
<th>% Elimination</th>
<th>Number of steps</th>
<th>Open list size</th>
<th>Cell size</th>
<th>Memory limit</th>
<th>Exceptions</th>
<th>Solution</th>
</tr>
</thead>
</table>

Data that was recorded include the size of the Open list and Closed list of the best-search algorithm, number of exceptional elements generated, solution type etc. The percent elimination column indicates the percentage of child states eliminated in order to minimize the search space as discussed in Section 3.2.5.2. The open list size holds information on the number of states yet to be visited while the closed list size holds information about the number of visited states in the space search. The maximum cell size column indicates the maximum size of each machine cell in the space state. The memory limit column indicates whether the Java Virtual Machine throws up the specified error when performing space search. An ‘yes’ in this column indicates that the JVM has indeed run out of memory and the solution, if any, returned by the Space Searcher is the best solution obtained until that point in the space search. A ‘no’ in this column indicates that the computer has completed the space search and selected the best state among all visited states. These simulation runs are conducted with a cut-off time of three minutes. This means that if the Space Searcher does not produce the child states from a state transformation for more than three minutes,
then the search for a solution is terminated. If a goal is found, it is recorded, otherwise, the search does not produce a solution.

The data about the factory model is generated with Factory Model Generator and saved out in XML. Figure 5-1 shows the configuration of space search information from the Factory model generator.

Figure 5-1: Configuration of space search from FMG.
Configuration of data for space search is also done from the Factory Model Generator. For the cell formation algorithm with alternative process plans, data like maximum cell size and percentage elimination are initialized in the Factory Model Generator. The configured space search is then performed on Space searcher until a goal is found. The goal state is displayed on Space Searcher as a cell configuration with parts assigned to families and machines assigned to machine cells.

5.1 Jae example

This section elaborates on the testing performed on Jae example [8]. It includes a discussion about data generation, results from performance testing and evaluation of different cell configurations.

5.1.1 Data generation

The factory model from [8] has 15 parts P1-P15 and 8 machines M1-M8. Each part model has one or more alternative process plans. Figure 5-2: shows the problem data generated with Factory model generator. Figure 5-3 shows the Special Plant list for this problem.
Figure 5-2: Part-machine incidence matrix for the 15 parts by 8 machines problem.

Figure 5-3: Special plant list for Jae.
The list shows the machines in the factory model, their SICGE code, number of units of each machine and the usage frequency.

5.1.2 Performance results from testing

Table 5-3 shows the performance testing results for this example from space search.

<table>
<thead>
<tr>
<th>State name</th>
<th>% Elimination</th>
<th>Number of steps</th>
<th>Open list size</th>
<th>Cell size</th>
<th>Memory limit</th>
<th>Exceptions</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor1</td>
<td>0</td>
<td>309</td>
<td>8913</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor2</td>
<td>5</td>
<td>309</td>
<td>8913</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor3</td>
<td>10</td>
<td>309</td>
<td>8913</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor4</td>
<td>15</td>
<td>309</td>
<td>8913</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor5</td>
<td>20</td>
<td>309</td>
<td>8913</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor6</td>
<td>25</td>
<td>757</td>
<td>7903</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor7</td>
<td>30</td>
<td>1208</td>
<td>6088</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor8</td>
<td>35</td>
<td>2299</td>
<td>2628</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor9</td>
<td>40</td>
<td>2664</td>
<td>1422</td>
<td>6</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor10</td>
<td>45</td>
<td>2569</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor11</td>
<td>50</td>
<td>1598</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor12</td>
<td>55</td>
<td>1598</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor13</td>
<td>60</td>
<td>769</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor14</td>
<td>65</td>
<td>769</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor15</td>
<td>70</td>
<td>53</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor16</td>
<td>75</td>
<td>33</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>Factor17</td>
<td>80</td>
<td>33</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>Factor18</td>
<td>85</td>
<td>33</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>Factor19</td>
<td>90</td>
<td>18</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor20</td>
<td>95</td>
<td>18</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor21</td>
<td>100</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>
Space search was performed by varying the percentage elimination factor from 0 to 100 in steps of 5 for virtual memory allocation of 512 MB and maximum cell size of 6. One solution state can be achieved for each variation of the test parameters provided the JVM does not run out of memory. For example, Factor 11 corresponds to 50% elimination of child states with a maximum cell size of 6. For this configuration, number of states explored is 1598. At the end of the search, open list size is zero as all possible states have been explored. The computer does not run out of memory and the best solution produced is A with an exception count of 1. Three different cell configurations (A, B, C) are produced by space search. While 17 configurations produce solution A, 3 configurations produce solution B while 1 configuration produces solution C. Twenty input configurations produce one exceptional machine operations in their solution state while one configuration produces two exceptions.

5.1.3 Comparison and evaluation of cell configurations

Of the three cell configurations, B is a subset of A as the only difference between A and B is that A has machine M5 included in the second cell while B does not have M5 in its configuration. As A has M5 included its second cell, it has additional part activities included for parts that can be completely manufactured with M5, M6, M7 and M8 (The second machine cell). Configuration A is shown in Table 5-4 in context of the initial part machine matrix shown in Figure 5-2.
Table 5-4: Best solution for Jae from space search (Configuration A)

<table>
<thead>
<tr>
<th>Part</th>
<th>Alternative</th>
<th>Selection</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
<th>M8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>A1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>A1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>A1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>A1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>A1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>A1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>A1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>A1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td>A1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td>A1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P11</td>
<td>A1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P12</td>
<td>A1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P13</td>
<td>A1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The only exception in this case is P7 A2 on M6. The two machine cells are (M1, M2, M3) and (M5, M6, M7 and M8) respectively. Machine M4 is not included in configurations A and B as it is not needed by any of the alternatives selected for family formation. Figure 5-4 shows 2 possible cell configurations (A and C) obtained from Space Searcher with their exceptional machine operations.

Figure 5-4: Possible cell configurations (A, C) obtained.
The cell configuration on the left was decided as the best solution. Table 5-5 shows the solution for this problem reported in Jae [8].

<table>
<thead>
<tr>
<th>Part/Machine</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M6</th>
<th>M7</th>
<th>M8</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 A1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2 A3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3 A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9 A1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P11 A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P12 A1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P14 A1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4 A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5 A3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6 A3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7 A1</td>
<td>[ ]</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10 A1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P15 A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8 A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P13 A2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Three cells are formed, as shown above and machine M4 is also included. The exceptional machine operation is P7 A1 on M2.

5.1.4 Evaluation of presence of alternatives on quality of solution

The effect of elimination of alternative process plans for parts was studied on this example. Variation was performed on the maximum number of
alternatives a part can have and its effect on the number of exceptional elements in the cell configuration from space search was studied. Testing was performed with each part having a maximum of one and two alternatives as shown in Table 5-6.

Table 5-6: Performance testing on Jae with elimination of alternatives with 512 MB VM allocation

<table>
<thead>
<tr>
<th>State name</th>
<th>% Elimination</th>
<th>Open list size</th>
<th>Cell size</th>
<th>Memory Limit</th>
<th>Exceptions</th>
<th>Solution</th>
<th>Max Plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor 1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>No</td>
<td>7</td>
<td>A-1</td>
<td>1</td>
</tr>
<tr>
<td>Factor 1</td>
<td>0</td>
<td>10612</td>
<td>6</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>Factor 11</td>
<td>50</td>
<td>2822</td>
<td>6</td>
<td>Yes</td>
<td>2</td>
<td>A-2</td>
<td>2</td>
</tr>
<tr>
<td>Factor 16</td>
<td>75</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>2</td>
<td>A-2</td>
<td>2</td>
</tr>
<tr>
<td>Factor 17</td>
<td>80</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>2</td>
<td>A-2</td>
<td>2</td>
</tr>
<tr>
<td>Factor 18</td>
<td>85</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>2</td>
<td>A-2</td>
<td>2</td>
</tr>
<tr>
<td>Factor 19</td>
<td>90</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>2</td>
<td>A-2</td>
<td>2</td>
</tr>
<tr>
<td>Factor 20</td>
<td>85</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>2</td>
<td>A-2</td>
<td>2</td>
</tr>
<tr>
<td>Factor 21</td>
<td>100</td>
<td>0</td>
<td>6</td>
<td>No</td>
<td>2</td>
<td>A-2</td>
<td>2</td>
</tr>
</tbody>
</table>

The purpose of this testing is to study the effect of elimination of alternatives on the solution quality in terms of the number of exceptions. The testing table has one additional column that just indicates the maximum number of alternatives that a part is allowed to have. Part activities were selected
randomly for the purpose of testing. Factor 1 corresponding to maximum alternatives of 1 for each part produces 7 exceptions (Solution A-1). In this case, a complete exploration of the search space is achieved. Factor 1 corresponding to maximum of two alternatives for each part does not produce a solution as the computer runs out of memory. The remaining factors produce the solution A-2 and have two exceptions. In both cases, it is seem that reduction of alternatives degenerates the solution. Figure 5-5 shows the results obtained from performing space search on this problem with reduced alternatives.

Figure 5-5 Solution A-1 and A-2 from Table 5-6
In both cases, the number of exceptions is higher than the one produced on the complete problem. This verifies the importance of presence of alternative process plans during cell formation. The process planner should make careful decisions during generation of alternative process plans.

5.2 Lee 2

This section elaborates on the data generation, performance testing and solution evaluation performed on Lee [18].

5.2.1 Data generation

The factory model for this problem from [18] has 40 part models P1-P40 and 30 machines M1-M30 shown in Figure 5-6 and Figure 5-7.
Figure 5-6: Part-machine incidence matrix for Lee 2 (P1-P22)
Figure 5-7: Part-machine incidence matrix for Lee 2 (P23-P40)
Figure 5-8 shows the SPL information for machines in this factory model.

5.2.2 Performance results from testing

Space search was performed on this problem by varying the percentage elimination factor from 0 to 100 in steps of 5 for virtual memory allocation 512 MB and a maximum cell size of 9.

Table 5-7 shows the performance testing results corresponding to a memory allocation of 512 MB.
Table 5-7: Performance testing with 512 MB memory allocation.

<table>
<thead>
<tr>
<th>State name</th>
<th>% Elimination</th>
<th>Number of steps</th>
<th>Open list size</th>
<th>Cell size</th>
<th>Memory limit</th>
<th>Exceptions</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factor1</td>
<td>0</td>
<td>19</td>
<td>685</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor2</td>
<td>5</td>
<td>19</td>
<td>685</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor3</td>
<td>10</td>
<td>19</td>
<td>685</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor4</td>
<td>15</td>
<td>19</td>
<td>685</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor5</td>
<td>20</td>
<td>19</td>
<td>685</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor6</td>
<td>25</td>
<td>19</td>
<td>685</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor7</td>
<td>30</td>
<td>19</td>
<td>677</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor8</td>
<td>35</td>
<td>58</td>
<td>1583</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor9</td>
<td>40</td>
<td>44</td>
<td>1296</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor10</td>
<td>45</td>
<td>44</td>
<td>1228</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor11</td>
<td>50</td>
<td>80</td>
<td>1669</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor12</td>
<td>55</td>
<td>80</td>
<td>1669</td>
<td>9</td>
<td>Yes</td>
<td>No memory</td>
<td>No</td>
</tr>
<tr>
<td>Factor13</td>
<td>60</td>
<td>110</td>
<td>1470</td>
<td>9</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor14</td>
<td>65</td>
<td>130</td>
<td>1865</td>
<td>9</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor15</td>
<td>70</td>
<td>212</td>
<td>1480</td>
<td>9</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor16</td>
<td>75</td>
<td>265</td>
<td>1332</td>
<td>9</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor17</td>
<td>80</td>
<td>303</td>
<td>1049</td>
<td>9</td>
<td>Yes</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>Factor18</td>
<td>85</td>
<td>313</td>
<td>874</td>
<td>9</td>
<td>Yes</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>Factor19</td>
<td>90</td>
<td>411</td>
<td>906</td>
<td>9</td>
<td>Yes</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>Factor20</td>
<td>95</td>
<td>411</td>
<td>906</td>
<td>9</td>
<td>Yes</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>Factor21</td>
<td>100</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>No</td>
<td>22</td>
<td>C</td>
</tr>
</tbody>
</table>

Three different cell configurations (A, B, C) are obtained by performing space search with exceptional machine operations of 1, 7 and 22 respectively. 9 input configurations produce a result while for the remaining; the computer runs out of memory before a solution can be found.
5.2.3 Comparison and evaluation of cell configurations

By manual evaluation of the solutions obtained by space search, solution A from Table 5-7 was decided as the best solution as only one exceptional machine operation is found. Figure 5-9 shows solution A.

![Figure 5-9: Cell configuration A.](image-url)
Factor 13 to Factor 17 produces this solution corresponding to percentage elimination of 60 to 80 percentages. Six part families and machine cells are formed. The exceptional machine operation for solution A is for part P9 A1 on M8. Figure 5-10 show solution B respectively. Factor 18 to Factor 20 produces this solution. In this solution also, six part families and machine cells are formed. Exceptions for B are P38 A2 on M11, P9 A1 on M8, P1 A1, P11 A2, P16 A2, P18 A1 and P35 A1 on M15.

Figure 5-10: Cell configuration B.
The solution reported by Lee also has one exception but for P9 A3 on M1. Four machine cells, namely, \{M1, M2, M21, M22, M24\}, \{M3, M4, M5, M23, M25, M27, M29\}, \{M10, M12, M13, M17\}, \{M26, M28, M30, M7\} are the same as reported by Lee.

5.3 Other examples

The space search algorithm was tested on other examples from literature to study performance. The best solutions obtained by space search for problems Burbidge [12] and Lee 1 [18] are shown in Figure 5-11 and Figure 5-12 respectively. The solutions returned contained one and two exceptions respectively.

![Table and Diagram]

Figure 5-11: Best solution for Lee 1 by space search.
Figure 5-12 shows the best solution obtained for Kasilingam [13]. This example has multiple units of machines M1 and M10. Two part families and two machine cells are formed. The first part family has 6 parts and the corresponding machine cell has 5 machines. The second part family has 9 parts and 8 machines. Two units of machines M1 and M10 are used in both the cells. The solution obtained has two exceptions.
Figure 5-13: Best solution for Kasilingam by space search.
6 Conclusions and Future Work

This chapter summarizes the research performed on cell formation with parts having alternative process plans with state space search. The contributions as well as future extensions to this research work are discussed extensively.

6.1 Research summary

The emphasis of this research has been on the development of a space search algorithm for cell formation in the presence of alternative process plans. Two software prototypes were developed a) Factory Model Generator for automating the process of data generation for factory models and b) Space Searcher for performing space search on the factory models to obtain cell arrangement. The algorithm was tested on several examples from literature. The Space Searcher tool was built in such a way so that it could display the result of the algorithm in a tabular format so that visual analysis of the solution enables better understanding of the performance of the algorithm.

6.2 Contributions

The contributions of this research are summarized based on the perspective of the space search algorithm and the software prototypes developed. The contributions towards the development of space search algorithm are summarized as follows:
• A space search cell formation algorithm was developed to form manufacturing cells and part families. The algorithm is capable of working with or without alternative process plans for part models. Formation of manufacturing cells is based on SICGE classification for machines. A machine cell and a root family define each space state. The algorithm performs space search to find a goal state, which is defined to have the minimum number of exceptional operations and generates a complete assignment of parts to different manufacturing cells. State expansion is based on a key machine in the current state and the candidate parts that require that key machine.

• A user defined expansion criteria was developed to restrict the number of states explored for finding optimal solution. The expansion criterion is used to decide which states should be included in space search. The expansion criterion is such that bigger part families are retained for expansion and smaller part families are eliminated.

• The algorithm is capable of handling multiple units of a machine. They are handled in such a way that if there is demand for a machine in a particular state, then that machine is assigned to that state and the number of available units of this machine is decremented by one. This ensures that the algorithm does not introduce exceptional machine operations in the presence of multiple units of a machine.
The space search algorithm for manufacturing cell formation was successfully tested on several examples from literature with and without alternative process plans in the presence and absence of multiple machine units. The efficiency of the algorithm was verified by recording the following run time parameters: the number of states explored, allocation of virtual memory to JVM, maximum cell size and percentage elimination values ranging from zero to hundred.

The contributions of this research towards the development of software tools for cellular manufacturing are summarized as follows:

- A tool Factory Model Generator was developed to build interactively factory models with details of parts and machines. The information about parts includes routing information including alternatives, cost of each part, number of units of each part to be manufactured etc. Information about machines includes SICGE classification for each machine, and number of units etc. The generated model can be saved into XML file and later retrieved for repeated run of algorithm.

- The Factory Model generator provides facility to configure space search. Options include sorting of machines according to different usage frequencies, information input for space search like maximum cell size, percentage elimination on newly generated states etc.
A tool Space Searcher was developed to perform space search on a specified problem. The Space Searcher tool was integrated with the cell formation algorithm with alternatives. The manufacturing cells formed were displayed in a cellular layout on the Space Searcher tool.

6.3 Limitations

This research though successfully tested, is somewhat limited in terms of the cell formation algorithm and its implementation. Identified limitations of the algorithm and prototype are:

- The space search algorithm defines a percentage elimination criteria based on the number of candidate part models that are selected for the key machine. Newly generated child states are eliminated based on this elimination criterion. This percentage elimination is made as a user-entered data, based on which states are eliminated. Due to this criterion, the algorithm cannot guarantee the optimal solution for a given problem.

- Load distribution is not considered in the space search algorithm. Any factory that manufactures hundreds of parts will have multiple units of a machine. The decision as to how many units of a machine have to be included in a cell increases the complexity of the space search algorithm.

- Only one criterion for machine ordering is considered. Machines are being sorted according to SICGE and usage frequency. The effect of using a
different sorting mechanism for the machines has not been tested completely. Machine usage frequencies may change because of assignment of machines to manufacturing cells. Frequencies are calculated only once at the beginning of the search. The effect of recalculating the usage frequencies at every stage of state transformation has not been studied.

- The Space Searcher tool is not capable of rendering the exceptional elements correctly when multiple units of a machine are present. The current implementation of the Space Searcher tool displays an exceptional element for a part, which has activity on a multiple unit machine, but the cell in which the part is assigned does not have that machine. In this case, only one exceptional element should be shown.

- The current implementation is not capable of storing out a space state into XML and regenerating the saved state on to a GUI. This capability would make the solution portable and the manufacturing cells formed can be saved and utilized in other manufacturing planning tasks, like layout design.

6.4 Future extension

Possible future extensions of this research work are summarized briefly. Successful testing of the algorithm and tools has resulted in the identification of possible extensions to this research.
• The space search algorithm for cell formation was tested on examples from literature with and without alternative process plans. In some cases, the results obtained are better than the solution reported in the literature. In some cases, the solution by space search algorithm has more exceptions than the one reported in literature. Further testing of the algorithm on larger number of examples is needed. In the case where the quality of the solution is the same, a time based evaluation criteria is needed.

• A heuristic evaluation function could be developed to substitute the percentage elimination criteria. The heuristic should be able to take a correct direction that will lead to minimum number of exceptional elements and also eliminate child states that produce bad solution. Such a heuristic that confirms to requirements of A* algorithm would ensure that the solution found is the optimum.

• Different criteria for machine sorting can be explored to study the effect on the solution quality. Different usage frequencies can be defined based on which the machine may be sorted. Also, the effect of performing sorting of machines after each transformation has to be studied. This may lead to formation of better manufacturing cells with reduced exceptional elements. Further testing is needed to study the effect of these parameters on cell formation.
The algorithm could be extended to include loading considerations to decide how many units of a particular machine should be assigned to a cell and how this assignment is going to affect the formation of manufacturing cells.
REFERENCES


The Java programming language, Sun Microsystems, Available from the Internet, URL: http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html