INTEGER PROGRAMMING AND HEURISTIC METHODS FOR THE CELL FORMATION PROBLEM WITH PART MACHINE SEQUENCING

by

Grammatoula Papaioannou

A DOCTORAL THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF

DOCTOR OF PHILOSOPHY

OF LOUGHBOROUGH UNIVERSITY

December 2007

Αφιερωμένο στο σύζυγό μου, στους γονείς μου και σε όλους όσοι βοήθησαν ώστε να γίνει εφικτή αυτή η διδακτορική διατριβή.

Dedicated to my husband, my parents and all the people that made this thesis possible.

"And the science which knows to what end each thing must be done is the most authoritative of the sciences, and more authoritative than any ancillary science; and the end is the good of that thing, and in general the supreme good in the whole of nature".

Aristotle (384-322 B.C.), Metaphysics

"ἀρχικωτάτη δὲ τῶν ἐπιστημῶν, καὶ μᾶλλον ἀρχικὴ τῆς ὑπηρετούσης, ἡ γνωρίζουσα τίνος ἕνεκέν ἐστι πρακτέον ἕκαστον"

Αριστοτέλη, Μεταφυσικά,
«περὶ τὰς αἰτίας καὶ τὰς ἀρχὰς ἐπιστήμη»

Contents

Synopsis ix

List of Publications x

Acknowledgements xi

List of Tables xii

List of Figures xiv

1 Group Technology & Cellular Manufacturing 1

1.1 Introduction 1
1.2 Traditional Production Systems 2
1.3 Cellular Manufacturing 4
1.3.1 Introduction 4
1.3.2 Benefits of Cellular Manufacturing 6
1.4 Cellular Manufacturing Design 8
1.4.1 Machine-Part Matrix Formulation 10
1.4.2 Drawbacks in Current CM Design Methods 12
1.5 Motivation for this Thesis 13
1.6 Work Addressed in this Thesis 14
1.6.1 Thesis Structure 15
1.7 Published Work 16
1.8 Thesis Contributions 16

2 Literature Survey 18

2.1 Introduction 18
2.2 Cell Formation - Existent Methodologies 19
2.2.1 Clustering 19
• Array-Based Clustering 20

• Hierarchical Clustering . . . 22
• Non-Hierarchical Clustering . . . 26
2.2.2 Graph Partitioning Approaches . . . 28
2.2.3 Mathematical Programming Methods . . . 30
2.2.4 Heuristics . . . 34
2.2.5 Metaheuristics . . . 35
• Simulated Annealing and Tabu Search . . . 35
• Evolutionary Algorithms and Ant Colony Optimisation . . . 37
2.2.6 Fuzzy Logic . . . 39
• Fuzzy Clustering . . . 39
• Fuzzy Mathematics . . . 41
• Fuzzy Mathematical Programming . . . 42
2.3 Summary . . . 42

3 Mixed Integer Mathematical Programming Model for CF **43**
3.1 Introduction . . . 43
3.2 Initial model for CF . . . 43
3.2.1 Notation for model formulation . . . 44
• Index Set . . . 44
• Input Parameters . . . 44
• Decision Variables . . . 45
3.2.2 Model Formulation and Solution . . . 45
3.3 An Extended Cell Formulation Model . . . 51
3.3.1 Solution for Extended Model . . . 52
3.4 Complete Model Formulation . . . 55
3.4.1 Notation for model formulation . . . 55
• Index Set . . . 55
• Input Parameters . . . 55
• Decision Variables . . . 55
3.4.2 Model Formulation . . . 55
• Illustration of Complete Model's Operation: Example (I) . . . 58
3.5 Summary . . . 62

4 Fuzzy Mathematical Programming for CF **63**
4.1 Introduction . . . 63
4.2 Fuzzy Set Theory and Linear Programming . . . 64
4.2.1 Membership Functions and Fuzzy Operators . . . 66
4.3 Fuzzy Cell Formation Models . . . 69

4.3.1 'Min' Aggregation Operator 73
• Linear Non-Increasing Membership Function 73
• Triangular Membership Function 73
4.3.2 'Fuzzy and' ('$\widetilde{and}$') Aggregation Operator 74
• Linear Non-Increasing Membership Function 74
• Triangular Membership Function 75
4.3.3 'Min-bounded-sum' Aggregation Operator 75
• Linear Non-Increasing Membership Function 75
• Triangular Membership Function 76
4.3.4 Modification of the Traditional Crisp Model 76
4.4 Models Assessment 77
4.4.1 Performance of Models 79
4.4.2 Concluding Remarks 86
4.5 Summary 87

5 Heuristic Approaches for CF - Initial Solution **88**
5.1 Introduction 88
5.2 A Heuristic in the Presence of Part Machine Sequence 89
5.3 Stage I: Random Allocation of Machines to Cells 91
5.3.1 Design of the Algorithm 91
• Cell Machine Capacity 93
• Machine Instances Rearrangement 94
5.4 Stage II: Part Allocation to Machine-Cells 96
5.4.1 Initialisation Process 96
5.4.2 Machine Types Identification 99
5.4.3 Cell Sequence Identification 100
• Approach 1: Cell Sequence Relative to Part Machine Sequence 101
• Approach 2: Continuous Cell Sequence Relative to Part Machine Sequence 102
5.4.4 Part Allocation 106
• Algorithm Design 106
• Illustration of Part Allocation Strategy: Example (II) 110
5.5 Stage III: Evaluation of Objective Value and Solution 116
5.5.1 Total Number of Part/Cell Distinct Allocations 116
5.5.2 Total Part/Machine Set-up Cost 117
5.5.3 Calculation of Part Cell Later Revisits 119
5.6 Summary 120

6 An Iterative Heuristic Approach for CF 122

6.1 Introduction . . . 122

6.2 Heuristic Components . . . 122

6.3 Heuristic Algorithm Design . . . 123

6.3.1 Initialisation Process . . . 124

6.3.2 Iterative Procedure . . . 126

• Single Move . . . 127

• Interchange . . . 129

6.3.3 Ending Process . . . 131

6.4 Generation of Data Sets . . . 132

6.5 Computational Results for the Heuristic Approach . . . 134

6.5.1 Computational Results for Medium and Large Sized Problems . 135

• Heuristic Iterative Procedure Behaviour . . . 138

• Heuristic Approach vs. Initial Solution . . . 141

• Additional Trends in the Objective Values . . . 142

6.5.2 Computational Results for Small Sized Problems . . . 145

6.6 Concluding Remarks - Heuristic Algorithm Limitations . . . 148

6.7 Summary . . . 149

7 A Tabu Search Algorithm in the Presence of Part Machine Sequence150

7.1 Introduction . . . 150

7.2 Metaheuristics . . . 150

7.2.1 Tabu Search . . . 152

• Tabu Search Illustration . . . 154

7.3 Tabu Search and Short Term Memory . . . 156

7.3.1 Main TS Components . . . 157

• Definition of a Feasible Solution . . . 157

• Neighborhood Generation - Moves Definition . . . 158

• Short Term Memory: Aspiration Criterion . . . 159

• Short Term Memory: Tabu List & Tabu Tenure . . . 159

• Stopping Criterion . . . 164

7.4 Tabu Search Algorithm Design . . . 164

7.4.1 Initialisation Process . . . 164

7.4.2 Iterative Procedure . . . 165

7.4.3 Termination . . . 170

7.5 Summary . . . 171

8 Tabu Search Computational Results **172**

8.1 Introduction 172

8.2 Computational Results for the TS Algorithm 172

8.2.1 Computational Results for Medium & Large Sized Problems . . 173

• Tabu Search Iterative Procedure Behaviour 175

• Tabu Search vs. Initial Solution 177

• Tabu Search: Tabu Tenure Size 177

• Additional Objective Value Fluctuations 180

8.2.2 Computational Results for Small Sized Problems 182

8.3 Tabu Search Algorithm vs. Heuristic Approach 184

8.4 Summary 192

9 Conclusions and Future Work **193**

9.1 Conclusions 193

9.2 Suggestions for Future Work 199

Bibliography **201**

A CF Mathematical Models - `XPRESS-MP` **215**

A.1 Initial Model: Solution 216

A.2 Extended Model: Solution 216

A.3 Complete Model - `XPRESS-MP` Code 217

A.3.1 Complete Model: Solution 220

A.3.2 Illustration of Later Revisits of Parts 221

B Generic Forms of Selected Fuzzy Aggregation Operators **224**

B.1 'Min' Operator 224

B.2 'Min-bounded-sum' Operator 225

B.3 'Fuzzy and' Operator 226

C Random Data Generation **227**

D Part Allocation - `MatLab` Code **229**

E Tabu Search Algorithm - `MatLab` Code **235**

Synopsis

Cell formation has received much attention from academicians and practitioners because of its strategic importance to modern manufacturing practices. Existing research on cell formation problems using integer programming (IP) has achieved the target of solving problems that simultaneously optimise machine-cell allocation and part-machine allocation.

This thesis presents extensions of an IP model where part-machine assignment and cell formation are addressed simultaneously, and integration of inter-cell movements of parts and machine set-up costs within the objective function is taking place together with the inclusion of an ordered part machine operation sequence. The latter is identified as a neglected parameter for the Cell Formation problem.

Due to the nature of the mathematical IP modelling for Cell Formation two main drawbacks can be identified: (a) Cell Formation is considered to be a complex and difficult combinatorial optimisation problem or in other words NP-hard (Non-deterministic Polynomial time hard) problem and (b) because of the deterministic nature of mathematical programming the decision maker is required to specify precisely goals and constraints.

The thesis describes a comprehensive study of the cell formation problem where fuzzy set theory is employed for measuring uncertainty. Membership functions are used to express linguistically the uncertainty involved and aggregation operators are employed to transform the fuzzy models into mathematical programming models. The core of the research concentrates on the investigation and development of heuristic and metaheuristic approaches. A three stage randomly generated heuristic approach for producing an efficient initial solution for the CF together with an iterative heuristic are first developed. Numerous data sets are employed which prove their effectiveness. Moreover, an iterative tabu search algorithm is implemented where the initial solution fed in is the same as that used in the descent heuristic. The first iterative procedure and the tabu search algorithm are compared and the results produced show the superiority of the latter over the former in stability, computational times and clustering results.

List of Publications

(i). Papaioannou, G. and J.M. Wilson, *Extensions to Integer Mathematical Programming models of Cell Formation in Machine Scheduling*, Applied Mathematical Programming and Modelling (APMOD) Conference 2006, Madrid, Spain, June. 19-21, 2006

(ii). Papaioannou, G. and J.M. Wilson, *Fuzzy Extensions to Integer Mathematical Programming models of Cell Formation in Machine Scheduling*, Annals of Operations Research (Accepted)

(iii). Papaioannou, G. and J.M. Wilson, *A Tabu Search Algorithm for the Cell Formation Problem with Part Machine Sequencing*, International Journal of Production Research (Under Review)

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Professor John Wilson, for introducing me to the cell formation problem in the area of manufacturing and giving me the opportunity to work on such an interesting problem. His support, guidance and continuous encouragement during all years of my research study were invaluable.

I would also like to thank my panel committee members, Professor Jiyin Liu, my Director of Research, and Dr Mahmut Sonmez, the independent panel member, for their help and useful comments in the panel meetings and for their continuous encouragement throughout the three years of my research studies.

Many thanks to Dr Alan French for been helpful on a number of technical issues related to my research work.

I would also like to acknowledge Paul Day and Simon Mee, the IT staff, for their support.

Not to forget my dearest friends and colleagues, Xiaozheng, Keme, Arvind, Teresa, Rima, Sigrun, Ximo, Sofia and Vaggelis all coming from different cultures and research interests, for been very nice people and very supportive and also for me having the opportunity to share experiences with them via enlightening conversations along with the PhD work.

I should also mention that my graduate studies in Loughborough were supported by the Department of Business School together with the Faculty of Social Sciences and Humanities. This support is gratefully acknowledged.

I am very grateful to all my family for their patience, trust, support and more importantly their *love*.

Last but not least, a very big *thank you* to my husband, Argyris, for his love, strong encouragement, patience and support during the PhD studies.

Loughborough, England — Grammatoula Papaioannou

September 2007

List of Tables

1.1 Reported benefits from CM in [WH89] 8
1.2 Reported benefits from CM in [WJ97] 9

3.1 Part/Machine Utilisation 46
3.2 Machine Cell Allocation 1 47
3.3 *Desired* Part Machine Operation 47
3.4 Part/Machine Assignment 48
3.5 Part/Machine Set-up Costs 52
3.6 Machine Cell Allocation: Extended Model 52
3.7 Part/Machine Assignment for Extended Model 53
3.8 Example(I):Machine Sequence of Parts 59
3.9 Example(I): Part/Machine Utilisation 59
3.10 Example(I): Machine Cell Allocation 59
3.11 Example(I): Part/Machine Assignment 60

4.1 Classical LP Model 65
4.2 Fuzzy LP Model 65
4.3 Commonly Used Operators in FMP Studies 68
4.4 Numerical Values for DS 1 78
4.5 Computational Results for Fuzzy and Crisp Models 80
4.6 Results of varying γ Values for the 'fuzzy and' Operator 84
4.7 Results of varying γ Values for the 'min-bounded-sum' Operator 85

5.1 Output for $PCMATRIX(4,:,2,:)$ 98
5.2 Remaining Capacity Versus Current Utilisation 107

6.1 Size Sources for Medium and Large Data Sets 133
6.2 Size Sources for Small Data Sets 134
6.3 Problem Data and Computational Results for Medium & Large Sized Problems 137

6.4 Objective Function Value Evolution for Problem 19 140
6.5 Problem Data and Computational Results on Small Problems 146
6.6 Objective Function Value Evolution for Problem Instance Twenty 147

8.1 Problem Data and Computational Results for Medium & Large Problem Sizes 174
8.2 Problem Data and Computational Results for Small Problems 183

List of Figures

1.1 Jobbing Manufacturing 3
1.2 Mass Manufacturing 4
1.3 Layout Types in Production Systems 5
1.4 Cellular Manufacturing 6
1.5 An initial matrix which gives perfectly separable cells 10
1.6 Two perfectly separable cells 11
1.7 Non-disjoint cells due to an exceptional part 11
1.8 Non-disjoint cells due to a bottleneck machine 12

2.1 Illustration of McAuley's algorithm 23
2.2 Similarity matrix 23
2.3 Dendrogram produced from SLCA algorithm 24

3.1 Part Flow: No Part Machine Sequence Employed 49
3.2 Part Routing with Part Machine Sequence 50
3.3 Part/Machine Cell Allocation 2 54
3.4 Example(I): Part/Machine Cell Allocation 61

4.1 Membership Function for Objective Function 71
4.2 Membership Functions related to Constraints 72

5.1 Graphical illustration of a 3D matrix named *CELLMATRIX* 95
5.2 Graphical illustration of a 3D matrix named *PARTMATRIX* 97

6.1 Problem instance 19: Objective Function Value Evolution 139
6.2 Problem 19: Objective Function Value Evolution when Initial Solution Differs 142
6.3 Problem instance 18: Objective Function Value Evolution 143
6.4 Problem instance 6: Objective Function Value Evolution 144
6.5 Problem instance 20: Objective Function Value Evolution 148

7.1 Flowchart of a "simple" tabu search algorithm 154
7.2 TS model: Minimum Spanning Tree 155
7.3 Minimum Spanning Tree: Iterations 1 to 4 156
7.4 TS Neighborhood Generation 159
7.5 Single Transition 160
7.6 Interchange Transition 161
7.7 Tabu Decision Tree 162

8.1 Problem instance 14 176
8.2 Problem instance 14: Objective Function Value Evolution when Initial Solutions Differs 178
8.3 Problem instance 14: Objective Function Value Evolution when TT=15 179
8.4 Problem instance 12: Objective Function Value Evolution 181
8.5 Objective Value Evolution when $TT = 5$ 185
8.6 Objective Value Evolution when $TT = 10$ 186
8.7 Objective Value Evolution for Problem Instance 18 190
8.8 Tabu Search Algorithm vs. Heuristic Approach 191

Chapter 1

Group Technology & Cellular Manufacturing

1.1 Introduction

Nowadays, in the business environment, manufacturing companies are under intense pressure from the increasingly-competitive global marketplace. Shorter product life-cycles, time-to-market, and diverse customer needs have challenged manufacturers to improve the efficiency and productivity of their production activities. Manufacturing systems must be able to output products with low production costs and high quality as quickly as possible in order to deliver products to customers on time. In addition, the systems should be able to respond quickly to changes in product design and product demand without major investment. Because traditional manufacturing systems such as mass and jobbing production systems are not capable of satisfying such requirements, companies employ other innovative concepts. They put in a lot of effort trying to achieve these capabilities through: Just In Time (JIT), Quick Response (QR), Total Quality Management (TQM) and Group Technology (GT)/ Cellular Manufacturing (CM). Among all these concepts GT/CM is recognised by the manufacturing companies as having a main role in achieving world-class capabilities. Many large and medium-size companies have adopted GT/CM concepts and experienced reduction in manufacturing lead time, material handling cost and improvement in quality [Sin93].

Group Technology was first introduced in the former USSR by Mitrofanov [Mit66] and popularised in the west by Burbidge [Bur75]. GT can be defined as a manufacturing philosophy identifying similar parts and grouping them together to take advantage of their similarities in manufacturing and design.

In order to have an understanding of the need and application of GT in manufacturing, an introduction to the main traditional production systems and layouts is provided next.

1.2 Traditional Production Systems

Manufacturing production may be classified into three main types according to the variety and volume of products they produce: *Mass* production, *Batch* production and *Jobbing* production [Bur75].

Jobbing production systems are designed to achieve maximum flexibility such that a wide variety of products with small lots can be manufactured. Jobbing systems employ a *Functional Layout* which is designed in such a way that pieces of equipment with the same function are located together. Products usually require different operations and have different operation sequences. Operating times for each operation could vary significantly. Products are released to the shops in batches (jobs). The requirements of the system dictate the kind of machines that need to be employed and how they should be grouped and arranged. General purpose machines are utilized because they are capable of performing many different types of operations. Machines are functionally grouped according to the general type of manufacturing process: lathes in one department, drill press in another, and so forth. Figure 1.1 illustrates a jobbing production system.

In jobbing systems, jobs can spend 95% of their time in nonproductive activity; much of the time is spent waiting in a queue and the remaining 5% is split between lot set up and processing [AS93]. When the processing of a part in the system has been completed, it usually must be moved a relatively large distance to reach the next stage. It may have to travel the entire facility to complete all of the required processes as shown in Figure 1.1. Therefore, to make processing times more economical, parts are moved into batches. Each part in a batch must wait for the remaining parts in its batch to complete processing before it is moved to the next stage. This leads to longer production times, high production costs and low production rates.

In contrast to jobbing systems, mass production systems are designed to manufacture high volumes of products with high production rates and low costs. Mass production employs a *Line Layout* where for the manufacture of a part a number of sequenced operations are performed. Specialised machines are dedicated to the manufacture of the

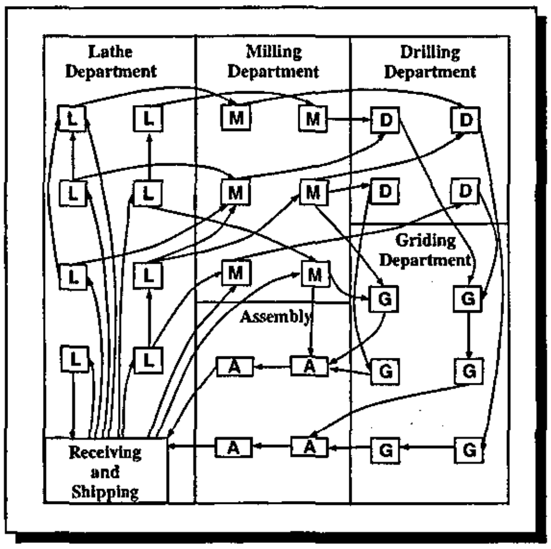


Figure 1.1: Jobbing Manufacturing

product and utilised to achieve high production rates. The advantages of flow layout are low work-in-process, high throughput time, low material handling cost and simple production scheduling. A major limitation of this layout is the lack of flexibility to produce products for which it is not designed. This is because specialised machines are set up to perform limited operations and are not allowed to be reconfigured. Also if one machine is broken that leads to the shutting down of the whole production line. Figure 1.2 illustrates mass manufacturing.

As indicated, jobbing and mass production systems cannot meet today's production requirements where manufacturing systems are often required to be reconfigured to respond to changes in product design and demand. For these kind of limitations Group Technology has emerged as a promising approach in manufacturing. Group Technology philosophy is one of the most important innovations to combine the benefits of both manufacturing strategies and produce a batch-oriented manufacturing system with a *Group Layout* [Bur75].

Group layout is one of the first key features in group technology. As original equipment manufacturers (OEM) are driven by the ever increasing market trends for short product life cycles and quick response to the changes in market condition, there are fewer and fewer products which enjoy the characters of large quantity and long product life time (this means dedicated product lines are often not financially justifiable). For this

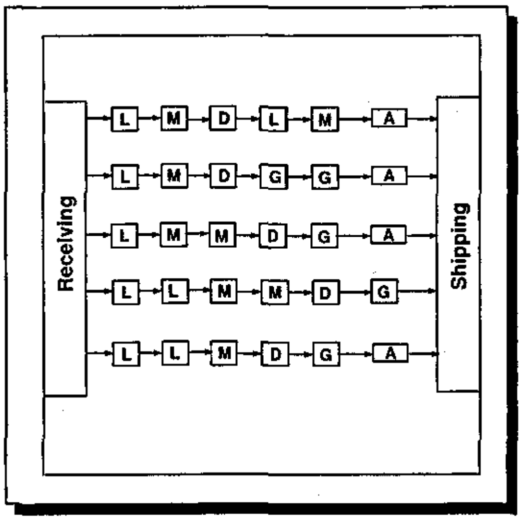


Figure 1.2: Mass Manufacturing

reason an increasing need for making the batch production more efficient is necessary. A good solution to this problem has been given through the group layout approach. Under this type of layout, the efficiency of line layout and the flexibility of functional layout are combined into one. The implementation of such a system at shop floor level is traditionally referred to as Cellular Manufacturing.

Before proceeding on CM and in order to illustrate the relationship among the three production layouts Figure 1.3 is provided. Line layout manufacturing systems provide more efficiency but lower flexibility whereas functional layout systems provide more flexibility but less efficiency. Group layout lies in between.

1.3 Cellular Manufacturing

1.3.1 Introduction

Cellular Manufacturing (CM) is an application of GT and has emerged as a promising alternative manufacturing system. CM could be characterised as a hybrid system linking the advantages of both the jobbing system (flexibility) and the mass production approach (efficient flow and high production rate). Within the manufacturing context, GT is identified as a manufacturing philosophy identifying similar parts and grouping them together into families to take advantage of their similarities in design and manufacturing [SAV98]. CM as the name suggests, entails the creation and operation of manufacturing cells. Parts are grouped into part families and machines into cells. A

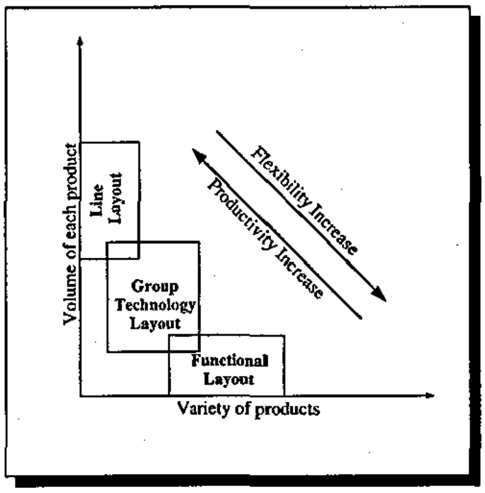


Figure 1.3: Layout Types in Production Systems

part family is a collection of parts which are similar either because of the geometric shape and size or similar processing steps are required in their manufacture. A manufacturing cell consists of several functional dissimilar machines which are dedicated to the manufacture of a part family.

CM is defined as the breaking up of a complex manufacturing facility into several groups of machines (cells) each being dedicated to process a part family. Under ideal conditions, each part type should be manufactured within a single cell.

As reported by Wemmerlov and Hyer the aim of CM [WH89] is comprised of three different scopes. First, reduce set-up times by using part-family tooling and sequencing. Secondly, reduce flow times by reducing set-up and move times, wait times for moves and using small transfer batches. Last but not least, there is the aim to reduce inventories and market response times. Moreover, in a survey by Wemmerlov and Johnson [WJ97], CM is promoted as the primary factor for the simplification of production planning and control procedures.

Practically the functionality of CM can be seen in Figure 1.4 where both jobbing and mass production systems are aggregated and converted into a cellular batch system. Obvious benefits gained from the conversion are less travel distance for parts, less space required and fewer machines needed. Since similar part types are grouped this could lead to a reduction of set-up and production time and allow a quicker response to changing conditions.

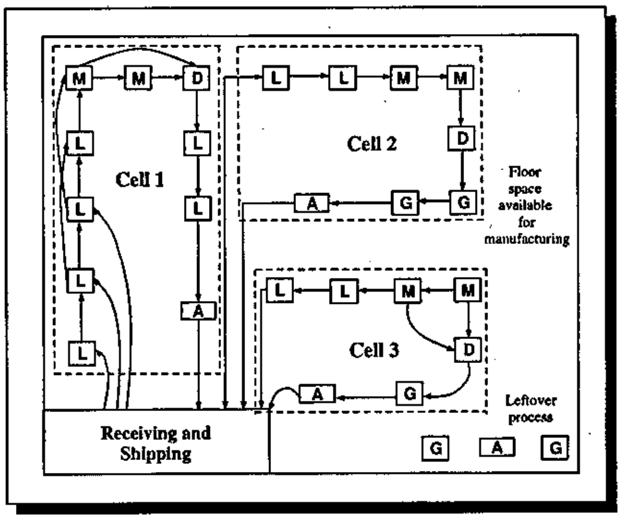


Figure 1.4: Cellular Manufacturing

1.3.2 Benefits of Cellular Manufacturing

The advantages of Cellular Manufacturing in comparison with the traditional production systems have been an important subject for many researchers for the past decades: [WH89], [HW89], [WJ97], [CS97], [Hye84], [Sin93], [Bur75], [LHZ95], [FWB87]. Analytical surveys, system implementations and simulation studies have been carried out in order to establish the benefits of cellular manufacturing. Some of the benefits are listed below:

- Reduced set-up time. Each cell is designed to contain part types belonging to the same part family. For this reason many of the parts can employ the same or similar holding fixtures. Therefore, when fixtures and tools for the parts need to be changed less time is required.
- Reduced throughput times. In CM, every part is transferred to the next machine immediately after it has been processed. For this reason the waiting time is reduced substantially.
- Reduced Work-In-Process (WIP) and finished goods inventory. Askin and Standridge [AS93] showed that the WIP can be reduced by 50% when the set-up time is cut in half. In addition to reduced set-up times and WIP, finished goods inventory is also reduced.

- Reduced material handling costs and time. In CM parts are produced within a single cell (when in ideal conditions). Thus part travel costs and time are minimal.
- Simplified flow of products. Reduced material set-up times and material handling time produce a simplification of flow products.
- Simplified scheduling. Since the manufacturing facility is broken down into cells and each part travels within a single cell, the schedule for the plant is simplified and it is easily controlled.
- Improved quality. Parts travel from one machine to another in small distances within the same cell till they have been produced. At certain stages immediate feedback is provided and if something goes wrong the process will be stopped and an inspection will follow.

A couple of reports have also been produced when real data were collected from firms that employed the cellular manufacturing approach. Wemmerlov and Hyer [WH89] reported the findings of a survey study of 32 U.S. firms involved in CM. These 32 firms represented a wide variety of product lines in machinery and machine tools, agricultural and construction equipment, hospital and medical equipment, defense products, engines, piece parts and components. Table 1.1 (page 8) shows the reported benefits.

Wemmerlov and Johnson [WJ97] conducted a similar survey in implementation experiences and performance measurements of CM at 46 user plants. In the survey, products manufactured in these 46 plants are electrical/electronic products and components, fluid handling and flow control devices, machinery and machine tools, heating, cooling products and components, tool engines and bearings. Table 1.2 (page 9) shows the reported improvements.

For introducing CM, it is necessary to identify parts and machines to be considered in the cellular configuration. This process differs with respect to whether cells are created by rearranging existing equipment in the shop floor or whether new equipment is acquired for the cells [SAV98], [WH86].

The problem of identifying machine cells and part families, has attracted considerable amounts of research interest. Burbidge [Bur63], [Bur75], proposed one of the earliest descriptive approaches for the CF problem which is referred to as Production Flow Analysis (PFA). PFA is a technique which analyses the information given in route cards

Table 1.1: Reported benefits from CM in [WH89]

Types of benefit	Number of responses	Average % Improvement	Minimum % Improvement	Maximum % Improvement
Reduction in throughput time	25	45.6	5.0	90.0
Reduction in WIP inventory	23	41.1	8.0	90.0
Reduction in material handling	26	39.3	10.0	83.0
Improvement of operator job satisfaction	16	34.4	15.0	50.0
Reduction in number of fixtures for cell parts	9	33.1	10.0	85.0
Reduction in set-up time	23	32.0	2.0	95.0
Reduction in space needed	9	31.0	1.0	85.0
Improvement of part quality	26	29.6	5.0	90.0
Reduction in finished good inventory	14	29.2	10.0	75.0
Reduction in labor cost	15	26.2	5.0	75.0
Increase in utilisation of equipment in the cells	6	23.3	10.0	40.0
Reduction in pieces of equipment required to manufacture cell parts	10	19.5	1.0	50.0

to form cells. A manual method for CF called "Nuclear Synthesis" was proposed where manufacturing cells are created around important *key machines*, used in conjunction with several other machines, to make many different parts. El-Essawy [EET72] proposed a method called Component Flow Analysis (CFA) at about the same time. In some respects, the methodology of CFA differs from that of Burbidge's PFA procedure since the latter first partitions the problem whereas the former does not.

1.4 Cellular Manufacturing Design

The design of cellular manufacturing systems has been called Cell Formation (CF) or Part Family/Machine Cell (PF/MC) formation or Manufacturing Cell Design (MCD). Given a set of part types, processing requirements, part type demand and available resources (machines, equipment, etc.), a general design of cellular manufacturing consists of the following approaches:

1. Part families are formed according to their processing requirements.

Table 1.2: Reported benefits from CM in [WJ97]

Performance	Number Measure	Average % Improvement	Minimum % Improvement	Maximum % Improvement
Reduction of move distance time	37	61.3	15.0	99.0
Reduction in throughput time	40	61.2	12.5	99.5
Reduction of response time orders	37	50.1	0.0	93.2
Reduction in WIP inventory	40	48.2	10.0	99.7
Reduction in set-up times	33	44.2	0.0	96.6
Reduction in finished goods inventory	38	39.3	0.0	100.0
Improvement of part quality	39	28.4	0.0	62.5
Reduction in unit costs	38	16.0	0.0	60.0

2. Machines are grouped into manufacturing cells.
3. Part families are assigned to cells.

Note that the above steps are not necessarily performed in the above order or even sequentially. Depending upon the procedures/formulations employed to form manufacturing cells and part families, three solution strategies are identified [SAV98]:

1. Part families are formed first and then machines are grouped into cells according to the part families. This solution strategy is referred to as Part Family Identification (PFI).
2. Manufacturing cells (grouped machines) are first created based on similarity in part routings and then the parts are allocated to cells. This solution strategy is referred to as Machine Groups Identification (MGI).
3. Part families and manufacturing cells are formed simultaneously. This is referred to as Part Families/Machine Grouping (PF/MG) solution strategy.

Numerous formulations exist for cellular manufacturing depending on the objective of optimisation at the level of manufacturing incorporated in the solution procedure. At the current stage, a simple formulation for the CF problem is described which is the machine/part matrix formulation. This formulation was first adopted by Burbidge as part of the Production Flow Analysis procedure for the implementation of CM system and also forms the basis of many procedures for cell formation in the years that followed.

1.4.1 Machine-Part Matrix Formulation

The Cell Formation problem in terms of matrix formulation can be defined as follows: given a machine-part incidence matrix showing which machines are required to produce each part, create cells of machines and identify families of parts, allocate part families to machine cells so that each part family is within a cell with minimum inter-cellular movement of parts. This CF definition could be illustrated with the help of a machine-component (m/c) matrix, $A_{m \times n}$, where:

- m is the total number of machines in the plant
- n is the total number of parts/components in the plant

Columns and rows of an incidence matrix represent parts and machines respectively. A matrix element a_{ip} is 1 if machine i is used to process part p, and 0 otherwise. The following example is an illustration for the binary matrix formulation: Consider a plant system with 5 parts and 4 machines. By analysing information carried on the route cards of the parts, the m/c matrix $A_{4\times5}$ in Figure 1.5 is obtained.

	p1	p2	p3	p4	p5
m1	0	1	0	1	1
m2	1	0	1	0	0
m3	0	1	0	1	0
m4	1	0	1	0	0

Figure 1.5: An initial matrix which gives perfectly separable cells

The value of $a_{3,4}$ entity is equal to 1, thus part 4 needs an operation on machine 3. In contrast part 4 does not need an operation on machine 2 since $a_{2,4}$ is equal to 0. A similar explanation applies to the rest of the matrix entries.

Once the m/c matrix has been obtained, the cell formation problem is now centralised on the transformation of the initial matrix into a solution matrix that has a block diagonal structure i.e. all positive entries are arranged inside blocks along the main diagonal of the m/c matrix. Rearranging all rows and columns results in the incidence matrix results in a solution diagonalised matrix shown in Figure 1.6.

By observing the solution matrix it is easy to identify two independent cells, the first one comprises of machines 2, 4 and parts 1, 3, whereas the second comprises of machines 1, 3 and parts 2, 4 and 5.

	p1	p3	p2	p4	p5
m2	1	1	0	0	0
m4	1	1	0	0	0
m1	0	0	1	1	1
m3	0	0	1	1	0

Figure 1.6: Two perfectly separable cells

The main objective of a cell-formation algorithm for the current version of the problem is the creation of completely independent cells. The last entails that given a number of components belonging to a specific part family these can only be processed within the cell initially assigned to. However, this is not what really happens in practice. Figure 1.7 illustrates a situation on a different matrix where the cells formed along the diagonal are not independent. More specifically, part 5 requires processing on machines 1, 3 and 4, therefore the matrix cannot be perfectly decomposed and no independent cells are formulated. Part 5 is called an *exceptional part.*

	p1	p2	p3	p4	p5
m1	1	1	0	0	1
m2	1	1	0	0	0
m3	0	0	1	1	1
m4	0	0	1	1	1

Figure 1.7: Non-disjoint cells due to an exceptional part

In a similar way to exceptional parts, *bottleneck machines* are defined. The matrix in Figure 1.8 cannot be perfectly decomposed because machine 5 processes parts belonging to more than one cell.

Exceptional parts and bottleneck machines are the sources of intercellular moves and significant consideration is given in cellular manufacturing research.

Furthermore, it is observed in Figure 1.8 a 0 in the bottom right cell. 0 represents a void which indicates that a machine assigned to a cell is not required for the processing of a particular part in a cell. In this example machine 5 is not necessary for part type 4.

	p1	p2	p3	p4	p5	p6
m1	1	1	0	0	0	0
m2	1	1	0	0	0	0
m3	0	0	1	0	1	1
m4	0	0	1	1	1	1
m5	1	1	1	0	1	1

Figure 1.8: Non-disjoint cells due to a bottleneck machine

The binary m/c matrix representation is a simple and efficient representation but captures only a limited amount of manufacturing data. It does not consider important information for the cell configuration. The more data types used the more advanced formulations (e.g. mathematical programming, graph-based methods) could be considered for the CF problem. These formulations and some more will be discussed in the literature survey.

1.4.2 Drawbacks in Current CM Design Methods

In the last three decades, over 200 research papers and practical reports have been published in the field of CM, seeking effective methods for designing Cellular Manufacturing Systems. CM design methods can be classified into clustering analysis, graph approaches, mathematical programming, heuristic strategies and fuzzy theory. A number of papers produced for each of these categories are described in Chapter 2.

Each design approach considers different numbers of design objectives and constraints, to different extent depending upon the scope and interest of each design approach. For instance clustering analysis approaches consider only one objective, i.e. the minimisation of intercell movements where only part operations and the machines for processing those operations are taken into account. Other product data (such as operational sequences and processing times) are not incorporated into the design process. Thus, solutions obtained may be valid in limited situations. However, they are simple to implement and solutions can be obtained in reasonable amounts of time.

Each design approach has its advantages and limitations. Some are simple to implement and to obtain solutions. Some capture the design more accurately by considering a number of objectives and constraints, but would require a substantial amount of time to obtain solutions. Among the available design approaches mathematical programming can capture the *reality* of the design better than others, since product data and production requirements can be incorporated. In general product data include processing times, set-up costs, tooling costs and other costs. Production requirements include product mix and demand in each period, available resources, machine cost, material handling cost. Most of these product data and requirements have been addressed before but none concentrated on the involvement of ordered part machine sequence within a CF system.

Moreover, a major drawback of mathematical programming approaches and especially those involving integer mathematical models, is computational time required for *large problems.* Obtaining optimal solutions from mathematical programming approaches can be infeasible due to the combinatorial complexity of the CM design methods [SAV98]. Thus, efficient heuristic solutions are needed to obtain reasonably good solutions for large scale problems in limited computational times producing a real system. As will be seen in Chapter 2, many heuristic approaches have been proposed in the last ten years, however, each is unique based each time on a particular model specifications.

Additionally, within mathematical programming modelling many system parameters are not easy to specify due to the deterministic nature of the former. For this reason fuzzy mathematical programming could be of application to model linguistic vagueness in information pertaining to design parameters and make the system more robust. The reasons that many CM parameters like machine capacity, processing time, machine duplication, intercell cost, could be uncertain is due to the following reasons:

- Substantial gap between design and implementation;
- High cost for acquiring these figures with precision.

1.5 Motivation for this Thesis

Cellular manufacturing has been a prosperous research area for the last three decades, however its design involved mainly classical methodologies (details can be found in Chapter 2). Relevant published material on cellular manufacturing design is primarily based on hard computing techniques. Only during the last decade some work has been carried out based on soft, breakthrough, computing techniques where additional criteria have been investigated for assessing the robustness of the cell formation system. The application of fuzzy logic within CF mathematical programming model where part families and groups of machines are formed simultaneously and the need for studying 'in-depth' the cell formation design from a mathematical and fuzzy point of view are of great importance. Moreover, many heuristic approaches have been developed in the past addressing the CF model with a number of constraints to be taken into consideration. However, in the author's knowledge none of the existing published material presents heuristic procedures where multiple machines of the same type are allocated to cells and parts are allocated to machine cells simultaneously when part machine operation sequence is taken into account. The latter forms a key constraint as CF is extended to incorporate more realism where data sets of significant size could be taken into account.

The motivation of this research is to undertake a rigorous study on fuzzy mathematical modelling analysis for the cell formation problem based on a sophisticated mathematical programming model. Cellular manufacturing systems have been mainly investigated in cases where no uncertainty is taken under consideration in the mathematical model. However, there is a need to investigate the systems behaviour with uncertainty taken properly into account as this is the situation in real applications. For better results reflecting real applications it is also necessary to develop heuristic algorithms via which all the benefits for the mathematical models will be revealed. This approach of solving the CF problem could provide feedback to the decision maker, implementer and analyst.

1.6 Work Addressed in this Thesis

The body of work described in this thesis takes the proven concept of cell formation and enhances it through the use of mixed integer mathematical programming. The principal area of investigation is that of cell formation when a specific type of uncertainty is taken into account and Fuzzy Set Theory is employed to measure it and make the system more robust. The core of the research involves the design of heuristic and metaheuristic algorithms in order to be able to incorporate realism and produce a practical CF system when large data sets are considered and a key constraint such as part machine operation sequence is taken into account. At present cell formation problems developed either as mathematical models or as heuristic strategies do not include the part machine ordered sequence and part/machine utilisation together with machines of multiple instances within their specifications.

Applying heuristic techniques and Fuzzy Set Theory, cell formation can be utilised by using more sophisticated model structures with larger scale data sizes, thus improving the concept of Cellular Manufacturing.

1.6.1 Thesis Structure

The thesis is laid out as follows:

- Chapter 2 is a detailed survey of work involving cell formation and its applications. It includes a number of different methodologies employed for CF over the past thirty years.
- Chapter 3 presents a mixed integer mathematical programming model for the cell formation problem for simultaneously grouping machines into cells and assigning parts to machine cells when a number of key constraints and an enhanced objective function are taken into account.
- Chapter 4 introduces the use of fuzzy set theory where fuzzy aggregation operators and membership functions are employed to measure the uncertainty involved for determining the maximum number of machines allowed in each cell within the cell formation system.
- Chapter 5 presents a three stage heuristic approach for designing and setting up an initial starting framework representing the CF problem.
- Chapter 6 proposes an iterative heuristics approach, for the cell formation mathematical programming model in order to be able to use large scale data sets and incorporate more realism. Computational results for small, medium and large data sets are also presented and the heuristic's behaviour is examined together with some of its limitations.
- Chapter 7 proposes an extension to the heuristic approach based on the framework of a metaheuristics strategy and more specifically on the principles of the tabu search where neighboring solutions are investigated and certain moves are forbidden as the search goes along.
- Chapter 8 presents the computational results for the tabu search algorithm when a number of problem instances small, medium and large are taken into account. The algorithm's performance is explored by examining its behaviour on a number of issues and certain conclusions are drawn when comparing the latter with the heuristic approach;
- Chapter 9 contains a conclusion and discussion on the overall thesis results, and suggests possible further work.

- The following information is included in the Appendix: (A) CF Mathematical Models - `XPRESS-MP`, (B) Generic Forms of Selected Fuzzy Aggregation Operators, (C) Random Data Generation, (D) Part Allocation - `MatLab` Code (E) Tabu Search Algorithm - `MatLab` Code.

1.7 Published Work

(i). Papaioannou, G. and J. M. Wilson, *Extensions to Integer Mathematical Programming models of Cell Formation in Machine Scheduling*, Applied Mathematical Programming and Modelling (APMOD) Conference 2006, Madrid, Spain, June. 19-21, 2006

(ii). Papaioannou, G. and J.M. Wilson, *Fuzzy Extensions to Integer Mathematical Programming models of Cell Formation in Machine Scheduling*, Annals of Operations Research (Accepted)

(iii). Papaioannou, G. and J.M. Wilson, *A Tabu Search Algorithm for the Cell Formation Problem with Part Machine Sequencing*, International Journal of Production Research (Under Review)

1.8 Thesis Contributions

This study addresses a number of issues concerning the problem of Cell Formation and makes contributions in the following areas:

(a). Rigorous investigation of the Cell Formation problem;

(b). Development of an advanced mixed integer mathematical programming model (MIMP) to simultaneously optimise cell formation, machine-cell allocation and part machine cell allocation when part machine sequencing is taken into account;

(c). Theoretical investigation of a type of uncertainty involved within the MIMP model and the employment of Fuzzy Theory for further experimentation. Assessment of Fuzzy Theory via membership functions and aggregation operators on a number of data sets;

(d). The novel idea of developing heuristic and metaheuristic algorithms for the Cell Formation problem when part machine operation sequence is taken into account incorporating realism;

(e). Assessment of the heuristic and metaheuristic strategies using small, medium and large data sets, where their performance is examined based on a number of criteria and a comparison among algorithms is made on the basis of producing a robust and stable cell formation system useful for production planners.

Chapter 2

Literature Survey

In the last three decades much work has been undertaken seeking effective methods for designing cellular manufacturing systems. A considerable number of papers have been published during this time making the task of surveying and classifying the solution approaches both difficult and challenging.

The purpose of this review is not to provide an extensive coverage of all the literature available but to highlight the relevant studies to this research. In this chapter a number of important solution methodologies for cellular manufacturing will be examined.

2.1 Introduction

There is no default way of classifying cell-formation methods. In CM literature a number of reviews have been written some of which could be found in [SAV98], [AS98], [JKC95], [Sin93], [GS84]. A first attempt to classify the approaches, results in the following three categories:

- Informal methods
- Part coding analysis methods
- Production based methods

Informal methods or visual methods or simply "eye-balling" methods rely on the visual identification of the correspondent part families and machine cells. This methodology is trivial only when the number of parts and machines is small or could be longer but with considerable flows. Otherwise, the identification task becomes impossible.

In part coding analysis (PCA) methodologies the design characteristic of the parts has an important role in the formation of part families. These methodologies use a coding system to assign numerical weights to part characteristics and identify families using some classification scheme. PCA-based systems are traditionally design oriented or shape-based, therefore they are ideal for component variety reduction.

The core classification for the purpose of this research is production based methods. These methods will be defined indirectly through a further classification consisting of the following categories:

- Clustering
- Graph partitioning approaches
- Mathematical programming methods
- Heuristics
- Metaheuristics
- Fuzzy logic

The following section presents for each of the categories, methodologies/algorithms proposed in a considerable number of papers.

2.2 Cell Formation - Existent Methodologies

2.2.1 Clustering

Cluster Analysis is composed of many diverse techniques for recognizing structure in a complex data set. The main objective of this cell formation tool is to group either objects or entities or attributes into clusters such that individual elements within a cluster have a high degree of "natural" association among themselves and a very little "natural association" between clusters. Clustering procedures can be classified as:

- Array based clustering techniques
- Hierarchical clustering techniques
- Non-hierarchical clustering techniques

• **Array-Based Clustering**

In array based clustering the processing requirements of components on machines can be represented by the part/machine matrix formulation which was described in Chapter 1. The array based techniques try to rearrange columns and rows of the matrix till blocks of entries equal to 1 are produced along the diagonal.

The literature yields at least seven array-based clustering algorithms namely: Rank Order Clustering, Rank Order Clustering 2, Bond Energy algorithm, Modified Rank Order Clustering, Direct Clustering Analysis, Clustering Identification algorithm, Cost Analysis algorithm and Close Neighbour algorithm.

Rank Order Clustering (ROC) was devised by King [Kin80] and was designed to generate diagonalised groupings of the matrix entries. ROC was based on reading each row of the cell entries as a binary word, represented by '0' or '1', and then ranking them in decreasing order of their ranking. The same procedure was repeated, only this time on columns. The process was iterative and continued until no further change could be achieved. ROC algorithm has its roots in the Bond Energy Algorithm (BEA) [MSW72], a general clustering algorithm applied to a wide range of clustering problems. It has been computationally proved that ROC algorithm is more efficient in terms of computer time than the bond energy method when applied to the machine/component matrix problem.

King and Nakornchai [KN82] introduced a modified version of ROC called ROC2. ROC2 utilises linked lists to overcome some of the limitations of ROC. With the ROC algorithm the storage of the incidence matrix as a two dimensional array puts a severe limit on the size of the problem that can be tackled. Moreover, because the sorting procedure has a complexity of cubic order, efficient implementation is not possible for large problems. By applying the ROC2 algorithm, linked lists enabled the use of fast and efficient sorting procedures, which resulted in an overall algorithm with linear time complexity. The ROC2 algorithm was also combined with other specialised procedures to deal with exceptional parts and bottleneck machines.

Chandrasekharan and Ragagopalan [CR86b] argued that the ROC algorithm tended to collect ones in the top-left hand corner of the machine/component matrix while the rest of the matrix was left highly disorganised. This tendency resulted in the erroneous identification of bottleneck machines. To overcome these limitations Chandrasekharan and Ragagopalan introduced a modified ROC called Modified Rank Order Clustering

(MODROC) algorithm, which removes the ROC defects to a great extent and enables the identification of bottleneck machines. MODROC started with the execution of two iterations of the ROC algorithm. Then, the block of machines and components created on the top left hand corner were removed from the matrix. This process was repeated until all components families were identified. From this algorithm mutually exclusive part families formed but the machines cells were not necessarily non-intersecting. For this reason, hierarchical procedures were applied and the final plant configuration was produced.

Direct Clustering Analysis (DCA) method, introduced by Chan and Milner [CM82], was an efficient clustering technique for producing a diagonalised matrix. This method employed a systematic way for the manipulation of rows and columns of the matrix. DCA consisted of going through the matrix sequentially, moving the rows with 'left-most' positive cells (those with an 'X' value in the m/c matrix) to the top and the columns with the 'top-most' positive cells to the left of the matrix. The basic rule was that each component or machine number had to be moved together with its respective row or column entries during matrix transformation as if the cells or the blocks were linked together by an imaginary rod. The procedure was iterative and continued until no further improvement could be achieved. The main advantage of this method over ROC was that the initial configuration of the matrix did not affect the final partition. This was achieved as in a pre-processing stage columns and rows were ranked in order of decreasing and increasing value of positive cell entries respectively.

Kusiak and Chow [KC87] developed two efficient algorithms for the CF problem: the Clustering Identification Algorithm (CIA) and the Cost Analysis Algorithm (CAA). CIA was developed and applied to a problem with a standard formulation, whereas CAA applied to an augmented structured problem. The standard formulation was based on the 0-1 m/c matrix and did not consider any costs. In the augmented formulation any part j had an associated cost c_j assigned to it and the number of machines in a cell were of limited size. CIA was based on the cutting algorithm, originally proposed by Iri [Iri68] and was able to define machine cells and part families by drawing vertical and horizontal lines on the m/c matrix. CAA was an extension of CIA and explicitly considered the cost of subcontracting parts that caused intercellular moves.

Boe and Cheng [BC91] produced an efficient heuristic method called Close Neighbour Algorithm (CNA) for the grouping of machines and components in a binary m/c matrix. This algorithm was comprised of two basic steps. First, it rearranged the machines and components by examining a measure of similarity for each pair of machines. This level of similarity was calculated from the 'closeness' of the machines in their part routings. The heuristic procedure applied resulted in the diagonalisation of the intermediate matrix. The algorithm was tested against ten existing algorithms in solving test problems from the literature. Test results showed that the algorithm was reliable and efficient.

In later work, Da Silveira [DS99] proposed a methodology for implementation of cellular manufacturing. This procedure was designed and refined along an action research project in a toy manufacturer plant in Brazil. The benefits of the implementation in terms of reduction in scrap, rework, work-in-process, final stock, batch size and delivery times were significant. The method employed for the identification of part families and machine cells was Boe and Cheng's [BC91] Close Neighbour Algorithm.

- **Hierarchical Clustering**

Hierarchical clustering for CF is mainly comprised of two stages. Initially, some form of similarity or dissimilarity between machines or parts is employed, in order to create machine cells or part families. Later, machines or parts are separated into a few broad cells, each of which is further divided into smaller groups and each of these further partitioned and so on until terminal groups are generated which cannot be subdivided. Essentially hierarchical techniques can be classified into two methods:

- *Divisive methods.* Start with all the data (machines or parts) in a single group and create a series of partitions until each machine (part) is in a singleton cluster.
- *Agglomerative methods.* Start with singleton clusters and proceed to merge them into larger partitions until a partition containing the whole set is obtained.

Hierarchical classifications may be represented by inverted dendrograms, which are two-dimensional diagrams illustrating the fusions or divisions which have been made at each stage of the analysis.

The first author who introduced agglomerative methods hierarchical clustering for the CF problem was McAuley [McA72]. Because his methodology forms the basis for the succeeded hierarchical algorithms developed for the CF problem, it will be described in more detail with the help of the example matrix given in Figure 2.1.

Initially, a similarity coefficient value (ranged between 0 and 1) is calculated for each pair of machines in the plant. The similarity coefficient indicates the similarity of the

	p1	p2	p3	p4	p5
m1	1	0	1	0	0
m2	0	1	0	1	1
m3	1	0	1	0	0
m4	1	1	0	1	0

Figure 2.1: Illustration of McAuley's algorithm

machine in terms of operations performed. McAuley employed the Jaccard's [Jac08] coefficient which for the CF problem is defined as follows:

$$S_{ij} = \frac{a_{ij}}{a_{ij} + b_{ij} + c_{ij}} \tag{2.1}$$

where, S_{ij} is the similarity between machines i and j, a_{ij} number of parts processed by both machines i and j, b_{ij} number of parts processed by machine i but not machine j, and c_{ij} number of parts processed by machine j and not machine i.

The similarity coefficient value for each of the pairs in matrix (Figure 2.1) is as follows:

$$S_{1,2} = \frac{0}{0+2+3} = 0.0 \qquad S_{1,3} = \frac{2}{2+0+0} = 1.0 \qquad S_{1,4} = \frac{1}{1+1+2} = 0.25$$
$$S_{2,3} = \frac{0}{0+3+2} = 0.0 \qquad S_{2,4} = \frac{2}{2+1+1} = 0.5 \qquad S_{3,4} = \frac{1}{1+1+2} = 0.25$$

Between machines 1 and 3 there is a total similarity, and no similarity between machine pairs 1, 2 and 2, 3. From these values the following similarity matrix in Figure 2.2 is produced.

	m1	m2	m3
m2	0.0	*	*
m3	1.0	0.0	*
m4	0.25	0.5	0.25

Figure 2.2: Similarity matrix

In the second stage and after the similarity between pairs of machines has been established, the numerical method for finding and defining clusters of mutually high similarity coefficients and presenting them in a dendrogram, was termed Cluster Analysis. The algorithm employed for the construction of the dendrogram is called Single

Linkage Cluster Analysis (SLCA) [Sne57]. This method clusters those machines mutually related with the highest possible similarity coefficient. In the previous example, machines 1 and 3 are grouped at the similarity level 1. Then the algorithm successfully lowers the level of admission by steps of predetermined equal magnitude. The next highest similarity level is found and the associated pair of machines is merged at this level. In this case machines 2 and 4 are merged at the similarity level 0.5. Later, and since machines 1 and 4 have already been grouped their groups are merged together as well at the next highest similarity level of value 0.25. A picture of how the algorithm can be represented in the dendrogram is illustrated in Figure 2.3.

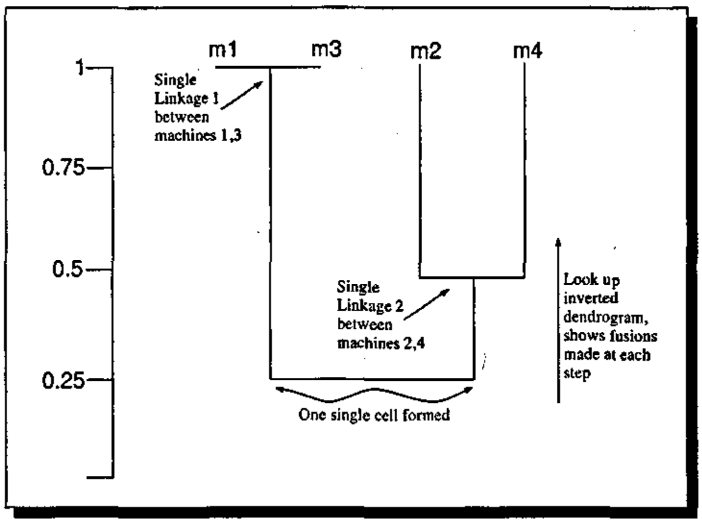

1
0.75
0.5
0.25
m1
m3
m2
m4
Single Linkage 1 between machines 1,3
Single Linkage 2 between machines 2,4
One single cell formed
Look up inverted dendrogram, shows fusions made at each step

Figure 2.3: Dendrogram produced from SLCA algorithm

The main disadvantage of this method is that while two clusters may be linked by this technique on the basis of a single bond, many of the members of the two clusters may be quite far removed from each other in terms of similarity. For instance, machines 1 and 2 were merged together at a similarity level of 0.25 although their calculated similarity value was 0.0.

In order to overcome this inefficiency of SLCA, McAuley and other researchers [GS90] suggested the use of alternative methods like Average Linkage Cluster Analysis (ALCA) and Complete Linkage Cluster Analysis (CLCA). ALCA calculated the average of similarity coefficients between groups of machines. The CLCA worked in a reverse way of SLCA by assigning the lowest and not the highest similarity coefficient between groups of machines. However, both ALCA and CLCA required the recalculation of the similarity matrix after each individual step resulting in greater computational complexity.

Gupta and Seifoddini [GS90] extended the applicability of the coefficient-based hierarchical clustering methods by introducing an enhanced version of McAuley's similarity coefficient. The researchers argued that the main disadvantage of McAuley's method was the limited amount on manufacturing information. Thus, they introduced the production-based similarity coefficient method which included relevant production data, such as part type production volume, routing sequence and processing times. The superiority of the production-based similarity coefficient over McAuley's similarity coefficient was verified on some test problems taken from the literature.

Gupta [Gup93] introduced an improved version of similarity coefficient that incorporated alternative process plans for the parts produced. Seifoddini and Djassemi [SD95] compared the performance of the production-based similarity coefficient and the coefficient-based similarity coefficient and found that the former was more able to effectively reduce material handling cost.

Vakharia and Wemmerlov [VW90] proposed a methodology for the cell formation problem which this time was based on the identification of part families rather than machine cells. The similarity measure proposed, considered not only the number of machines visited by each part but also the operation sequences. The merging procedure of the parts was highly interactive with the operator having the power to control the system by approving or objecting to merging based on some information. This information was supplied from resultant skip and backtracking moves. Skip moves were generated when a part required processing on one or two machines. This part in that case was identified and removed from the total population of parts. On the other hand, when a part required processing on a machine type more than once then backtracking was necessary. After the grouping of the parts in part families was achieved, the operator dealt with backtracking and single operation machine parts that had been removed from the system initially, and decided where to allocate 'key' equipment (equipment required by many parts). The main disadvantage of this highly interactive methodology was that it required an expert operator in this field to make specific decisions.

Stanfel [Sta85] introduced a clustering algorithm which was not based on agglomerative techniques but on divisive solutions. Divisive solutions were generated by progressively breaking down a single cell or part family to individual machines or parts. In other words, Stanfel's algorithm started with a single, parent cell including all machines and did not make use of a similarity coefficient between machines in order to form machine cells like the previous described algorithms. An iterative procedure followed with each machine selected to leave the parent cell to either form a new cell or to join an already formed cell. All evaluations made at each iterative step, based upon both the resulting intercell moves and the number of extraneous transitions caused by the presence of machines within a cell, were not processing all the family parts.

• Non-Hierarchical Clustering

Non-hierarchical clustering methods are iterative methods but they also employ a measure of similarity or dissimilarity for grouping parts or machines. They begin with either an initial partition of the data set or the choice of a few seed points. In either case, one has to decide the number of clusters in advance. Non-hierarchical procedures have been developed by Chandrasekharan and Rajagopalan [CR86a],[CR87], Srinivasan and Narendran [SN91], and Jayakrishnan Nair and Narendran [JNN98], [JNN99].

Chandrasekharan and Rajagopalan [CR86a] introduced the first non-hierarchical clustering algorithm for the CF problem which was comprised of three stages. Initially, a method employed for the clustering of parts and machines which was called the *K-means method* developed by MacQueen [Mac67]. The K-means method was reported to have many advantages [And73] however, it required the number of clusters to be specified in advance. Therefore, a modified version of the K-means method was proposed and a new formula was derived for the calculation of the maximum number of independent cells that could be formed for a specific problem. For the second stage of the algorithm part families were allocated to machine cells and a diagonalised matrix was produced. The last was achieved with the calculation of an *efficiency factor*, an indicator of maximising the within cell utilisation for each part family and of minimising intercell movements. A further improvement in terms of both utilisation and intercell movement was achieved with the introduction of ideal-seed points in the third stage. These seed points initialised a new run of the algorithm having as a result the elimination of singleton clusters. Within the same paper a new concept that of *grouping efficiency* was developed to provide a quantitative standard on a national scale for comparing different solutions to the same problem. Grouping efficiency was utilised by many researchers in the area of cellular manufacturing.

An extension and improved version of the ideal seed clustering algorithm was introduced by the same authors [CR87] and called ZODIAC. The acronym ZODIAC stands for Zero One Data: Ideal seed Algorithm for Clustering. This algorithm dealt with concurrent formation of part families and machine cells and identified them in a block diagonalisation of the zero-one matrix. ZODIAC and the ideal seed clustering algorithm were much the same except that with ZODIAC different methods of choosing seeds had been developed and tested.

Srinivasan and Narendran [SN91] identified that ZODIAC suffered from two inefficiencies. The choice of the initial number of seeds needed careful consideration and the use of the city block distance did not reflect the extent of processing required by components. For these reasons the same authors introduced a new non-hierarchical algorithm called GRAPHICS (GRouping using Assignment method For Initial Cluster Seeds). GRAPHICS identified initial machine cells by solving an assignment problem [SNM90] which assigned machines to form cells such that similar machines were grouped together and the similarity between machines was maximised. The machine groups obtained, formed an ideal starting point for the algorithm, since a set of machine cells provided the seeds to cluster the parts and the given set of part-families in sequence was used to generate seeds to cluster machines. The main algorithm processed with the alternative clustering of machines and parts until no improvement could be made in terms of the number of exceptional elements and the voids. GRAPHICS was compared with ZODIAC and it was found to fare better in terms of grouping efficiency and computational time, particularly for ill structured matrices.

An extended version of the GRAPHICS algorithm was proposed by Srinivasan [Sri94]. A minimum spanning tree (MST) for machines was constructed from which seeds to cluster components were generated. Machine cells and component families were not formed simultaneously. This algorithm was compared to GRAPHICS and ZODIAC and in both cases it was observed that it had a better performance. For the comparison of the algorithms a number of examples were taken from the literature.

Jayakrishnan Nair and Narendran [JNN98] introduced a non-hierarchical clustering algorithm called CASE (Clustering Algorithm for Sequence Data). Machine cells and component families were identified on the basis of production-sequence data and for this reason they presented a new similarity coefficient which considered the sequence of operations for each part, multiple visits to machines and part demands. The coefficient was used for the identification of the initial seeds for the start of the clustering. Unlike previous methods described above, CASE does not demand a priori specification of the maximum number of machines in a cell or the number of cells. It allows natural clusters to emerge and yields solutions of higher quality.

Later, Jayakrishnan Nair and Narendran [JNN99] presented an enhanced version of CASE called ACCORD (A bicriterion Clustering algorithm for Cell-formation using Ordinal and Ratio-level Data). This algorithm was developed because of the need to utilize data on production sequence like volumes, processing times and machine capacities. Taking the sequence directly as the ordinal data and combining the rest into ratio-level data, ACCORD had a twin objective of minimizing within-cell load as well as intercell moves. ACCORD combined the similarity coefficient used in CASE with a new similarity coefficient that captured the workload similarity between any pairs of machines in the system.

2.2.2 Graph Partitioning Approaches

Graph partitioning approaches employ a graph or network representation for the CF problem, where machines and/or parts are treated as vertices and the processing of parts as edges connecting these nodes.

One of the first graph approaches for the CF problem was introduced by Ragagopalan and Batra [RB75]. This approach used Jaccards's [Jac08] similarity coefficient and graph theory to form machine groups. Each vertex in the graph represented a machine and any pair of vertices was connected by an edge if and only if the 'similarity' between the machines was greater than a pre-specified threshold value. After all the allowable edges were introduced, cliques (complete maximal subgraphs) were formed and then merged to create hybrid cells. At this stage, many cells were observed to have a number of the same machines and thus a new procedure was needed for the creation of mutually independent cells. A new graph was created with each vertex representing a cell (cliques of the machine-graph assumed to be the starting cells) and each edge representing intercellular moves between the hybrid cells. In this work the Kernighan-Lin [KL70] heuristic procedure was utilised for partitioning the corresponding graph. The objective of the algorithm was the minimisation of the total number of intercellular moves.

De Witte [DW80] combined the hierarchical clustering algorithm of McAuley [McA72] and the graph partitioning procedure by Ragagopalan and Batra [RB75] for solving the following problem: 'Group components into families and machines into cells, in such a way that each component can be fully processed in a cell using existing plant, tooling and processing methods'. In an attempt to analyse the existing machines De Witte distinguished them into three main categories:

- Primary machines: Machines types of which only one unit was available, which could be allocated to one cell only.
- Secondary machines: Machine types of which multiple units were available, which could be allocated to several cells.
- Tertiary machines: Machine types of which enough units were available to cover every cell in the plant.

A machine-to-machine combination matrix (indicating relations between the machine types) was then created based on the components operations routing sequence and machining times, and their required quantity. In order to analyze the relations between machine types three different similarity coefficients were calculated. These values were then used as input to Ragagopalan and Batra's graph partitioning procedure for the creation of machine cells. Primary, secondary and tertiary cells were created sequentially. Finally, secondary and tertiary cells were added to the primary cells to obtain the final design of the plant.

Vannelli and Kumar [VK86] focused on finding bottleneck parts or machines when creating manufacturing cells. It was shown that this problem was equivalent to finding the minimal cut-nodes of a graph while disconnecting the graph into a number of subgraphs. Since the problem was NP-complete a heuristic dynamic programming approach [LVM79] was employed for its solution.

Later, Vannelli and Kumar [VK87] presented a new methodology for using a subcontracting strategy to induce manufacturing efficiency by re-organizing the existent parts and machines into disaggregated cells. Two efficient algorithms were developed which identified the minimal number or minimal total cost of subcontractible parts while achieving disaggregation. For the development of the second mentioned algorithm the concept of the weighted graph was introduced.

Askin *et al.* [ACGV91] proposed a new method called the Hamiltonian Path Heuristic (HPH) for identifying machine cells and part families simultaneously. Machines and parts in the m/c incidence matrix were ordered using a 'distance' measure. For the calculation of the distance a modified version of Jaccard's similarity coefficient was employed and the distance matrix was formulated. For rearranging the rows and the columns of the matrix the graph-based Travelling Salesman Problem (TSP) was first considered with the objective of finding the shortest tour of all vertices. However, because TSP required a cyclic solution, the associated Hamiltonian Path Problem (HPP) was later employed since it did not require a return tour to the starting vertex. For both problems, graph heuristic procedures were utilised.

Ng [Ng93] introduced a Minimum Spanning Tree (MST) methodology for the solution of the binary CF problem. Each row of the incidence matrix was treated as a node and there existed an arc between every pair of nodes. Arcs denoted distances between nodes or in other words represented the level of nodes dissimilarity. K machine cells were obtained by deleting the $(k-1)$ largest arcs from the tree. A methodology was also presented for reassigning parts to machines aiming to improve the derived partitions. Also a worst-case analysis of the algorithm was performed in terms of the grouping efficiency and grouping efficacy [KC90] measures. Finally, deficiencies of both measures were illustrated and a new measure called *weighted grouping efficacy* was proposed.

MSTs were also employed by Lin *et al.* [LDKN96] in an attempt to solve a more enhanced version of the CF problem. For the initial modelling of the system, mathematical programming was used. A multi objective function was defined as: minimize the sum of intercell processing costs, intracell processing costs and total cell balance delay costs. This model formulation is a non-linear integer program that is generally difficult to solve [Kus90]. For this reason a MST heuristic was employed for its solution. Cells were created by continuously deleting arcs from the graph till no further configuration could be found that resulted in lower overall costs. This model compared with existing array-based methods on examples taken from the literature and produced excellent results.

2.2.3 Mathematical Programming Methods

Mathematical Programming formulations can be used in a number of circumstances involving a wide range of manufacturing data. Several types of integer programming formulations have been proposed over the past years. These formulations simultaneously assign parts to individual machines and group the individual machines into cells. However, most of the these proposed models require the prior specification of the total number of manufacturing cells.

Purcheck [Pur75], [Pur85], and Olivia-Lopez and Purcheck [OLP79] were among the first researchers to apply linear programming techniques to the group technology problem. They essentially applied the technique of combinatorial grouping and LP to the

CF problem.

Kusiak [Kus87] developed a *p-median* zero-one integer programming problem for the formation of part families. The objective of the problem was the maximization of the total sum of similarities of parts within the part families in terms of the common machines used. For the solution of this integer programming problem the software package LINDO was employed. Kusiak also proposed a generalised group technology concept, generalised *p-median* formulation, based on generation of a number of different process plans for each of the parts.

Kusiak and Heragu [KH87] proposed a quadratic formulation for clustering machines and parts in an m/c. However, since a quadratic formulation is computationally complex they applied a *p-median* formulation and also a generalised *p-median* formulation. For the latter the m/c incidence matrix was extended to obtain a generalised matrix in which for every part there was more than one column corresponding to a different process plan. They concluded that the generalised *p-median* formulation was the most efficient for solving the clustering problem in GT.

Shtub [Sht89] proved that a simple CF problem, and a CF problem where the generalized group technology concept is employed [Kus87], are equivalent to the Generalised Assignment Problem(GAP).

Choobineh [Cho88] proposed a two stage procedure for the design of a cellular manufacturing system. For the first stage where part families were identified, a hierarchical clustering algorithm was employed. The similarity between parts was calculated through an enhanced version of Jaccard's similarity coefficient. Through this coefficient other process plans for each of the part types were also addressed. The technical goals of stage two were to find the number of cells, assign part families to cells and assign machines to cells. For the achievement of these goals a linear integer programming formulation was presented. The objective of the model was the minimisation of production costs and the costs of acquiring and maintaining machine tools.

Wei and Gaither [WG90] developed a zero-one integer programming formulation for determining which machines and parts should be assigned to cells. The objective of the model was the minimisation of the opportunity cost of manufacturing exceptional elements outside the cellular systems (either produce them in a job shop remainder cell or subcontract their production to a supplier), subject to machine capacity constraints.

The authors also considered other objectives like minimisation of intercell capacity imbalance, minimisation of cost, distance, or weighted distance of intercell shipments. These objectives were also used in conjunction with the proposed mathematical programming formulation.

Kasilingham and Bhole [KB90] developed a zero-one integer programming model to form machine-part cells and to decide on the number of machines and the number of copies of tools required to achieve minimum overall system cost. The overall system cost was defined as the sum of the annual cost of processing the parts, cost of tooling and annualised machine investment costs.

Sankaran [San90] considered multiple goals for the cell formation procedure by developing a model with a single objective function defined by five distinct cost functions. The optimal solution of the single objective model was broken down into two aspiration levels: operating cost and capital investment cost. These two costs along with five other goals were then combined in an integer linear *goal* programming model. The set of goals considered by the authors included: minimum similarity of parts based on their needed machines and tools (two goals), available machining capacity, minimum and maximum number of total parts movement (two goals), the optimal capital investment on machines and the optimal operating cost. The main disadvantage of this methodology was that the decision maker had to specify in advance priority weights for each of the goals. This made the problem solving more difficult.

Boctor [Boc91] proposed a zero-one formulation and considered only the data that were available from the machine part binary matrix. The objective of the model was the minimisation of the total number of the exceptional elements. An efficient procedure for the linearisation of the objective function was also developed. Boctor showed that some of the large number of integrality constraints in the problem could be relaxed without changing the binary outcome of the model. However, for large scale problems even with the above modifications the problem was still computationally intractable. The use of a Simulated Annealing heuristic algorithm, initially developed by Metropolis *et al.* [MRR+53], was proposed for these cases.

Zhu *et al.* [ZHR95] developed a zero-one integer programming formulation and compared it with the formulation that Wei and Gaither [WG90] had proposed. Zhu *et al.* proposed as objective the maximization of opportunity costs of the total number of parts to be produced within the system, whereas Wei and Gaither tried to minimise the opportunity costs for those parts that were to be manufactured outside the system. These two objectives were the same in the sense that both sought to produce the parts with higher opportunity costs and leave the less costly parts to be produced elsewhere. Zhu *et al.* showed that the maximisation was able to achieve the same objective using a lean structure with fewer decision variables and constraints. They also proved that their formulation was computationally faster and more efficient than the corresponding formulation of Wei and Gaither [WG90].

Selim *et al.* [SAV98] proposed a comprehensive mathematical programming formulation with the objective of minimising cost assignment of part operations, machines, workers and tooling to cells. However, this CF model was combinatorially complex and would not be solvable for any real problem. The authors identified sub problems (with fewer constraints and variables) which had been proposed by several researchers and proceeded with a general classification of CF procedures based on their employed methodology and identification of a number of useful suggestions for future research. More specifically one of their observation was that many of the known techniques for the CF problem do not include machine utilisation, multiple machines of the same type nor part operation requirements when larger scale models are taken into account. The work of this thesis will extend the scope for modelling CF based upon most of these suggestions.

Won and Lee [WL04] proposed a modified *p-median* approach for efficient GT cell formation with the objective of maximising the sum of similarities between machines. The authors commented that the original *p-median* formulation [Kus87] when applied to real applications was severely restricted due to two major factors: problem size and software type. Their new formulation had two major advantages when compared with the classical *p-median* model: speedy implementation, and large CF problem capability even when using education-purpose software.

Foulds *et al.* [FFW06] developed a mixed integer mathematical programming model where machine modification was introduced. It is often the case for CF that it is important to be able to reassign parts to additional machine parts in order to create better cell system configuration and also avoid duplication of machines which might be very expensive. Thus, they introduced machine modification to reduce intercellular travel and they claimed that the cost of such modifications could be balanced by the consequent reduction in intercell travel cost. The objective was to minimise the sum of the machine modification costs and the intercell travel. This problem was called

Sustainable Cell Formation Problem (SCFP). For using the current algorithm for large scale problems the authors proposed and analyzed greedy and tabu search heuristics.

2.2.4 Heuristics

Heuristic algorithms have popularly been implemented for many practical applications as they are designed to provide an alternative framework for solving a problem in contrast with a set of restricted rules-constraints that cannot vary. Although heuristic approaches do not guarantee to provide optimal solutions (usually sub-optimal results are derived) they are very useful in producing an acceptable solution in reasonable time. A number of heuristic algorithms were developed recently for the CF problem by Mukattash *et al.* [MAT02], Chan *et al.* [CCI02] and Kim *et al.* [KBB04] and are presented below.

Mukattash *et al.* [MAT02] proposed three heuristic procedures. Given a CF solution, the heuristics were designed to assign parts to the cells in the presence of alternative process plans, multiple alternative machines and processing times. Cell formation with the presence of alternative process plans and multiple types of machines led to the elimination of exceptional elements. When multiple types of machines were considered some exceptional elements were also eliminated. The exceptional elements could be further added to the bottleneck machines thus increasing machine utilisation.

Chan *et al.* [CCI02] developed a heuristic algorithm that addressed problems of machine allocation in cellular manufacturing only when the intra-cell materials flow was taken into account. The proposed algorithm used an adaptive approach to relate machines in a cell by examining the merged part flow weights of machine pairs. The establishment of the part flow weight included practical constraints, such as the part-handling factor and the number of parts per transportation. The objective function employed was to minimise the total travelling score in which the total travelling distance was covered. The current algorithm outperformed other approaches as it provided near optimum solutions.

Kim *et al.* [KBB04] considered a multi-objective machine CF problem. Part route families and machine cells needed to be determined in such a way that minimisation of the total sum of intercell part movements and maximum machine workload imbalance could be achieved. A two-phase heuristic algorithm was proposed. In the first phase, representative part routes with part route families were determined whereas in the second phase the remaining part routes were allocated to part route families. The authors concluded that the two-phase heuristic algorithm was effective in minimising intercell part movements and maximum machine workload imbalance.

2.2.5 Metaheuristics

Metaheuristics form a group of search algorithms that have been mainly developed during the last two decades for the solution of difficult combinatorial optimisation (CO) problems or in other words NP-hard[1] (Non-deterministic Polynomial time hard) problems. The cell formation problem is considered to be a complex and difficult optimisation problem. Many researchers in order to gain more benefits of the CF problem have applied metaheuristic algorithms. Four of the most notable members of the metaheuristics group are:

- Simulated Annealing
- Tabu Search
- Evolutionary Algorithms
- Ant Colony Optimisation

• Simulated Annealing and Tabu Search

Simulated Annealing (SA) and Tabu Search (TS) algorithms have a common characteristic as the search process starts from one initial state (the initial solution) and describes a trajectory in the state space. SA is commonly said to be the oldest among metaheuristics. The origins of the algorithm are in statistical mechanics (see the Metropolis algorithm [MRR+53]) and it was first presented as a search algorithm for CO problems in Kirkpatrick *et al.* [KGJV83]. TS is one of the most successful metaheuristics for the application to CO problems. The basic ideas were introduced by Glover [Glo86], based on his earlier ideas [Glo77]. Both SA and TS have been substantially employed for the CF problems and some of the most significant work is presented here.

Vakharia and Chang [VC97] developed two heuristic methods for the CF problem both based on simulated annealing and tabu search algorithms. The objective function of their model was the minimisation of the total cost of machines required as well as materials handling cost for loads transferred between cells. A considerable amount of data

[1]A problem Π is NP-hard if an algorithm for solving it can be translated into one for solving *every problem* in NP (Non-deterministic Polynomial time). NP-hard therefore means 'at least as hard as any NP problem', although it might in fact be harder.

was considered, such as processing times and transportation costs. The performance of their heuristics was examined using published and industrial data and the result indicated that simulated annealing outperformed tabu search in terms of solution quality and computational complexity.

Aljaber *et al.* [ABC97] modelled the CF problem using a pair of shortest spanning path problems, one for the machines (rows) and one for the parts (columns). Then they employed a modified version of Jaccard's similarity coefficient for calculating the distance between pairs of machines or parts. The authors proposed a tabu search heuristic algorithm for the solution of both problems.

Spiliopoulos and Sofianopoulou [SS03] proposed a two stage heuristic approach for the manufacturing cell design problem and a tabu search scheme for its solution. The first stage tackled parts grouping whereas the second eliminated intercellular traffic flow. The tabu search algorithm, as the third stage to be implemented, integrated proper short and long term memory structures and an overall search strategy for their use. At the code development phase special care was taken to enhance the exploration capability of the algorithm by correlating search statistics with the values of the search parameters. The resulting algorithm was proved to be robust and the results were recorded to be encouraging.

Wu *et al.* [WLW04] considered a CF problem when process plans for parts and production factors such as production volume and cell size were taken into account. The aim was to decompose the manufacturing shop into several manufacturing cells so that the total part flow within the cells can be maximised. For solving this problem a tabu search heuristic algorithm that consisted of a dynamic tabu tenure and a long term memory structure was proposed. Two methods for quickly generating the initial solutions were proposed, namely the group-and-assign and the random approach. Computational results were observed to be very good for a group-and-assign methodology applied to the proposed tabu search approach for small to medium sized problems. The generation of the initial solution in the above way will be a starting point for the work addressed in this thesis and more specifically for the design of the initial heuristic solution reflecting the CF model requirements.

Lei and Wu [LW06] presented a Pareto optimality based multi-objective tabu search (MOTS) algorithm to the part/machine grouping problem with multiple objectives: minimisation of the weighted sum of intercell and intra-cell moves and minimisation of the total cell load variation. A new approach was proposed to determine the non-dominated solutions among the solutions produced by the tabu search algorithm. The computational results demonstrated the strong ability of MOTS in multi-objective optimisation.

• **Evolutionary Algorithms and Ant Colony Optimisation**

Both Evolutionary Algorithms (EA) and Ant Colony Optimisation (ACO) could be characterised as population-based searches. Evolutionary algorithms (EA) are inspired by the nature's capability to evolve living beings well adapted to their environment. EA algorithms prove to be particularly popular due to their added characteristic of being able to search the solutions' space not from a single point but from a population of points in parallel. There are several variants of evolutionary algorithms but one of the newest and the most popular members is Genetic Algorithms (GAs) [Koz92], which were initiated by Holland [Hol75]. ACO is one of the newest metaheuristics for the application to CF problems. The basic ideas of ACO were introduced in Dorigo *et al.* [DMC96], [DDCS99]. ACO was inspired by the foraging behaviour of real ants [DAGP90] and its search process can be described as the evolution of a probability distribution over the search process.

Venugopal and Naredran [VN92] were the first researchers to approach the CF problem using GAs. Their objective was the minimisation of the intercell movements of parts and balancing of loads in the cells. A different population of solutions was employed for each of the objectives. The solution representation was simple and efficient where each machine in the plant corresponded to a gene in the chromosome. The value of the gene defined the owing cell of the respective machine. The total number of cells was predefined and the processing time of parts was also taken under consideration. Gupta [GGKS96] enhanced this formulation by considering the intracell movements of parts and the intracell layout. Special care was taken to ensure that no cell remained empty during this evolutionary process.

Gravel [GNP98] considered a version of the cell formation problem that allowed the existence of alternative process plans for the parts. A double-loop EA was employed for the solution of the problem with the objective of minimising the volume of intercell moves and balancing the workload within cells. For the external loop of the EA, Venugopal and Naredran's coding for the assignment of machines to cells was used. A second internal loop that determined the allocation of process plans to parts was employed for the evaluation of solution created in the external loop. Different multi-objective optimisation approaches were tested, including the epsilon-constraint approach and the weighted-sum approach.

Mak *et al.* [MWW00] proposed an adaptive genetic approach as an effective means of providing the optimal solution to the cell formation problem. The objective was to group parts and machines into clusters by sequencing the rows and columns of a part/machine incidence matrix to maximise the bond energy measure of the matrix. The proposed approach was different from the use of traditional genetic algorithms, because an adaptive scheme was adopted to adjust the genetic parameters during the genetic search process. The effectiveness of the approach was demonstrated by applying it to numerical results and a number of benchmark problems obtained from the literature.

Solimanpur *et al.* [SVS04b] formulated a multi-objective integer programming model for the cell formation problem with independent cells. A genetic algorithm with multiple fitness functions was proposed to solve their model. Two features made their proposed algorithm to differ from previous approaches i.e. : (1) a systematic uniform design-based technique which was used to determine the search directions, and (2) the search of the solution space in multiple directions instead of a single direction. The results validated the effectiveness of the proposed algorithm.

Vila Gonçales Filho and José Tiberti [VGFT06] proposed a new genetic algorithm for the cell layout problem with several new features such as chromosome codification scheme, correction mechanism, crossover and mutation operators that worked directly with the group of machines as opposed to individual machines. Tests using published data sets proved that the algorithm could find the group structure present in data sets.

James *et al.* [JBK07] presented a hybrid grouping genetic algorithm for the cell formation problem that combined a local search with a standard grouping genetic algorithm to form part/machine cells. Computational results were produced using the grouping efficacy measure for a set of cell formation data sets borrowed from the literature. The hybrid grouping genetic approach outperformed the standard grouping genetic algorithm by exceeding the solution quality on all test problems and by reducing the variability among the solutions found. Overall, the proposed algorithm performed well on all test problems by either exceeding or matching the solution quality of the results presented in the literature.

Wu *et al.* [WCHWY07] proposed a hierarchical genetic algorithm to simultaneously form manufacturing cells and determine the group layout of cellular manufacturing. The main features of this algorithm was the development of a hierarchical chromosome structure to encode two important cell design decisions, a new selection scheme to dynamically consider two correlated fitness functions and a group mutation operator to increase the probability of mutation. From the computational results it was proved that both proposed structures and operators developed were effective in terms of improving solution quality as well as accelerating convergence.

ACO was recently applied for the CF problem by Solimanpur *et al.* [SVS04a]. This paper presented an affective and efficient ant algorithm for the inter-cell layout problem, which was formulated as a quadratic assignment problem. The sequence of operations and the production volume of parts were introduced as two major factors effecting the flow of materials. A mathematical model was proposed for calculating the material flow among cells whereas an ant algorithm was proposed to solve the formulated problem. It was proved that the proposed algorithm outperformed previous techniques in terms of the total material handling and distance in all problems.

2.2.6 Fuzzy Logic

The decision making process in a manufacturing system often involves uncertainties and ambiguities. Under such circumstances, fuzzy methodologies have been proved to be effective tools for taking fuzziness into consideration. A significant number of researchers have applied fuzzy clustering and fuzzy mathematics by employing fuzzy set theory for addressing uncertainty or vagueness in system parameters whereas, a limited number of researchers attempted to use fuzzy mathematical programming in the design of manufacturing systems.

- **Fuzzy Clustering**

Chu and Hayya [CH91] applied a fuzzy c-means clustering algorithm, developed by Bezdek [Bez81], to identify the part family configuration for each part and also to provide the degree of membership associated with each part family. The fuzzy c-means clustering can be classified as non-hierarchical, therefore it suffers from the need to specify *a priori* the number of part families. However, the technique was unaffected by exceptional elements. Overall, the advantage of this methodology was that it provided the designer with more information than the available given from a "crisp" definition of families and cells.

Zhang and Wang [ZW92] proposed a fuzzy version of single linkage clustering and rank order clustering algorithms, where a non-binary part/machine matrix was considered. As mentioned earlier, SLCA requires a secondary process of component allocation to the machine groups after the machine grouping whereas, ROC algorithm performes both tasks simultaneously. In addition, fuzzy ROC preserved all the benefits from ROC and also ranked machines with priority so problems like exceptional and bottleneck elements and machine capacity could also be addressed.

Gindy *et al.* [GRC95] developed an extended version of fuzzy *c*-means clustering algorithm for component grouping with cluster validation procedure. With the latter the authors were aiming at component grouping for cellular manufacturing systems when maximum diversity (i.e. minimum overlapping in terms of the repeated machines between cells) was considered. The validity measure proved very useful in optimising component partitioning by forming component groups with the maximum compactness (maximum number of components in each group requiring the full set of machine resources available in a cell) and of machining cells with a minimum number of repeated machines. The validity measure was experimentally assessed using industrial data and compared with similar validity measures used in fuzzy clustering analysis. The results showed that the fuzzy clustering approach and the validity measure provide a realistic solution methodology useful for part family formation.

Masnata and Settineri [MS97] tailored a fuzzy *c*-means clustering algorithm for developing a non-binary approach to group technology based on the capabilities of fuzzy logic. They also integrated fuzzy *c*-means with the strategy for minimum make-span scheduling.

Ravichandran and Rao [RR01] proposed a new fuzzy clustering algorithm and a new similarity coefficient for sub-grouping parts/machines before the optimal grouping and for optimal grouping. For analysing output, the proposed algorithm performed quite well when compared to a fuzzy *c*-means clustering algorithm and other conventional algorithms. The results showed that the new approach to fuzzy part-family formation and grouping efficiency provided a more realistic solution methodology for part family formation in cellular manufacturing applications.

• Fuzzy Mathematics

Xu and Wang [XW89] incorporated uncertainty or imprecision in the measurement of similarity between parts by employing fuzzy mathematics. In considering issues related to part families two intermingled themes appeared:

(i). Fuzzy mathematics as part clustering
The authors selected a sample part set, determined the features which were relevant to the machining process under consideration and defined a membership function for each selected feature. Then they calculated a similarity matrix and assigned a value λ corresponding to fuzzy equivalence analysis of fuzzy classification to obtain a set of part families [LSC83].

(ii). Fuzzy mathematics as pattern recognition
Pattern recognition was applied to assign new parts into part families. By using this technique the authors calculated the closeness values between the new part and the existing part families and then assigned the new part to the family which showed the maximum closeness value.

The results produced, when several rotational parts from a local company were tested, were satisfactory.

Ben-Arieh and Triantaphyllou [BAT92] used fuzzy set theory for grouping a significant number of different part types into groups based on predetermined set of features. The methodology proposed was based on a modification to the revised analytical hierarchy process [Saa80]. The discrete values of the fuzzy features were compared to each other. This allowed the same features to be used for classification of a large collection of parts, unlike the more traditional part to part comparison. For the fuzzy features matrices were generated and the eigenvectors for each matrix were calculated expressing the feature value for each part. Then the weight (importance) of each feature was estimated by using pairwise comparisons. The last step was to cluster the parts based on features values and weights. This methodology had many advantages: the user was able to define the features used for parts grouping, the user could define the importance of the features in a specific scenario, the methodology was not constrained by the number of parts but by the number of different values the features could have. However, a major disadvantage for the existent methodology was the estimation of the pairwise comparisons. In the case of a large system, the required number of comparisons increased very fast.

Gill and Bector [GB97] developed an approach, based on fuzzy linguistics, to deal with data quantification and construction of distance measures for part family formation.

• **Fuzzy Mathematical Programming**

Tsai *et al.* [TCB97] proposed a Mixed Integer Mathematical Programming (MIMP) formulation with a single objective to minimise the cost of eliminating exceptional elements. They illustrated how a Fuzzy Mixed Integer Mathematical Programming (FMIMP) approach can be used to solve the CF problem when applied to the initial deterministic MIMP model. Two membership functions with four operators, including a newly proposed operator, were applied and the results compared. They observed that the proposed operator always outperformed the other three operators and it was more stable and robust. The authors concluded that FMP provided a better and a more flexible way of presenting the problem domain than the traditional MP.

2.3 Summary

This review covers a significant number of methodologies developed for the cell formation problem in the past decades. Methodologies employed were discussed through a core classification consisted of array based methods, graph based methods, mathematical programming methods, heuristics, metaheuristics and fuzzy theory. The evolution of cellular manufacturing systems was mainly based on academic studies.

Chapter 3

Mixed Integer Mathematical Programming Model for CF

3.1 Introduction

The activities required for the formation of a system of manufacturing cells can be described by the following steps:

(a). Assigning part families to groups of machine types;

(b). Finding lot sizes, i.e. quantities, of the parts to be produced;

(c). Determining the number of machine instances for each machine type;

(d). Assigning parts to individual machines;

(e). Grouping the individual machines into cells.

For the purpose of this chapter, the researcher is interested in investigating steps (d) and (e) and solving them simultaneously when a number of key constraints and an enhanced objective function are proposed. The most effective solution procedures that solve the problems represented by activities (d) and (e) simultaneously involve integer mathematical programming models.

3.2 Initial model for CF

An integer programming model developed by Foulds *et al.* [FFW06] assigns parts to machines and groups machines to cells simultaneously. Because this was a comprehensive model but still had scope of further development it was decided to use this as a starting point. The minimisation of the intercell movement of parts was considered as an objective function and a number of constraints were included, some of which are described next.

- Utilisation levels. Ensure that existing machines in the plant aren't overloaded and that new arriving machines are economically justifiable to be included in a cell.
- Cell size. The cell size is determined by the number of machines it includes and needs to be controlled for the following reasons: First, space availability imposes constraints on the number of machines in a cell. Second, in the case where the plant is run by operators the larger the size of the cell the more difficult for it to be controlled.
- Machine capacity. The capacity of a machine should be adequate in order to be able to process all the parts assigned to it.
- Multiple machines of the same type. Many machines of a specific type ensure that all the relevant parts are processed without duplication of machines. When the latter happens increment in the equipment cost occurs.

3.2.1 Notation for model formulation

The following notation has been used for the model formulation:

- **Index Set**

i	machine type index	$i = 1, \dots, NM$
j	part index	$j = 1, \dots, NP$
q	cell index	$q = 1, \dots, NC$
k	machine instance index	$k = 1, \dots, KM_i$

- **Input Parameters**

E_{MIN}	minimum number of machines allowed in a cell
E_{MAX}	maximum number of machines allowed in a cell
$UTIL_{i,j}$	utilisation of machine of type i by part j
NC	number of cells to be created
NM	number of machines
NP	number of parts
KM_i	number of machine instances for each machine of type i

$KMAX$	the maximum number of machine instances recorded for all machine types in KM_i
M_i, P_j	denote machine of type i and part of type j respectively
M_i^k	denotes the k^{th} individual machine instance of type i

- **Decision Variables**

$x_{i,j,q}$	number of machines or fraction thereof (in terms of machine capacity used) of type i that process part j on cell q
$y_{i,k,q}$	=1 if k^{th} machine of type i is assigned to cell q, 0 otherwise
$w_{j,q}$	=1 if part j is processed in cell q, 0 otherwise
v_q	=1 if cell q is formed, 0 otherwise

3.2.2 Model Formulation and Solution

A mathematical programming model representing the cell formation is as follows:

$$Min \sum_{j=1}^{NP} \sum_{q=1}^{NC} w_{j,q} \tag{3.1}$$

subject to

$$\sum_{q=1}^{NC} y_{i,k,q} = 1 \ \ \forall\, i,k \tag{3.2}$$

$$\sum_{q=1}^{NC} x_{i,j,q} = UTIL_{i,j} \ \ \forall\, i,j \tag{3.3}$$

$$\sum_{j=1}^{NP} x_{i,j,q} \leq \sum_{k=1}^{KM_i} y_{i,k,q} \ \ \forall\, i,q \tag{3.4}$$

$$\sum_{i=1}^{NM} \sum_{k=1}^{KM_i} y_{i,k,q} \leq v_q \times E_{MAX} \ \ \forall\, q \tag{3.5}$$

$$\sum_{i=1}^{NM} \sum_{k=1}^{KM_i} y_{i,k,q} \geq v_q \times E_{MIN} \ \ \forall\, q \tag{3.6}$$

$$v_{q+1} \leq v_q \ \ \forall\, q \tag{3.7}$$

$$\sum_{q=1}^{NC} q \times y_{i,k,q} \leq \sum_{q=1}^{NC} q \times y_{i,k+1,q} \ \ \forall\, i,k \tag{3.8}$$

$$x_{i,j,q} \leq UTIL_{i,j} \times w_{j,q} \ \ \forall\, i,j,q \tag{3.9}$$

$$y_{i,k,q},\ v_q,\ w_{j,q} = 0 \ or\ 1; \quad 0 \leq x_{i,j,q} \quad \forall\ i,j,k,q$$

The objective function (3.1) minimises the number of distinct cells used by each part. Constraint (3.2) ensures that the k^{th} machine of type i must be assigned to exactly one cell. Constraint (3.3) serves to satisfy the requirements for processing part j on machine i: the number of machines required to process part j in cell q is equal to the utilisation of machine i required to process part j in cell q. Constraint (3.4) serves to indicate that the total number of machines of type i used in cell q should be less than or equal to the number of machines of type i assigned to cell q. Constraints (3.5), (3.6) limit the number of machines in each cell. Constraint (3.7) ensures that cells are formed in successive numerical order for convenience. Constraint (3.8) assigns multiple instances of machine of type i to lower numbered cells in successive numerical order. Constraints (3.7) and (3.8) are included to eliminate certain symmetries. Lastly, constraint (3.9) picks out the intercellular movements.

The above model was rigorously examined by the researcher for a good understanding of the objective and the constraints involved. The software package XPRESS-MP [XM], general purpose mathematical programming solver, was employed for finding the optimum value of the objective function and its related solutions. At this stage the data employed consist of seven machine types and ten parts respectively. Also the maximum allowable number of cells to be created is assigned to five. The maximum number of machines allowed in each cell is defined to be of magnitude six, whereas the minimum as four. Moreover, the part machine utilisation needed is presented in Table 3.1.

Table 3.1: Part/Machine Utilisation

M_i/P_j	1	2	3	4	5	6	7	8	9	10
M_1	0.0	0.3	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.0
M_2	0.0	0.0	0.0	1.0	0.5	0.0	0.9	0.0	0.1	0.6
M_3	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.2	0.1	0.0
M_4	0.0	1.2[1]	0.0	0.9	0.5	0.4	0.0	0.0	0.0	0.0
M_5	0.4	0.2	0.0	0.8	1.1	1.0	0.6	1.0	1.0	0.0
M_6	0.0	0.7	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.3
M_7	0.0	0.5	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0

Also, the number of machines of type i, KM_i, needed to process any part j can be

[1]When the amount of the machine capacity required from a specific part to be processed by a machine of type i is greater than one, then more that one machines instances of type i will be utilised.

mathematically calculated by the equation (3.10).

$$KM_i = \lceil \sum_{j=1}^{NP} UTIL_{i,j} \rceil \tag{3.10}$$

Therefore, for the part/machine utilisation in Table 3.1 the KM_i will be as follows:

$$KM_i = [1 \;\; 4 \;\; 2 \;\; 3 \;\; 7 \;\; 2 \;\; 1] \tag{3.11}$$

Appendix A.1 presents the output generated when the above data is employed. From the produced solution four cells are created in total, each consisting of machine instances of certain types. The cell configuration is shown in Table 3.2.

Table 3.2: Machine Cell Allocation 1

Cell 1:	M_2^1, M_4^1, M_5^1, M_5^2, M_6^1
Cell 2:	M_2^2, M_2^3, M_4^2, M_5^3, M_5^4, M_7^1
Cell 3:	M_2^4, M_3^1, M_5^5, M_5^6, M_5^7, M_6^2
Cell 4:	M_1^1, M_3^2, M_4^3, M_5^7

For the current model it is assumed that the machine utilisation for processing a part j is equal to the processing time of part j in machine i. For this reason no time element is considered for the current model. It is also assumed that for each machine of type i only a maximum of a unity of its capacity can be spent for processing a part j. By taking the latter into consideration and in conjunction with the part/machine utilisation, Table 3.1, and the machine sequences (routings) of parts, Table 3.3, the part/machine assignment could be obtained as shown in Table 3.4.

Table 3.3: *Desired* Part Machine Operation

P_j	Machine Sequence
1	M_5
2	M_1, M_4, M_5, M_6
3	M_3
4	M_1, M_2, M_4, M_5, M_7
5	M_2, M_5, M_4, M_7
6	M_4, M_5
7	M_5, M_2, M_6
8	M_1, M_3, M_5
9	M_3, M_5, M_2
10	M_2, M_6

It can be seen from Appendix A.1 that the total cost, i.e. the value of the objective function, is equal to twelve which implies that two intercellular movements of parts

Table 3.4: Part/Machine Assignment

P_j/M_i^k	1	2	3	4	5	6	7	8	9	10	$\sum_{j=1}^{10} UTIL_{i,j}$
M_1^1	0.0	0.3	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.5
M_2^1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.6
M_2^2	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.5
M_2^3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
M_2^4	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0	0.1	0.0	1.0
M_3^1	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0	1.0
M_3^2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.2
M_4^1	0.0	0.6	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	1.0
M_4^2	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	1.0
M_4^3	0.0	0.6	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	1.0
M_5^1	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2
M_5^2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
M_5^3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
M_5^4	0.0	0.0	0.0	0.8	0.1	0.0	0.0	0.0	0.0	0.0	0.9
M_5^5	0.4	0.0	0.0	0.0	0.0	0.0	0.6	0.0	0.0	0.0	1.0
M_5^6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
M_5^7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
M_6^1	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0
M_6^2	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.8
M_7^1	0.0	0.0	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.4

occur since in total ten distinct part cell allocations are counted. The cells formed and the flow of parts together with all the intercellular movements are illustrated in Figure 3.1. Note that in cell one, M_5^2 is fully dedicated to part six; in cell two, M_5^3 and M_2^3 are dedicated to parts five and ten; in cell three, M_5^6 is dedicated to part nine; and in cell four, M_5^7 is dedicated to part eight. Also parts two and four are those causing the two intercellular moves.

It is also worth commenting on the part routing machine sequence. For example part four follows a slightly different machine operation sequence in Figure 3.1 from the one indicated in Table 3.3. More specifically, part four visits machine instances in the following order: M_1, M_4, M_2, M_5 and M_7. The latter though is expected since no part machine operation sequence is included within the current model. In order to illustrate how the machine sequence could affect the cell formation configuration, Figure 3.2 is designed. All parts this time follow strictly the sequence of machines in Table 3.3. Thus, employing part machine operation sequence makes the system more complex but incorporates realism. The latter will be studied in section 3.4.

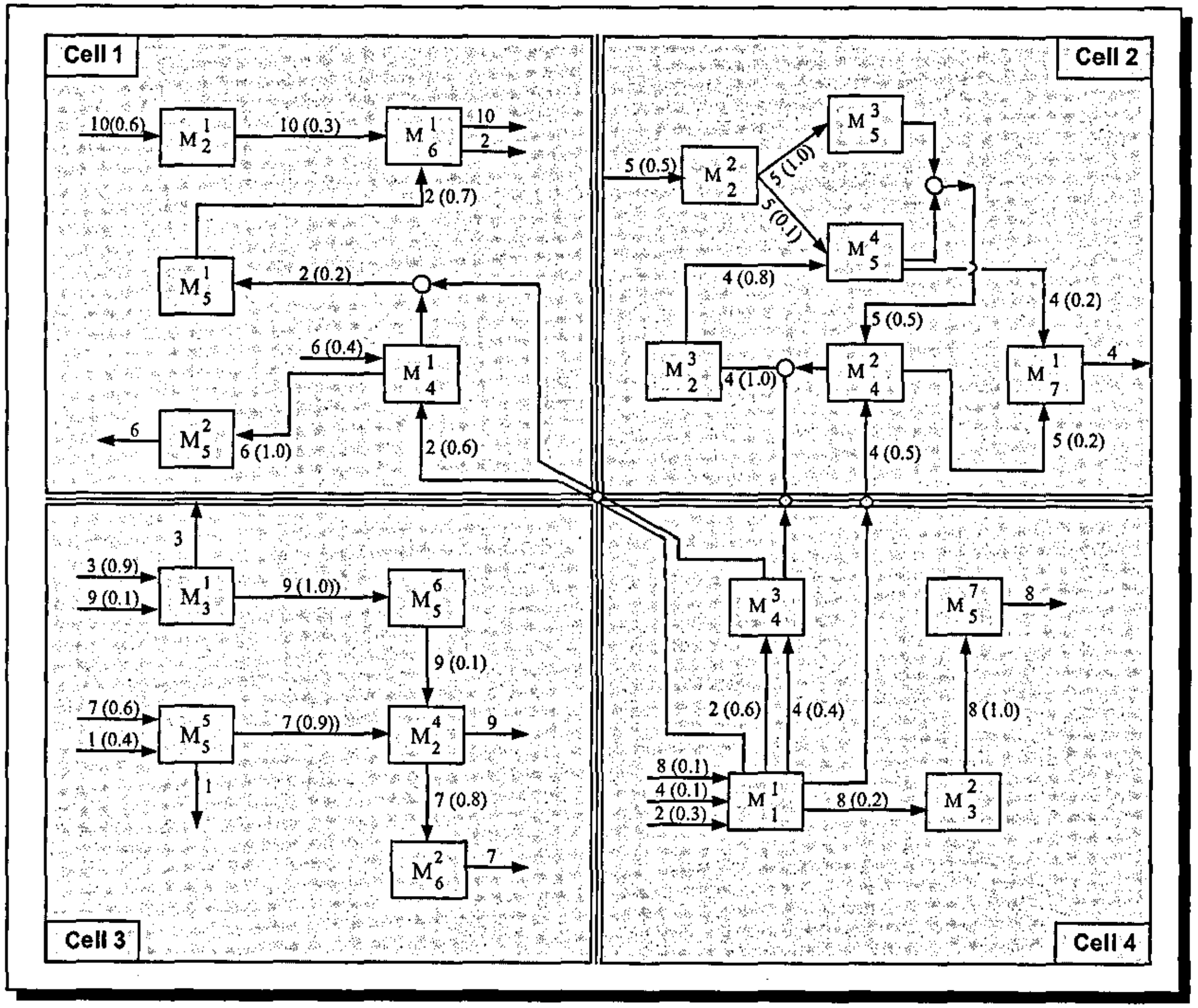


Figure 3.1: Part Flow: No Part Machine Sequence Employed

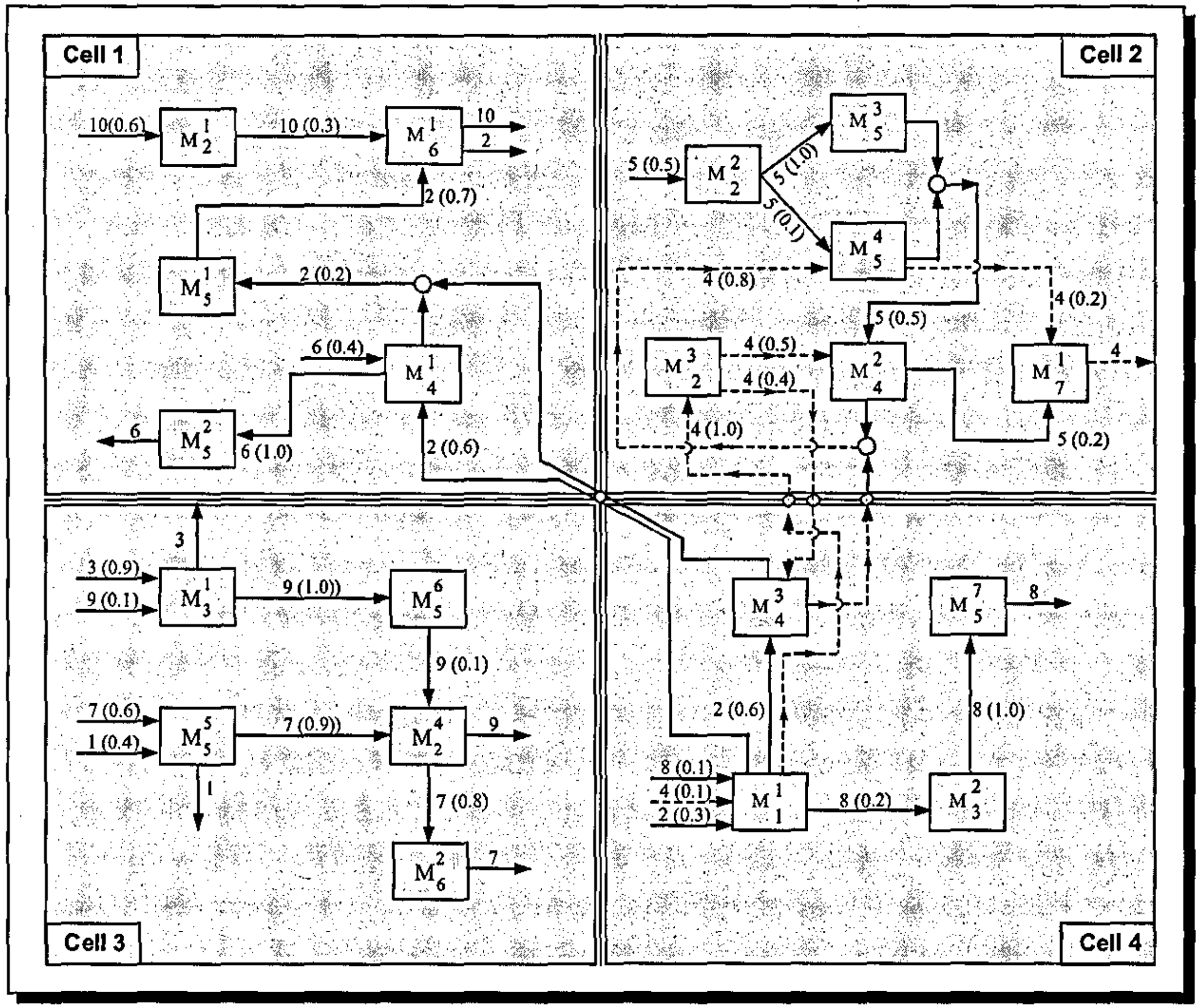


Figure 3.2: Part Routing with Part Machine Sequence

3.3 An Extended Cell Formulation Model

In addition to the constraints included in the initial model, *set-up costs* for setting up machines required by certain parts every time they start to be processed are now considered. This aspect was ignored in previous model making the problem less realistic. It is worth examining whether this update will affect significantly the overall part/machine cell allocation. In order for the set-up costs to be considered a few things need to be added in the initial model. More specifically, in addition to the old index set, parameters and decision variables, (section 3.2.1), two new parameters and one decision variable are included as follows:

$SETUP_{i,j}$	set-up cost of machine i needed to process part j
$UTIL_{MIN}$	minimum amount of utilisation in $UTIL_{i,j}$ matrix
$M_{j,q}$	cost of allocating part j in cell q

$s_{i,j,q}$	integer number of machines of type i that will be used by part j in cell q

Moreover, in addition, to the constraints (3.2)-(3.9), defined in section (3.2.2), the following two constraints are now added:

$$x_{i,j,q} \leq s_{i,j,q} \quad \forall\, i, j, q \tag{3.12}$$

$$x_{i,j,q} \geq UTIL_{MIN} \times s_{i,j,q} \quad \forall\, i, j, q \tag{3.13}$$

$$s_{i,j,q} \ \ integer$$

Also the new objective function is produced as seen in equation 3.14.

$$Min \ \ (\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q} \times w_{j,q}) + \sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j} \times \sum_{q=1}^{NC} s_{i,j,q})) \tag{3.14}$$

Constraint (3.12) simply serves to capture that the total number of machines (in terms of machine utilisation) required to process part j in cell q is less than or equal to the integer number of machines of type i that will used by part j in cell q. Constraint (3.13) forces variable s to get the value *zero* whenever an x variable is *zero* and is not strictly necessary but aids branch and bound.

Objective, (3.14), ensures that two requirements are satisfied:

- *Minimise number of intercellular movements of parts*

- *Minimise set-up costs when allocating machines to cells*

It was decided that the cost coefficient, $M_{j,q}$, for the first product of the objective function should take the value ten and is used to weight intercellular movements of parts against total set-up cost.

For testing the above model and for comparison purposes of the current model with the initial, it is assumed that the number of machines and parts remains the same, i.e. seven machines and ten parts. Also the part machine utilisation as seen in Table 3.1 is employed here as well. For the added feature, i.e. the set-up costs, the part/machine set-up costs are presented in Table 3.5.

Table 3.5: Part/Machine Set-up Costs

M_i/P_j	1	2	3	4	5	6	7	8	9	10
M_1	0.00	2.95	0.00	1.96	0.00	0.00	0.00	1.92	0.00	0.00
M_2	0.00	0.00	0.00	5.12	2.42	0.00	5.05	0.00	1.54	2.16
M_3	0.00	0.00	2.93	0.00	0.00	0.00	0.00	1.94	1.45	0.00
M_4	0.00	5.54	0.00	2.91	2.42	2.38	0.00	0.00	0.00	0.00
M_5	2.91	2.59	0.00	2.88	5.42	4.91	2.63	4.93	4.81	0.00
M_6	0.00	2.82	0.00	0.00	0.00	0.00	2.73	0.00	0.00	2.49
M_7	0.00	0.00	0.00	2.12	2.12	0.00	0.00	0.00	0.00	0.00

3.3.1 Solution for Extended Model

From the extended model four new cells are created (see Appendix A.2) each consisting of the following machine instances:

Table 3.6: Machine Cell Allocation: Extended Model

Cell 1:	M_2^1, M_4^1, M_5^1, M_5^2, M_6^1
Cell 2:	M_1^1, M_3^1, M_4^2, M_5^3
Cell 3:	M_2^2, M_2^3, M_4^3, M_5^4, M_5^5, M_7^1
Cell 4:	M_2^4, M_3^2, M_5^6, M_5^7, M_6^2

Comparing Tables 3.2 and 3.6 it is observed that the new cells created contain different types of machines. Also, using Tables 3.1, 3.3 in conjunction with Table 3.6, part/machine assignment is created and presented in Table 3.7. From Tables 3.6 and 3.7 it can easily be detected the effect of the newly added feature to the CF configuration.

For better illustration of the actual part machine cell allocation, the cells formed and the flow of parts see Figure 3.3. Similar to the initial model, parts two and four are processed in more than one cell. Note also that in cell one, M_5^2 is dedicated to part six; in cell two M_5^3 is dedicated to part eight; and in cell three M_5^5 and M_2^3 are dedicated to parts five and four respectively.

Although the involvement of set-up costs within the CF problem could affect the part machine cell allocation, it won't change significantly the value of the objective function as the latter is driven mostly by the number of distinct allocation of parts to cells.

Table 3.7: Part/Machine Assignment for Extended Model

P_j/M_i^k	1	2	3	4	5	6	7	8	9	10	$\sum_{j=1}^{10} UTIL_{i,j}$
M_1^1	0.0	0.3	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.5
M_2^1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.6
M_2^2	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.5
M_2^3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
M_2^4	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0	0.1	0.0	1.0
M_3^1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.2
M_3^2	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0	1.0
M_4^1	0.0	0.6	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	1.0
M_4^2	0.0	0.6	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	1.0
M_4^3	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	1.0
M_5^1	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2
M_5^2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
M_5^3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
M_5^4	0.0	0.0	0.0	0.8	0.1	0.0	0.0	0.0	0.0	0.0	0.9
M_5^5	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
M_5^6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
M_5^7	0.4	0.0	0.0	0.0	0.0	0.0	0.6	0.0	0.0	0.0	1.0
M_6^1	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0
M_6^2	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.8
M_7^1	0.0	0.0	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	0.4

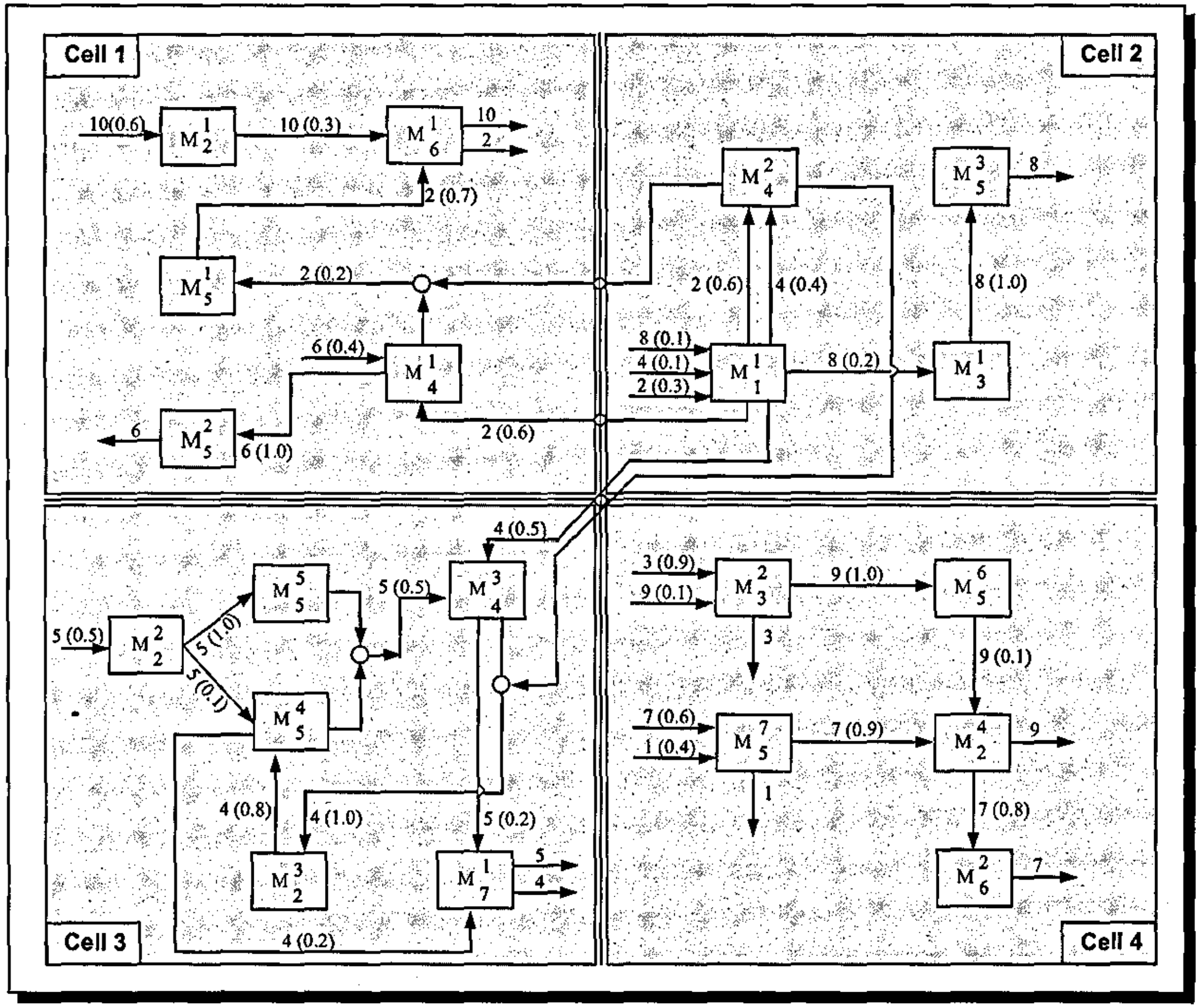


Figure 3.3: Part/Machine Cell Allocation 2

3.4 Complete Model Formulation

Within this section a more sophisticated model will be presented where the part machine operation sequence is taken into account incorporating more realism into the system. The current objective function involves minimisation of both the distinct allocation of parts to cells and machine set-up costs. The former element of the objective function will now be complemented with an additional feature. Before proceeding with more details, new indices, parameters, variables and constraints needed together with a presentation of the complete form of the model (objective and constraints) are presented next.

3.4.1 Notation for model formulation

- **Index Set**

z	machine operations index	$z = 1, \dots, ZOPER$[2]
r	machine operations index	$r = 1, \dots, ZOPER$

- **Input Parameters**

$ZOPER$	number of machine operations
$ZTYPES_j$	number of different operations (machine types) required by part j
$L_{j,z}$	for part j the machine used for the z^{th} machine operation in sequence
$UTIL_{MAX}$	biggest amount of machine utilisation used
A_j	cost for part j traveling back to an already visited cell

- **Decision Variables**

$extra_{q,j,L_{j,z}}$	=1 if after the z^{th} operation of part j in cell q the part leaves cell q but returns later
$xx_{L_{j,z},j,q}$	=1 if part j is processed in cell q for z^{th} machine operation, 0 otherwise

3.4.2 Model Formulation

The complete formulation of the mathematical programming model is shown below:

$$Min \ (\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q}\times w_{j,q})+\sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j}\times\sum_{q=1}^{NC}s_{i,j,q})+\sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j\times extra_{q,j,i})) \tag{3.15}$$

subject to

[2]$ZOPER$ is defined within the Input Parameters section.

$$\sum_{q=1}^{NC} y_{i,k,q} = 1 \quad \forall\, i,k \tag{3.16}$$

$$\sum_{q=1}^{NC} x_{i,j,q} = UTIL_{i,j} \quad \forall\, i,j \tag{3.17}$$

$$x_{i,j,q} \leq s_{i,j,q} \quad \forall\, i,j,q \tag{3.18}$$

$$x_{i,j,q} \geq UTIL_{MIN} \times s_{i,j,q} \quad \forall\, i,j,q \tag{3.19}$$

$$\sum_{j=1}^{NP} x_{i,j,q} \leq \sum_{k=1}^{KM_i} y_{i,k,q} \quad \forall\, i,q \tag{3.20}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \leq v_q \times E_{MAX} \quad \forall\, q \tag{3.21}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \geq v_q \times E_{MIN} \quad \forall\, q \tag{3.22}$$

$$v_{q+1} \leq v_q \quad \forall\, q \tag{3.23}$$

$$\sum_{q=1}^{NC} q \times y_{i,k,q} \leq \sum_{q=1}^{NC} q \times y_{i,k+1,q} \quad \forall\, i,k \tag{3.24}$$

$$x_{i,j,q} \leq UTIL_{i,j} \times w_{j,q} \quad \forall\, i,j,q \tag{3.25}$$

$$x_{i,j,q} \leq UTIL_{MAX} \times xx_{i,j,q} \quad \forall\, i,j,q \tag{3.26}$$

$$x_{i,j,q} \geq UTIL_{MIN} \times xx_{i,j,q} \quad \forall\, i,j,q \tag{3.27}$$

$$xx_{L_{j,z},j,q} + xx_{L_{j,r},j,q} - \sum_{zz=z+1}^{r-1} xx_{L_{j,zz},j,q} \leq extra_{q,j,L_{j,z}} + 1 \quad \forall\, q,j,z,r \tag{3.28}$$

$$y_{i,k,q},\ v_q,\ w_{j,q},\ extra_{q,j,i},\ xx_{i,j,q} = 0 \text{ or } 1;\quad 0 \leq x_{i,j,q};\quad s_{i,j,q}\ integer\ \forall\, i,k,j,q \tag{3.29}$$

New constraints added are (3.26), (3.27) and (3.28). Both constraints (3.26) and (3.27) ensure that whenever a part uses a machine or a fraction thereof ($x_{i,j,q} > 0.0$) variable

$xx_{i,j,q}$ is assigned the value 1, otherwise it is assigned to 0. The *key constraint* (3.28) picks out the number of times a part travels back to a cell for a later machine operation. Please note that it is assumed that $z < r$. A part j, whose z^{th} machine operation is processed in cell q, could revisit the cell q for a later machine operation i.e. $(r)^{th}$, only when the 2^{nd} machine operation $(z+1)$ is not processed within the same cell. In this case the value of $extra_{q,j,L_{j,z}}$ is assigned to 1.

Objective function, (3.15), ensures that three requirements are now satisfied:

- *Minimise number of distinct cells used by each part*
- *Minimise set-up costs when allocating machines to cells*
- *Minimise number of times a part revisits a cell for a later machine operation*

The latter mentioned item acts as an extra feature for the minimisation of the inter-cellular movements of parts. The inclusion of part machine operation sequence within the system can cause an increment on the number of possible forward and backward movements of parts. Note that the first part of the objective function counts the movements of parts between two cells only once. No indication of direction exists. On the other hand, the last item of the objective function simply serves to examine, count and minimize any revisits of a part to a cell. For simplicity reasons the value of cost coefficient A_j is assigned the value one.

The XPRESS-MP file for the implementation of the final form of the integer programming model can be seen in Appendix A.3. The data employed to test the model's performance are the same with those used for the initial model where the number of machines, NM, and number of parts, NP are seven and ten respectively, number of cells, NC, is five, and the maximum and minimum number of machines, E_{MAX} and E_{MIN}, is six and four respectively. Also the part/machine utilisation, $UTIL_{i,j}$, is shown in Table 3.1 and the number of instances needed by each part, KM_i, is presented in equation (3.11). Moreover, the number of operations for each part, $ZTYPES_j$ can be seen in equation (3.30), whereas the part machine sequence, $L_{j,z}$, is provided in matrix (3.31).

$$ZTYPES_j = [1\ \ 4\ \ 1\ \ 5\ \ 4\ \ 2\ \ 3\ \ 3\ \ 3\ \ 2] \qquad (3.30)$$

$$L_{j,z} = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 1 & 4 & 5 & 6 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 1 & 2 & 4 & 5 & 7 \\ 2 & 5 & 4 & 7 & 0 \\ 4 & 5 & 0 & 0 & 0 \\ 5 & 2 & 6 & 0 & 0 \\ 1 & 3 & 5 & 0 & 0 \\ 3 & 5 & 2 & 0 & 0 \\ 2 & 6 & 0 & 0 & 0 \end{bmatrix} \tag{3.31}$$

Comparing the solutions produced from both the current model (see Appendix A.3.1), where part machine operation is taken into account, and the extended model (see Appendix A.2), where no sequence was considered, it can be observed that although the value of the objective function remains the same the cell machine allocation changes. The latter happens since every part now follows strictly its part machine operation sequence resulting in a significant change to the configuration of the CF system.

Please note also that for the current model no part revisits a cell for a later operation since $extra_{q,j,i}$ is 0 for all parts, machines and cells. Although this is happening when current data set is employed it could change with the usage of a different problem instance.

• Illustration of Complete Model's Operation: Example (I)

In order to illustrate the operation for the complete model a new problem instance is assumed. The machine operation sequence adopted here is presented in Table 3.8. It can be seen from the latter that part 2 is now has a different and longer machine operation sequence compared to the sequence that it initially had, as it was shown in Table 3.3.

Moreover, the number of allowable cells and the maximum and the minimum number of machines allowed in a cell are now changing to three, eight and six respectively. The output produced according to the above specifications is presented in Appendix A.3.2. The value of $extra$ variable for $q = 2$, $j = 2$ and $i = 4$ is 1, i.e. $extra_{2,2,4} = 1$. This means that part 2 after the operation on machine 4 in cell 2 leaves cell 2 but returns later.

Table 3.8: Example(I):Machine Sequence of Parts

P_j	Sequence
1	M_5
2	M_1, M_4, M_5, M_7, M_6
3	M_3
4	M_1, M_2, M_4, M_5, M_7
5	M_2, M_5, M_4, M_7
6	M_4, M_5
7	M_5, M_2, M_6
8	M_1, M_3, M_5
9	M_3, M_5, M_2
10	M_2, M_6

In order to be able to demonstrate this graphically, the part/machine assignment for current CF system is needed. For producing the latter the part/machine utilization matrix, as shown in Table 3.9, and the new machine cell allocation, presented in Table 3.10, are used in conjunction. The result is shown in Table 3.11.

Table 3.9: Example(I): Part/Machine Utilisation

P_j/M_i	1	2	3	4	5	6	7	8	9	10	KM_i
M_1	0.0	0.3	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.0	1
M_2	0.0	0.0	0.0	1.0	0.5	0.0	0.9	0.0	0.1	0.6	4
M_3	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.2	0.1	0.0	2
M_4	0.0	1.2	0.0	0.9	0.5	0.4	0.0	0.0	0.0	0.0	3
M_5	0.4	0.2	0.0	0.8	1.1	1.0	0.6	1.0	1.0	0.0	7
M_6	0.0	0.7	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.3	2
M_7	0.0	0.5	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	1

Table 3.10: Example(I): Machine Cell Allocation

Cell 1:	M_1^1, M_2^1, M_3^1, M_4^1, M_4^2, M_5^1, M_5^2, M_7^1,
Cell 2:	M_2^2, M_3^2, M_4^3, M_5^3,M_5^4, M_6^1,
Cell 3:	M_2^3, M_2^4, M_5^5, M_5^6, M_5^7, M_6^2

Moreover, for a visual representation of the cells formed and the flow of parts consult Figure 3.4.

Table 3.11: Example(I): Part/Machine Assignment

Parts/M_i^k	1	2	3	4	5	6	7	8	9	10	$\sum_{j=1}^{10} UTIL_{i,j}$
M_1^1	0.0	0.3	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.5
M_2^1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
M_2^2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.6	0.7
M_2^3	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.5
M_2^4	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.9
M_3^1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.2
M_3^2	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.1	0.0	1.0
M_4^1	0.0	0.0	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	1.0
M_4^2	0.0	0.6	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	1.0
M_4^3	0.0	0.6	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	1.0
M_5^1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
M_5^2	0.0	0.2	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	1.0
M_5^3	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
M_5^4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
M_5^5	0.4	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.5
M_5^6	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
M_5^7	0.0	0.0	0.0	0.0	0.0	0.0	0.6	0.0	0.0	0.0	0.6
M_6^1	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0
M_6^2	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.0	0.8
M_7^1	0.0	0.5	0.0	0.2	0.0	0.2	0.0	0.0	0.0	0.0	0.9

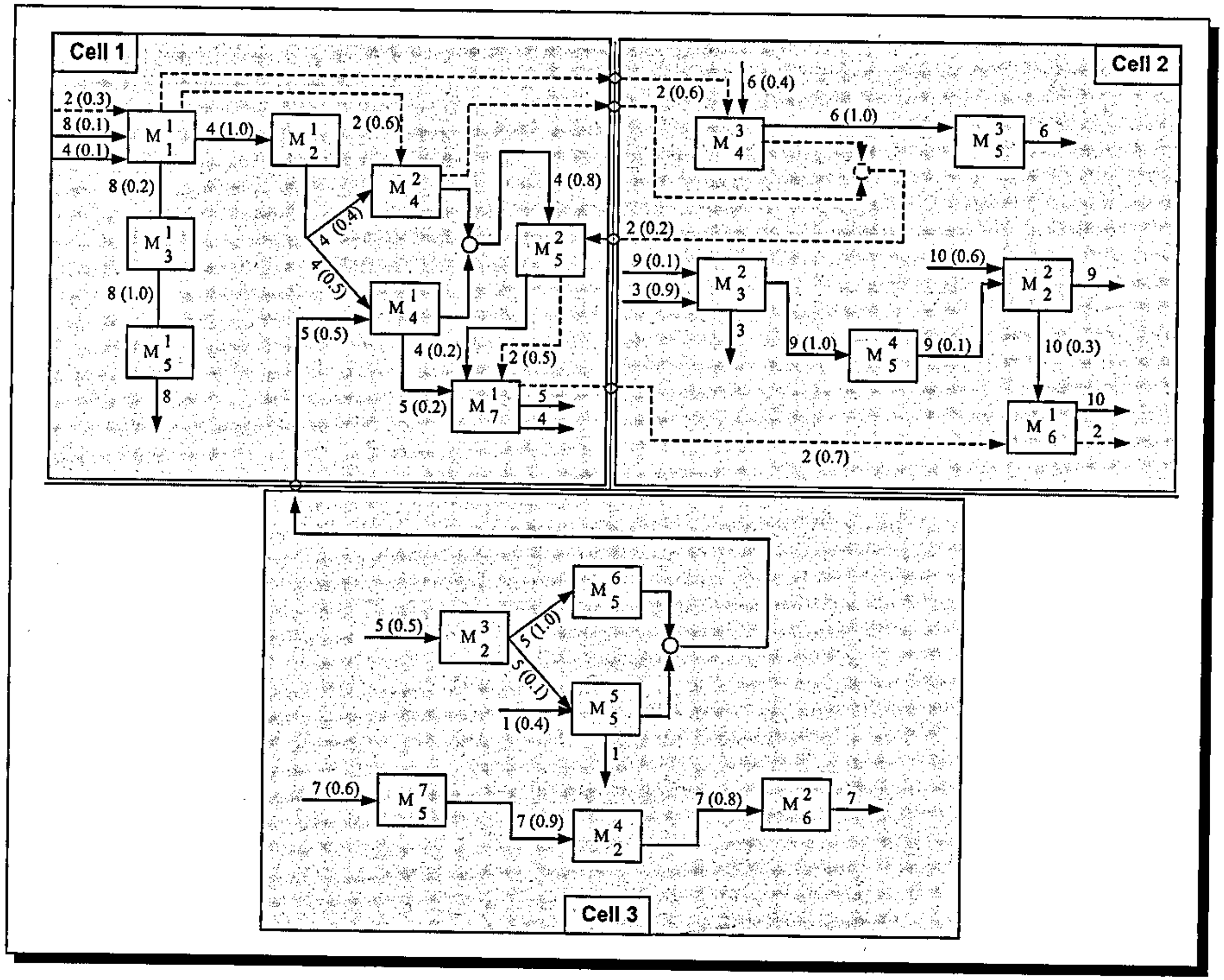


Figure 3.4: Example(I): Part/Machine Cell Allocation

According to Figure 3.4 the route that part two follows in order to be processed is represented by a dotted line. The machine sequence operation for part two as seen in Table 3.8 is: M_1, M_4, M_5, M_7, M_6. Part two starts getting processed in cell one and then moves into cell two; while the latter is in cell two the value of the 'extra' variable for $q = 2$, $i = 4$ is one. Because of that it is expected for part two to leave cell two and return for a later operation. Indeed, part two leaves cell two and moves into one and then returns to cell two for the final machine processing operation.

3.5 Summary

This chapter presented an enhanced mixed integer mathematical programming model for the CF problem. Initially the main operation for the model was the minimisation of the distinct allocation of parts to cells. Later the model's operation was enhanced by considering within the objective function the part/machine set-up costs and some additional constraints needed for the current task. Finally, a key constraint such as the part machine operation sequence was added together with an additional element minimising the revisits of parts to already visited cells. This final form of the mathematical programming model represents a more sophisticated CF problem which will be the foundation for the employment of other theories and methodologies, presented in the chapters to follow, incorporating realism to the system.

Chapter 4

Fuzzy Mathematical Programming for CF

4.1 Introduction

Applying mathematical programming problems to real world problems is a challenging task due to the following reasons:

- decision makers find it difficult to specify goals and
- parameters used in these models cannot be specified precisely.

Over the past twenty five years, fuzzy set theory, firstly developed by two pioneers Bellman and Zadeh [BZ70], has been applied to many disciplines, including operations research, control theory, artificial intelligent/expert systems, dealing with situations or problems involving ambiguities and fuzziness. Fuzzy Mathematical Programming (FMP) is one area in which fuzzy set theory has been applied extensively. For instance, FMP has been applied to problems regarding transportation [CKM84], location planning [DE90], project networks [Mje86], resource allocations [SP78], air pollution regulations [WZ78] and media selection for advertising [Dar87]. Although there has been an intense study of the application of fuzzy set theory to industrial engineering [EKW89], and operations management [KE86] very few studies have attempted to use FMP in the design of cellular manufacturing systems. This chapter will focus attention on applying fuzzy set theory to the mixed integer mathematical cell formation model described in Chapter 3.

4.2 Fuzzy Set Theory and Linear Programming

The term "fuzzy" was proposed by Zadeh in 1962 [Zad62]. In 1965, Zadeh [Zad65] formally published the famous paper "Fuzzy Sets". The fuzzy set theory [BZ70], was developed to improve the oversimplified model, thereby developing a more robust and flexible model in order to solve real-world complex systems involving human decision making.

Zadeh [Zad65] writes:

> "The notion of a fuzzy set provides a convenient point of departure for the construction of a conceptual framework which parallels in many respects the framework used in the case of ordinary sets, but is more general than the latter and, potentially, may prove to have a much wider scope of applicability, particularly in the fields of pattern classification and information processing. Essentially, such a framework provides a natural way of dealing with problems in which the source of imprecision is the absence of sharply defined criteria of class membership rather than the presence of random variables."

"Imprecision" here is meant in the sense of vagueness rather than the lack of knowledge about the value of a parameter. Fuzzy set theory provides a strict mathematical framework (there is nothing fuzzy about fuzzy set theory!) in which vague conceptual phenomena can be precisely and rigorously studied [Zim88a].

In fuzzy set theory constraints as well objectives are regarded as fuzzy sets. In general a fuzzy set is defined as follows [Zad65]:

If $X = \{x\}$ is a collection of objects denoted generically by x then a fuzzy set A in X is a set of ordered pairs:

$$A = \{(x, \mu_A(x)) | x \in X\} \tag{4.1}$$

$\mu_A(x)$ is called the membership function or grade of membership, also degree of truth, of x in A which maps X to the membership space M. The space of these values is again a fuzzy set, which is called "decision". But how can the decision space be determined for a linear programming model? According to Bellman and Zadeh [BZ70]:

Assume the existence of a fuzzy goal $\tilde{G}$ and a fuzzy constraint $\tilde{C}$ in a space of alternatives X. Then $\tilde{G}$ and $\tilde{C}$ combine to form a decision, $\tilde{D}$, which is a fuzzy set resulting from intersection of $\tilde{G}$ and $\tilde{C}$. In symbols, $\tilde{D}=\tilde{G} \cap \tilde{C}$, and correspondingly

$$\mu_{\tilde{D}}(x) = \{\mu_{\tilde{G}}(x) * \mu_{\tilde{C}}(x)\} \tag{4.2}$$

where '$*$' denotes an appropriate, possibly context dependent "aggregator" (connective operator[1]). Let M be the set of points $x \in X$ for which $\mu_{\tilde{D}}(x)$ attains its maximum, if it exists. Then M is called the maximising decision.

If $\mu_{\tilde{D}}(x)$ has a unique maximum at x_M, then the maximising decision is a uniquely defined crisp decision which can be interpreted as the action which belongs to all fuzzy sets representing either constraints or goals with the highest possible degree of membership.

For the classical form of a Linear Programming model where the decision made is decision under certainty and the decision space is defined by the constraints and the 'goal' representing the objective function is of the following classical form:

Table 4.1: Classical LP Model

minimise	$\mathbf{f(x) = c^T x}$
such that	$\mathbf{Ax \leq b}$
	$\mathbf{x \geq 0}$

In the case where goal and constraints are fuzzy the problem can be modelled as:

Table 4.2: Fuzzy LP Model

find x such that
$\mathbf{c^T x} \lesssim z$
$\mathbf{Ax} \lesssim b$
$\mathbf{x \geq 0}$

Here '$\lesssim$' denotes the fuzzified version of '$\leq$' and has the linguistic interpretation "essentially smaller than or equal".

From Table 4.2, objective and constraints are treated equally and there is no longer a difference between them. Therefore the relationship between constraints and objectives

[1]Commonly used operators applied in mathematical programming will be presented in next section.

in a fuzzy environment is fully *symmetric* [Zim91], [LH92a].

It is worth mentioning, that a model, where the objective function is crisp, that is, has to be maximised or minimised and in which the constraints are all or partially fuzzy, is no longer symmetrical.

4.2.1 Membership Functions and Fuzzy Operators

Although FMP is different from other fuzzy applications such as fuzzy ranking and fuzzy inference, two factors - membership functions and fuzzy operators - common to other fuzzy applications deserve attention.

Membership functions are used to incorporate fuzziness or to represent the linguistic variables for applications of fuzzy set theory. Many types of membership functions have been used in practice [LH92a], [Zim85], [Dom90], [LL91], [Leb81]. However, *linear non-increasing* [WZ78], [SP78], and *triangular* [YI91] are two of the most popular functions in use.

In order to manipulate fuzzy numbers in fuzzy sets, a variety of fuzzy operators extended from the traditional (crisp) mathematical programming have been proposed. For instance, let $\tilde{A}$ and $\tilde{B}$ be two fuzzy sets in X, the fuzzy sum $\tilde{C}=\tilde{A}+\tilde{B}$ is defined as:

$$\tilde{C} = \{x, \mu_{\tilde{A}+\tilde{B}}(x) | x \in X\}$$

where

$$\mu_{\tilde{A}+\tilde{B}}(x) = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \times \mu_{\tilde{B}}(x)$$

In FMP, fuzzy aggregation operators are used to transform FMP to traditional mathematical programming so that the FMP can be solved via traditional mathematical programming.

There are two basic classes of operators: operators for the intersection and the union of fuzzy sets - referred to as triangular norms (*t-norms*) and triangular conorms (*t-conorms*) respectively - and the class of *averaging operators* which model connectives for fuzzy sets between t-norms and t-conorms [Zim91].

One of the most commonly used operators [Zad65] applied for the intersection of fuzzy goals and constraints is the '*min*' operator. Although, this operator is very practical when applying it, it does not allow for compensation at all. Compensation, in the context of aggregating operators for fuzzy sets, could be interpreted as follows [Zim91]:

Given that a degree of membership to the aggregating fuzzy set is:

$$\mu_{\tilde{Agg}}(x_k) = f(\mu_{\tilde{A}}(x_k), \mu_{\tilde{B}}(x_k)) = k \tag{4.3}$$

f is compensatory if $\mu_{\tilde{Agg}}(x_k) = k$ is obtainable for a different $\mu_{\tilde{A}}(x_k)$ by a change in $\mu_{\tilde{B}}(x_k)$.

The 'min' operator together with some commonly used compensatory operators (most of which are averaging operators) are presented in Table 4.3.

Please note that the formulations for each of the aggregation operators presented in Table 4.3 have a generic form. For more details on the generic forms of some of the aggregation operators refer to Appendix B.

Table 4.3: Commonly Used Operators in FMP Studies

Operator	Formulation*	Compensatory†	Format after transformation	References
'Min'	$\mu_D = min \ \ \mu_S$††	No	Linear	[CKM84],[Mje86] [WZ78],[Dar87]
'Fuzzy and'($\tilde{and}$)	$\mu_D = \gamma min \ \mu_S + (1-\gamma)/(T+1) \times \sum_{S=0}^{T} \mu_S$	Positive	Linear	[Wer88]
'Min – bounded sum'	$\mu_D = \gamma min \ \mu_S + (1-\gamma) min(1, \sum_{S=0}^{T} \mu_S)$	Positive	Linear	[Luh82]
'Compensatory and'	$\mu_D = \gamma min \ \mu_S + (1-\gamma) max \ \mu_S$	Positive	Linear	[Zim88b]
'Product'	$\mu_D = \Pi_{S=1}^{T} \ \mu_S$	Negative	Nonlinear	[Zim78]
'γ'	$\mu_D = (\Pi_{S=0}^{T})^{1-\gamma} \ [1 - \Pi_{S=0}^{T}(1-\mu_S)]^{\gamma}$	Positive	Nonlinear	[Luh82], [Zim83]

* The objective is to maximise μ_D.

† The definitions for both positive and negative compensatory operators are adapted from [KLL93].

†† μ_S is membership function of the S^{th} fuzzy constraint (S is index of fuzzy constraints, $S = 0, ..., T$).

4.3 Fuzzy Cell Formation Models

In the aforementioned deterministic model, section 3.4.2, it is assumed that both the objective function and all related constraints can be defined precisely. In practice it is very difficult for the decision maker to specify the exact goals and constraints when modelling the problem. Tools for experimenting with changes in both coefficients and constraints by doing either sensitivity or postoptimality analysis are well established for linear programming (LP) models. For IP these tools are less well developed because the absence of continuity precludes the natural extension of these tools from LP to IP, however certain experimentation is still possible. In what follows the authors consider aspects of fuzzification within CF problems.

Although in the current model there are a lot of elements that could be fuzzy such as set-up costs and utilisation amounts, the author considers the fuzziness concept on the number of machines included in cells. The latter is chosen based upon the model's main operation which is the creation of cells with a specific number of machines in them. The resulting analysis can thus be considered as just one example of analysing fuzziness out of a range of possible elements that could be made fuzzy. Thus the analysis is to some extent exploratory and intended to show the possibilities of extending CF problems to incorporate more realism. The analysis will show ways in which a range of configurations can be offered to decision makers and this range could be extended by introducing fuzziness into other parameters, of which the number of machines in a cell is one.

In the deterministic model, equation (3.21), the maximum number of machines, E_{MAX}, allowed in each of the cells has been precisely specified. What will happen if the maximum number of machines varies between a range, thus is uncertain? The range assumed is defined by the *upper bound* of the maximum number of machines and the *lower bound* of the maximum number of machines. The following two fuzzy equations describe the situation where the maximum number of machines takes values between a range. Depending upon the *type of the membership function* used, either both equations (4.4), (4.5) are utilised or only one is employed as will be seen later.

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \overset{<}{\approx} v_q \times E_{MAX} \quad \forall\, q \tag{4.4}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \overset{>}{\approx} v_q \times E_{MAX} \quad \forall\, q \tag{4.5}$$

The objective function, equation (3.15), can also be assumed as fuzzy (see Table 4.2) and according to Werners [Wer87] and Lai and Hwang [LH92b] can be fuzzified as follows:

$$\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q} \times w_{j,q}) + \sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j} \times \sum_{q=1}^{NC} s_{i,j,q}) + \sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j \times extra_{q,j,i}) \lesssim D^0 = D^1 - P_0 \quad (4.6)$$

where D^0 is the feasible value of the best goal, which can be obtained by solving the traditional model (see section 3.4.2) with the total number of machines instances in the system. D^1 is the feasible value of the worst goal which can be obtained by solving the deterministic model with the minimum feasible number of machines. More details for the above parameter values are provided within section 4.4.

In order to transform the fuzzy model to its equivalent traditional formulation three tolerance values, P_0, P_{R1} and P_{R2} are used within the objective function (4.6) and constraints (4.4) and (4.5) respectively. The value for parameter P_0 can be determined as the value equal to D^0 subtracted from D^1, whereas the values of parameters P_{R1} and P_{R2} depend upon the decision maker's experience on the characteristics of the problem.

For the fuzzy incorporation within the current model two membership functions linear non-increasing [WZ78], [Zim91], and triangular [YI91] will be considered. For the transformation of the fuzzy formulation to MP formulation, fuzzy aggregation operators will be used. As already stated, Table 4.3 (page 68) summarizes a number of operators that have been applied before in fuzzy mathematical programming. The first three operators have linear forms after transformation whereas the last two are non-linear and thus more difficult to handle. Moreover, all operators except the 'min' classical operator allow some type of compensation; either a positive or negative [KLL93]. For example, the 'fuzzy and' operator [Wer88] combines the minimum and maximum operator with the arithmetic mean and allows compensation between the membership values of the aggregated sets leading to very good results with respect to empirical data [ZZ88]. For the current study 'min', 'fuzzy and' and 'min-bounded sum' operators will be examined. Last but not least, objectives and constraints are treated equally and there is no difference between them, therefore their relationship is fully *symmetric* [Zim91], [LH92a].

Before continuing with the model formulation, it is useful to have a visual presentation of the membership functions involved. The membership function for the objective function, currently added as a new constraint (4.6), is linear non-increasing and can be seen in Figure 4.1.

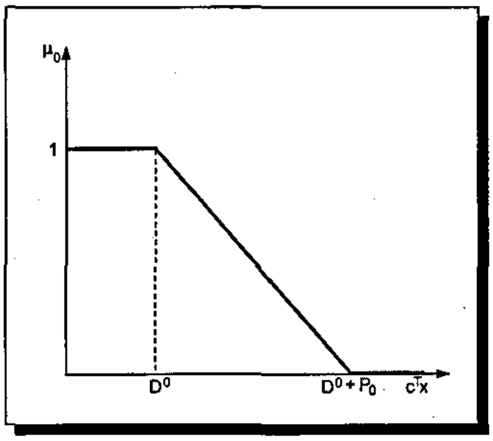


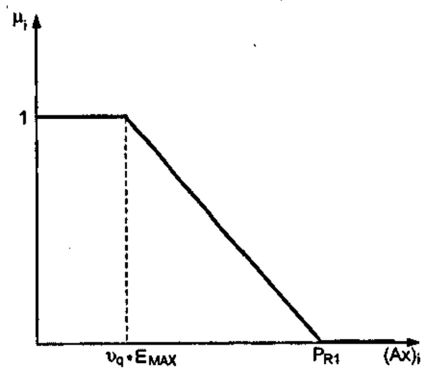
Figure 4.1: Membership Function for Objective Function

The mathematical presentation for the above membership function is as follows:

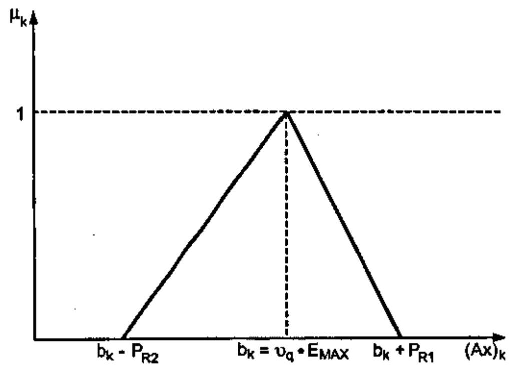
$$\mu_0(x) = \begin{cases} 1, & if\ c^Tx \leq D^0 \\ 1 - \frac{c^Tx - D^0}{P_0}, & if\ D^0 \leq c^Tx \leq D^0 + P_0 \\ 0, & if\ c^Tx > D^0 + P_0 \end{cases}$$

In the case where the only fuzzy constraint considered in the model is (4.4) (i.e. minimum number of maximum number of machines is equal to the minimum number of machines as rigidly defined in constraint (3.22)), the membership function is linear non-increasing and can be seen in Figure 4.2(a). However, when both fuzzy constraints, (4.4) and (4.5), are involved the membership function is triangular and presented in Figure 4.2(b). Parameter P_{R2} is involved as the 'negative' tolerance value for maximum number of machines and P_{R1} is involved as the 'positive' tolerance value as the frame of reference for both tolerance values is the crisp value of maximum number of machines. In other words, the maximum number of machines is the intermediate value between the lower bound of maximum number of machines determined by the tolerance value P_{R2}, and the upper bound of the maximum number of machines determined by the tolerance value P_{R1}.

Mathematical formulations for the membership functions, as presented in Figures 4.2(a) and 4.2(b) respectively, are shown next.



(a) Membership function for fuzzy constraint (4.4)



(b) Membership function for fuzzy constraints (4.4) and (4.5)

Figure 4.2: Membership Functions related to Constraints

$$\mu_i = \begin{cases} 1, & if \;\; y_{i,k,q} \leq v_q \times E_{MAX} \\ 1 - \frac{y_{i,k,q} - v_q \times E_{MAX}}{P_{R1}}, & if \;\; v_q \times E_{MAX} \leq y_{i,k,q} \leq v_q \times E_{MAX} + P_{R1} \\ 0, & if \;\; y_{i,k,q} > v_q \times E_{MAX} + P_{R1} \end{cases}$$

$$\mu_k = \begin{cases} 0, & if \;\; y_{i,k,q} \geq v_q \times E_{MAX} + P_{R1} \\ 1 - \frac{y_{i,k,q} - v_q \times E_{MAX}}{P_{R1}}, & if \;\; v_q \times E_{MAX} \leq y_{i,k,q} \leq v_q \times E_{MAX} + P_{R1} \\ 1 - \frac{v_q \times E_{MAX} - y_{i,k,q}}{P_{R2}}, & if \;\; v_q \times E_{MAX} - P_{R2} \leq y_{i,k,q} \leq v_q \times E_{MAX} \\ 1, & if \;\; y_{i,k,q} = v_q \times E_{MAX} \\ 0, & if \;\; y_{i,k,q} \leq \times v_q \times E_{MAX} - P_{R2} \end{cases}$$

The next task is to present the way that each of these membership functions are utilised within the CF problem when different aggregation operators are employed.

4.3.1 'Min' Aggregation Operator

For 'min' operator one additional variable is needed and specified below:

λ : minimum value of all membership functions

• Linear Non-Increasing Membership Function

The equivalent formulation can be obtained as follows:

$$Max \;\; \lambda \tag{4.7}$$

subject to

$$\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q} \times w_{j,q}) + \sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUNP_{i,j} \times \sum_{q=1}^{NC} s_{i,j,q}) + \sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j \times extra_{q,j,i}) + \lambda P_0 \leq D^0 + P_0 \tag{4.8}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} + \lambda P_{R1} \leq v_q \times E_{MAX} + P_{R1} \;\; \forall\, q \tag{4.9}$$

$$0 \leq \lambda \leq 1 \tag{4.10}$$

Besides those equations noted above, equations (3.16)-(3.20) and (3.22)-(3.29) are added here as well.

From equation (4.9), the number of machines allowed in a cell ranges from E_{MIN} (crisply defined by the decision maker as in the traditional model) to $v_q \times E_{MAX} + P_{R1}$.

• Triangular Membership Function

The equivalent MP formulation consists of equations (4.7) to (4.10) and equation (4.11) below.

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} - \lambda P_{R2} \geq v_q \times E_{MAX} - R_{R2} \;\; \forall\, q \tag{4.11}$$

It also includes crisp constraints (3.16)-(3.20) and (3.23)-(3.29), defined in Chapter 3.

4.3.2 'Fuzzy and' ('$\tilde{and}$') Aggregation Operator

For the MP formulation using the '$\tilde{and}$' operator new index, parameter and new decision variables are needed.

l	membership functions index	$l = 0, \ldots, MF$ [2]
γ	parameter for fuzzy modelling	
a_l	extra variable used for '$\tilde{and}$' operator	
ω	extra variable used for '$\tilde{and}$' operator	
c	total number of fuzzy constraints	

• Linear Non-Increasing Membership Function

The equivalent LP formulation is as follows:

$$Max \;\; \omega + (1-\gamma) \times \frac{1}{c+1} \sum_{l=0}^{MF} a_l \tag{4.12}$$

subject to

$$\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q} \times w_{j,q}) + \sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j} \times \sum_{q=1}^{NC} s_{i,j,q}) + \sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j \times extra_{q,j,i}) + \omega \times P_0 + a_0 \times P_0 \leq D^0 + P_0 \tag{4.13}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} + \omega \times P_{R1} + a_1 \times P_{R1} \leq v_q \times E_{MAX} + P_{R1} \quad \forall\, q \tag{4.14}$$

$$\omega + a_l \leq 1 \tag{4.15}$$

$$\omega \leq 1 \tag{4.16}$$

$$a_l \geq 0, \;\; \gamma < 1 \tag{4.17}$$

Also crisp constraints (3.16)-(3.20) and (3.22)-(3.29) are included in this model as well.

[2]Please note, that in the case where a linear non-increasing membership function is used two membership functions are considered, whereas for the triangular case the number of membership functions taken into account is three.

- **Triangular Membership Function**

The '$\tilde{and}$' operator and triangular membership function will now be considered. The objective function for this formulation is the same as equation (4.12), and only one new constraint is added as follows:

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} - \omega \times P_{R2} - a_2 \times P_{R2} \geq v_q \times E_{MAX} - R_{R2} \quad \forall\, q \qquad (4.18)$$

Also constraints (4.13)-(4.17), (3.16)-(3.20) and (3.23)-(3.29) are included when triangular membership function is employed.

4.3.3 'Min-bounded-sum' Aggregation Operator

For the MP formulation the γ parameter and the ω decision variable used for the 'fuzzy and' operator are preserved here as well. Additionally, an extra index and a new decision variable are needed and defined as follows:

t	additional cell index	$t = 1,\dots,NC$
u_t	extra variable used for 'min-bounded sum' operator	

- **Linear Non-Increasing Membership Function**

The equivalent LP formulation is as follows:

$$Max \;\; \gamma \times \omega + (1-\gamma)\sum_{t=1}^{NC} u_t \qquad (4.19)$$

subject to

$$\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q} \times w_{j,q}) + \sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j} \times \sum_{q=1}^{NC} s_{i,j,q}) + \sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j \times extra_{q,j,i}) + \omega \times P_0 \leq D^0 + P_0 \qquad (4.20)$$

$$u_t \leq [\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q} \times w_{j,q}) + \sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j} \times \sum_{q=1}^{NC} s_{i,j,q}) + \sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j \times extra_{q,j,i}) - D^0]/P_0 + [\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,t} - v_t \times E_{MAX}]/R_{R1} \quad \forall\, t \qquad (4.21)$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} + \omega \times P_{R1} \leq v_q \times E_{MAX} + P_{R1} \;\; \forall\, q \tag{4.22}$$

$$u_t \leq 1 \;\; \forall\, t \tag{4.23}$$

$$0 \leq \omega \leq 1 \tag{4.24}$$

$$\gamma < 1 \tag{4.25}$$

Also equations (3.16)-(3.20) and (3.22)-(3.29) are included in the above formulation.

- **Triangular Membership Function**

For this case one more membership function is included, therefore equation (4.21) is differently formulated producing equation (4.26)) and constraint (4.27) is also added.

$$\begin{aligned} u_t \leq [&\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q} \times w_{j,q}) + \sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUNP_{i,j} \times \sum_{q=1}^{NC} s_{i,j,q}) + \\ &\sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j \times extra_{q,j,i}) - D^0]/P_0 + [\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,t} - v_t \times E_{MAX}]/R_{R1} + \\ &[v_t \times E_{MAX} - \sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,t}]/R_{R2} \;\; \forall\, t \end{aligned} \tag{4.26}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} - \omega \times P_{R2} \geq v_q \times E_{MAX} - R_{R2} \;\; \forall\, q \tag{4.27}$$

Objective function (4.19) and constraints (4.20), (4.22)-(4.25) are preserved and added here. Also equations (3.16)-(3.20) and (3.23)-(3.29) are included in this formulation.

4.3.4 Modification of the Traditional Crisp Model

In all fuzzy models considered in previous sections the objective function has changed its status compared with the traditional model (described in section 3.4). When fuzzy aggregation operators and membership functions are taken into account the objective function changes its status and its set to maximise.

Although this change does not affect the plant functioning, it leads to a computational operation increase of the branch and bound algorithm and production of some faulty results when assigning machines to cells for a specific part operation. Hence, in order to obtain some reasonably good results, another constraint is added in the traditional model. The new constraint has the following form:

$$s_{i,j,q} \leq \sum_{k=1}^{KM_i} y_{i,k,q} \quad \forall\ j, i, q \tag{4.28}$$

Constraint (4.28) simply forces the machine(s) of type i used by part j in cell q to be always less than total k number of machines of type i assigned in cell q.

It is worth mentioning that this crisp constraint is added now to the six fuzzy models: 'min' operator with linear and triangular membership function, 'fuzzy and' operator with linear and triangular membership functions and 'min-bounded sum' operator with linear and triangular membership functions.

4.4 Models Assessment

To assess the results of each of the aggregation operators within the CF model six data sets are used. The first data set (DS 1) was created by adopting numerical values from Foulds *et al.* [FFW06] and randomly generated some more needed to meet the requirements of the current model. The numerical values of DS 1 are presented in Table 4.4. The remaining data sets were randomly generated by a computer program developed with MatLab (see Appendix C). More discussion of this program will be presented later in Chapter 6. The sets were chosen in order to provide variety of parameter values in the models.

Data sets two (DS 2) and three (DS 3) have the same size with data set one (DS 1) as they employ the same number of parts and machine types but they differ from each other as their parameters values are randomly generated. A similar case applies for the remainder of the data sets, i.e. data sets four to six, where all consist of nine parts and machine types but their parameter values like part/machine set-up costs and part/machine utilisation are different. The latter parameter determines the total number of machine instances. The greater the total number of machines in the system the more intense the problem as will be seen when examining the model's performance.

Table 4.4: Numerical Values for DS 1

Part/Machines utilisation & Number of machine instances											
P_j/M_i	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	KM_i
M_1	0.0	0.3	0.0	0.1	0.0	0.0	0.0	0.1	0.0	0.0	1
M_2	0.0	0.0	0.0	1.0	0.5	0.0	0.9	0.0	0.1	0.6	4
M_3	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.2	0.1	0.0	2
M_4	0.0	1.2	0.0	0.9	0.5	0.4	0.0	0.0	0.0	0.0	3
M_5	0.4	0.2	0.0	0.8	1.1	1.0	0.6	1.0	1.0	0.0	7
M_6	0.0	0.7	0.0	0.0	0.0	0.0	0.8	0.0	0.0	0.3	2
M_7	0.0	0.0	0.0	0.2	0.2	0.0	0.0	0.0	0.0	0.0	1
Set-up Costs											
P_j/M_i	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	
M_1	0.00	2.95	0.00	1.96	0.00	0.00	0.00	1.92	0.00	0.00	
M_2	0.00	0.00	0.00	5.12	2.42	0.00	5.05	0.00	1.54	2.16	
M_3	0.00	0.00	2.93	0.00	0.00	0.00	0.00	1.94	1.45	0.00	
M_4	0.00	5.54	0.00	2.91	2.42	2.38	0.00	0.00	0.00	0.00	
M_5	2.91	2.59	0.00	2.88	5.42	4.91	2.63	4.93	4.81	0.00	
M_6	0.00	2.82	0.00	0.00	0.00	0.00	2.73	0.00	0.00	2.49	
M_7	0.00	0.00	0.00	2.12	2.12	0.00	0.00	0.00	0.00	0.00	
Part-Machine operation Sequence & ZTYPES											
P_j/Seq.	1	2	3	4	5						ZTYPES
P_1	M_5										1
P_2	M_1	M_4	M_5	M_6							4
P_3	M_3										1
P_4	M_1	M_2	M_4	M_5	M_7						5
P_5	M_2	M_5	M_4	M_7							4
P_6	M_4	M_5									2
P_7	M_5	M_2	M_6								3
P_8	M_1	M_3	M_5								3
P_9	M_3	M_5	M_2								3
P_{10}	M_2	M_6									2
$E_{MIN} = 3$, $E_{MAX} = 6$ & NCELLS=5											

Lastly the maximum number of machines allowed in a cell takes different values depending upon the total number of machine instances in each of the data sets and the number of cells allowed to be created. When fuzzy models are used, fuzzy intervals are used to define those numbers; upper bound of maximum number of machines and lower bound of maximum number of machines allowed in a system is the interval specified when a triangular membership function is used. Sometimes depending upon the size of the problem, the value of the lower bound of maximum number of machines becomes equal to the minimum number of machines rigidly defined by the decision maker. For the linear non-increasing membership function the upper bound of the maximum number of machines is also specified. Both for linear non-increasing and triangular cases the minimum number of machines is explicitly required to be equal to three in order to prevent the assignment of fewer than three machines in each cell formed.

It is worth commenting on the tolerance value of an objective function and how this can be determined via an example. For example, for DS 1 the tolerance value P_0 for the objective function is assigned the value 32.91. As stated already, this value is produced by subtracting D^0 from D^1. D^0 in this case is equal to 207.01 (objective value of the traditional model, where a maximum number of machines is assumed) and D^1 is equal to 239.92 (where a minimum number is used). More specifically, for defining D^0 value the maximum number of machines used is assumed to be equal to the total number of machine instances in the system. On the other hand for determining the D^1 parameter the minimum number of machines used is required to be equal to the least number of machines which when multiplied with the default number of cells will be equal or greater to the total number of machines in the system so that the system won't be infeasible. For example, if the total number of machines is 20 and the number of cells is 5 then the minimum number of machines will be at least 4; otherwise the problem will turn out to be infeasible.

4.4.1 Performance of Models

All models were solved by running `Xpress-MP` on a Linux machine (Intel (P4 Xeon) 3GHz, 1.00 GB of RAM) and accessed remotely. Clustering performances were measured in terms of number of cells created, distinct cells used by each part, number of times a part re-visits a cell for later machine operation, CPU time and total cost of dealing with intercell movements and set-up costs.

Table 4.5 (page 80) summarises the computational results for all six data sets. For each data set, six cases for the fuzzy models (three operators and two membership functions), plus two cases for the proposed deterministic model are examined. The final case (case 8) where the deterministic model is utilised, examines a possible impact on the solution quality when the maximum number of machines is relaxed having the value of the specified upper bound.

Table 4.5: Computational Results for Fuzzy and Crisp Models

DS 1: Problem size $(NM \times NP) = (7 \times 10)$; NCELLS=5; $E_{MAX} = 6$; $P_0 = 32.91$; $P_{R1} = 2$; $P_{R2} = 3$, Machine Instances= 20

Case	Operator	Membership Functions	Cells Created	Distinct Cells Used by Each Part	Later Revisits of Parts	CPU Time (secs)	Total Cost
1	**'Min'**	Linear non-increasing	4	12	0	2010	211.47
2	**'Min'**	Triangular	3	12	1	32	219.95
3	**'$\widetilde{\text{and}}$'**	Linear non-increasing	4	12	0	40	217.01
4	**'$\widetilde{\text{and}}$'**	Triangular	3	12	1	38	212.47
5	**'Min – bounded sum'**	Linear non-increasing	3	14	0	10	231.47
6	**'Min – bounded sum'**	Triangular	3	14	0	9	231.47
7	Deterministic Model	$E_{MAX} = 6$	4	12	0	109	219.92
8	Deterministic Model	$E_{MAX} = 8$	4	12	0	32	210.92

DS 2: Problem size $(NM \times NP) = (7 \times 10)$; NCELLS=5; $E_{MAX} = 6$; $P_0 = 33$; $P_{R1} = 2$; $P_{R2} = 3$; Machine Instances= 20

Case	Operator	Membership Functions	Cells Created	Distinct Cells Used by Each Part	Later Revisits of Parts	CPU Time (secs)	Total Cost
1	**'Min'**	Linear non-increasing	3	13	0	10875	229.24
2	**'Min'**	Triangular	3	13	0	1149	229.24
3	**'$\widetilde{\text{and}}$'**	Linear non-increasing	4	13	2	5913	225.82
4	**'$\widetilde{\text{and}}$'**	Triangular	3	13	0	996	229.24
5	**'Min – bounded sum'**	Linear non-increasing	3	14	3	35	236.82
6	**'Min – bounded sum'**	Triangular	3	14	3	21	236.82
7	Deterministic Model	$E_{MAX} = 6$	4	13	2	4554	225.82
8	Deterministic Model	$E_{MAX} = 8$	3	12	1	433	220.24

... continues on next page

Table 4.5 – continues from previous page

DS 3: Problem size $(NM \times NP) = (7 \times 10)$; NCELLS=5; $E_{MAX} = 8$; $P_0 = 21$; $P_{R1} = 2$; $P_{R2} = 5$; Machine Instances= 27

Case	Operator	Membership Functions	Cells Created	Distinct Cells Used by Each Part	Later Revisits of Parts	CPU Time (secs)	Total Cost
1	**'Min'**	Linear non-increasing	-	-	-	$>$ 50 hours	-
2	**'Min'**	Triangular	3	12	1	6966	246.91
3	**'añd'**	Linear non-increasing	-	-	-	$>$ 50 hours	-
4	**'añd'**	Triangular	3	12	1	10431	246.91
5	**'Min – bounded sum'**	Linear non-increasing	3	13	1	17	251.49
6	**'Min – bounded sum'**	Triangular	3	13	1	51	267.75
7	Deterministic Model	$E_{MAX} = 8$	4	12	2	16499	264.14
8	Deterministic Model	$E_{MAX} = 10$	3	12	0	8341	262.14

DS 4: Problem size $(NM \times NP) = (9 \times 9)$; NCELLS=5; $E_{MAX} = 6$; $P_0 = 32.41$; $P_{R1} = 2$; $P_{R2} = 3$; Machine Instances= 22

Case	Operator	Membership Functions	Cells Created	Distinct Cells Used by Each Part	Later Revisits of Parts	CPU Time (secs)	Total Cost
1	**'Min'**	Linear non-increasing	-	-	-	$>$ 50 hours	-
2	**'Min'**	Triangular	3	11	2	4576	195.57
3	**'añd'**	Linear non-increasing	4	11	0	7686	189.19
4	**'añd'**	Triangular	4	11	0	578	187.78
5	**'Min – bounded sum'**	Linear non-increasing	3	12	1	33	200.19
6	**'Min – bounded sum'**	Triangular	3	12	1	11	200.19
7	Deterministic Model	$E_{MAX} = 6$	4	11	0	1214	187.78
8	Deterministic Model	$E_{MAX} = 8$	4	10	0	196	177.78

Table 4.5 – continues from previous page

DS 5: Problem size $(NM \times NP) = (9 \times 9)$; NCELLS=5; $E_{MAX} = 8$; $P_0 = 35$; $P_{R1} = 4$; $P_{R2} = 5$; Machine Instances= 28

Case	Operator	Membership Functions	Cells Created	Distinct Cells Used by Each Part	Later Revisits of Parts	CPU Time (secs)	Total Cost
1	**'Min'**	Linear non-increasing	-	-	-	> 50 hours	-
2	**'Min'**	Triangular	3	11	0	25448	243.27
3	**'añd'**	Linear non-increasing	4	11	1	114171	249.69
4	**'añd'**	Triangular	3	11	0	2752	243.27
5	**'Min – bounded sum'**	Linear non-increasing	3	12	5	9	263.69
6	**'Min – bounded sum'**	Triangular	3	12	5	49	263.69
7	Deterministic Model	$E_{MAX} = 8$	4	11	1	48974	257.99
8	Deterministic Model	$E_{MAX} = 12$	3	10	1	688	245.11

DS 6: Problem size $(NM \times NP) = (9 \times 9)$; NCELLS=5; $E_{MAX} = 9$; $P_{R1} = 5$; $P_{R2} = 6$; Machine Instances= 35

Case	Operator	Membership Functions	Cells Created	Distinct Cells Used by Each Part	Later Revisits of Parts	CPU Time	Total Cost
1	**'Min'**	Linear non-increasing	-	-	-	-	-
2	**'Min'**	Triangular	-	-	-	-	-
3	**'añd'**	Linear non-increasing	-	-	-	-	-
4	**'añd'**	Triangular	-	-	-	-	-
5	**'Min – bounded sum'**	Linear non-increasing	-	-	-	-	-
6	**'Min – bounded sum'**	Triangular	-	-	-	-	-
7	Deterministic Model	$E_{MAX} = 9$	-	-	-	> 50 hours	-
8	Deterministic Model	$E_{MAX} = 14$	3	11	1	40614 secs	259.08

It is observed that the CPU times for both 'fuzzy and' and 'min-bounded sum' operators vary but they differ marginally, whereas 'min' operator requires more execution time. However, the clustering results for all operators are different and this can be verified by the number of cells created, the distinct number of cells used by each part and the number of later revisits of parts to an already visited cell. The performance of 'min' operator is not promising especially when a linear non-increasing membership function is employed as it requires the longest time to process (especially when problem size increases i.e data sets 3, 4, and 5), and the clustering results are not better than those produced from the other two operators. The performance of the latter slightly improves when a triangular membership function is assumed. Overall and as can be seen from the table the computation time for each problem is very long for the 'min' operator and for some of the data sets the algorithm fails to converge after excessive computational time.

The 'fuzzy and' operator arrives at acceptable clustering results and the required CPU time is quite low for either non-increasing or triangular membership functions when smaller data sets are utilised. It is worth noting that the 'min-bounded sum' operator results in significant CPU time reduction when larger data sets are used (i.e. DS 3, DS 5) compared to the other two operators but it has two weaknesses: a) it is time consuming to obtain one of its constraints, in which all the membership functions must be summed up and, b) clustering results seem to be affected because all the membership functions of the constraints are added within one constraint when formulation takes place.

It is particularly encouraging that on the data sets considered, for the linear non-increasing version of the 'min-bounded sum' operator only small amounts of CPU time are required and these are substantially lower than those required for the deterministic versions of the problem. Constraints (4.20), (4.21) may be helping to reduce the integrality gap and aid convergence.

Moreover, for both 'fuzzy and' and 'min-bounded sum' operators experiments were carried out in order to determine a proper γ value (γ parameter can take values between the range [0.0, 0.9]). Both Tables 4.6 and 4.7 (pages 84 and 85) summarise the results of using DS 1 for the 'fuzzy and' and 'min-bounded sum' operators respectively.

The best γ value for case 3 is 0.1 whereas, the best γ value for case 4 is 0.9. For both cases the best γ is determined from the executing CPU time and not from the total cost found. The reason for this is that a small variation is observed in the total cost found mainly because of the machine set-up costs configuration. Moreover, according to Table 4.7 the best γ value for cases 5 and 6 are 0.4 and 0.5 respectively. For determining the remaining best γ values a similar process was followed.

Table 4.6: Results of varying γ Values for the 'fuzzy and' Operator

Case 3: Linear non-increasing membership function		
γ	CPU Time (secs)	Total Cost
0.1	**40**	217.01
0.2	131	228.01
0.3	84	227.01
0.4	80	227.01
0.5	150	228.01
0.6	42	211.47
0.7	120	222.47
0.8	73	222.59
0.9	95	221.47

Case 4: Triangular membership function		
γ	CPU Time	Total Cost
0.1	94	222.47
0.2	198	221.59
0.3	179	211.59
0.4	142	211.47
0.5	186	211.59
0.6	164	221.59
0.7	202	211.47
0.8	185	211.47
0.9	**38**	211.47

For considering the combinatorial explosion on the CPU times of all operators used a comparison between DS 1 (the smallest data set recorded in Table 4.5) and DS 5 (a larger data set) will now be made. As the size of the problem increases computation intensifies especially for the 'min' operator where no compensation between the membership functions of the aggregated sets is performed. For the 'fuzzy and' operator the CPU time increases significantly as the size of the problem becomes larger especially when a linear non-increasing membership function is employed. For the 'min-bounded sum' operator things are different as CPU time is kept low regardless of how big the problem is. Although CPU times are very promising for the latter, clustering results are not so good (see DS 5) compared with the 'fuzzy and' operator. Therefore, the

Table 4.7: Results of varying γ Values for the 'min-bounded-sum' Operator

Case 5: Linear non-increasing membership function		
γ	CPU Time (secs)	Total Cost
0.1	97	237.01
0.2	23	237.01
0.3	28	231.47
0.4	**10**	231.47
0.5	12	231.89
0.6	202	231.59
0.7	549	231.47
0.8	325	237.01
0.9	135	231.47

Case 6: Triangular membership function		
γ	CPU Time (secs)	Total Cost
0.1	18	237.01
0.2	27	231.47
0.3	63	231.47
0.4	19	237.01
0.5	**9**	231.47
0.6	55	231.47
0.7	103	237.01
0.8	77	231.47
0.9	47	212.47

'fuzzy and' operator, when a triangular membership function is employed, can be characterised as a good operator with promising clustering results and reasonable CPU times even when data sets become larger.

For cases 7 and 8, the deterministic model is utilised and it can be observed that more effort with trial and error is needed before an appropriate maximum number of machines allowed in a cell is decided to obtain a good solution. Also as the size of the problem increases (see DS 3 or DS 5) the CPU time increases significantly. However, employing a good operator (in terms of clustering results and CPU time) the use of the fuzzy model is quite straightforward. It can flexibly adjust the number of machines, given a tolerance value, thus avoiding time consuming trial and error.

Last but not least, it is worth commenting on DS 6 which is the largest of all the data sets used for this study. As can be seen from Table 4.5 the optimum value could only be found for case 8. All fuzzy models could not be tested since it was impossible to determine the tolerance value for the objective function. More specifically, finding D^1

it was not feasible because of the excessive amount of computational time. Thus, DS 6 gives an indication and also puts a limit to the size of the data sets that could be solved both for the deterministic and fuzzy models.

4.4.2 Concluding Remarks

From all the computational analyses the following conclusions can be drawn:

- It is a difficult and a very important issue for the decision maker to choose the appropriate number of machines allowed in a cell. If the size of the problem is small then there are some chances with trial and error that a good solution may be obtained. However, once the problem size increases, trial and error is not effective. Thus, fuzzy mathematical programming is a more promising alternative methodology.
- The fuzzy mathematical programming provides a more flexible way of representing the problem and it leads in most cases to better clustering results, especially when the 'fuzzy and' operator was employed. The CPU time depends upon the operator used. Although the 'min' operator is the most frequently used method, it did not perform well especially when a linear non-increasing membership function was used. The 'min-bounded sum' operator shortened the CPU significantly and outperformed the rest of the operators as well as the deterministic model, especially when a bigger problem was considered.
- The time advantage of using a fuzzy model rather than a deterministic model becomes significant once a larger scale model is used and the tolerance value of the constraints becomes bigger.
- The triangular membership function performed better than the linear non-increasing membership function for the CF problem and it seems to be more appropriate for modelling the constraints when a number of machines, either maximum or minimum are involved.
- Last but not least, although the fuzzy models have a lot of advantages towards the deterministic model they still have a major limitation. More specifically, as the sizes of the data sets increase computation intensifies and it is difficult for the decision maker to find solutions when attempting for example to determine the tolerance value of the objective function which is included within all fuzzy models. It might be possible to solve some larger sized problems, but ultimately a limit will be reached when computation times become excessive. For this reason the data sets chosen for the current study had to be of smaller sizes.

4.5 Summary

This chapter has introduced the use of fuzzy set theory where fuzzy aggregation operators and membership functions were employed to measure the uncertainty involved for determining the maximum number of machines allowed in each cell in a CF system. A number of fuzzy mixed integer programming models were formulated, each corresponding to the utilisation of a single operator and the membership function involved at each time. Based upon established theory on the aggregation operators and membership functions all models where defuzzified in order to be solvable by `XPRESS-MP`. Finally, all models including also the traditional were tested with a number of data sets and results compared.

Chapter 5

Heuristic Approaches for CF - Initial Solution

5.1 Introduction

The cell formation problem is a combinatorial optimisation problem that is NP-hard [BFHS88] and therefore an optimisation model (e.g. either a mixed integer mathematical programming model as described in Chapter 3 or a fuzzy mixed integer mathematical formulation as presented in Chapter 4) will yield a globally optimal solution or suboptimal solution in a very large computation time or no solution when a large scale data set is to be considered. The inability to address greater instances properly is due to the use of many variables and the significant number of constraints considered in *one step* when traditional mathematical programming software is employed (i.e. XPRESS-MP) aiming to group machines into cells and assign parts to machine cells simultaneously when both ordered part machine operation sequence and machine set-up costs are taken into account. To gain more benefits from the cell formation, heuristic approaches will be developed and presented in this chapter. More specifically, a three stage approach will be proposed underlying the system requirements and effectively represent the CF mathematical model (see section 3.4). This three stage approach will form the foundation of an initial starting solution to be fed into more advanced search strategies developed in later chapters.

In areas with practical applications such as industrial engineering, heuristics is an analytical method i.e. an algorithm for problem solving. It is originally derived from the Greek verb "heurísko" (ευρίσχω), which means "to search". Although heuristic algorithms are not guaranteed to provide optimal solutions (usually sub-optimal results are obtained), they are very useful in producing an acceptable solution in reasonable time. In fact, optimal results can only be obtained under very restricted conditions as mathematical programming indicates and this makes heuristic approaches more practical in real life applications.

5.2 A Heuristic in the Presence of Part Machine Sequence

In the present work, a three stage approach for the CF problem is proposed based upon the CF model requirements. In the first stage, a preliminary allocation of machines to cells is introduced which forms an initial phase and basic input for later stages. In the second stage key restrictions together with special input parameters of the above model form the foundation of building a new approach for allocating parts to machine-cells. The last stage involves the evaluation of both the value of the objective function involved and its solution.

The model adopted for heuristic development is the complete formulation of Chapter 3. For reference reasons within this Chapter the complete model is provided below as well.

$$Min\ (\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q}\times w_{j,q})+\sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j}\times\sum_{q=1}^{NC}s_{i,j,q})+\sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{i=1}^{NM}(A_j\times extra_{q,j,i})) \tag{5.1}$$

subject to

$$\sum_{q=1}^{NC} y_{i,k,q} = 1 \quad \forall\ i,k \tag{5.2}$$

$$\sum_{q=1}^{C} x_{i,j,q} = UTIL_{i,j} \quad \forall\ i,j \tag{5.3}$$

$$x_{i,j,q} \leq s_{i,j,q} \quad \forall\ i,j,q \tag{5.4}$$

$$x_{i,j,q} \geq UTIL_{MIN} \times s_{i,j,q} \quad \forall\ i,j,q \tag{5.5}$$

$$\sum_{j=1}^{NP} x_{i,j,q} \leq \sum_{k=1}^{KM_i} y_{i,k,q} \quad \forall\ i,q \tag{5.6}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \leq v_q \times E_{MAX} \quad \forall\, q \tag{5.7}$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \geq v_q \times E_{MIN} \quad \forall\, q \tag{5.8}$$

$$v_{q+1} \leq v_q \quad \forall\, q \tag{5.9}$$

$$\sum_{q=1}^{NC} q \times y_{i,k,q} \leq \sum_{q=1}^{NC} q \times y_{i,k+1,q} \quad \forall\, i,k \tag{5.10}$$

$$x_{i,j,q} \leq UTIL_{i,j} \times w_{j,q} \quad \forall\, i,j,q \tag{5.11}$$

$$x_{i,j,q} \leq UTIL_{MAX} \times xx_{i,j,q} \quad \forall\, i,j,q \tag{5.12}$$

$$x_{i,j,q} \geq UTIL_{MIN} \times xx_{i,j,q} \quad \forall\, i,j,q \tag{5.13}$$

$$xx_{L_{j,z},j,q} + xx_{L_{j,r},j,q} - \sum_{zz=z+1}^{r-1} xx_{L_{j,zz},j,q} \leq extra_{q,j,L_{j,z}} + 1 \quad \forall\, q,j,z,r \tag{5.14}$$

$$y_{i,k,q},\ v_q,\ w_{j,q},\ extra_{q,j,i},\ xx_{i,j,q} = 0 \text{ or } 1; \quad 0 \leq x_{i,j,q}; \quad s_{i,j,q} \ \textit{integer} \ \forall\, i,k,j,q \tag{5.15}$$

Moreover, part of the notation defined in Chapter 3 will also be used for the current analysis, together with some extra features as follows:

$CELL_{MIN}$	minimum number of cells to be formed
$CELL_{MAX}$	maximum number of cells to be formed
TMI	total number of machine instances in the system

It is worth noting that the heuristic algorithms were developed and solved in `MatLab`(TM)[1]. `MatLab` is a high-performance language for technical computing. It integrates computation and programming in an easy-to-use environment where problems and solutions are expressed in a familiar mathematical notation.

The CF model is a rather complex problem where a significant number of indices is used to address all the input elements. In an integer programming environment it is

[1]`MatLab` is a trademark of the MathWorks, Inc., 1994-2007.

quite straightforward to interrelate indices of certain attributes i.e. with the use of variables. However, in a `MatLab` environment all the processing is based on matrix manipulation. For this reason, arrays and matrices of two to four dimensions will be employed to address certain problem operations.

5.3 Stage I: Random Allocation of Machines to Cells

Before presenting an approach for allocating machines to cells there is a need to determine the number of cells in the system. It has been stated by Foulds *et al.* [FFW06] that the number of cells created could be any integer number between:

$$CELL_{MIN} = \lceil (1/E_{MAX} \times \sum_{i=1}^{NM} KM_i) \rceil$$

and

$$CELL_{MAX} = \lfloor (1/E_{MIN} \times \sum_{i=1}^{NM} KM_i) \rfloor$$

where KM_i, as already stated in Chapter 3, is given by the formulae below.

$$KM_i = \lceil \sum_{j=1}^{NP} UTIL_{i,j} \rceil$$

However, it was found in practice, that in the majority of cases the best solutions are produced when the number of cells is set to $CELL_{MIN}$. The same scheme is adopted here as well.

Moreover, a set of restrictions as expressed via constraints (5.2), (5.7), (5.8), and (5.10) will be interpreted in `MatLab` when allocating machines to cells. These can be described in words as:

1. k^{th} instance of machine of type i must be assigned to exactly one cell;
2. Each cell can accommodate between E_{MIN} and E_{MAX} machine instances;
3. Any duplicate machines are allocated to lower numbered cells in successive numerical order.

5.3.1 Design of the Algorithm

A natural representation of the initial solution involves a random allocation of machines to cells. In order to fill each cell with machine instance pairs two procedures are examined:

- the random selection of the capacity for the chosen cell;
- the random selection of the machine pair (machine type and its instance) to allocate to the chosen cell.

The random generation of the above elements is based on the uniform distribution[2] of random elements.

The algorithm that follows, named as Routine 1, describes in pseudo terms the implementation of allocating machines to cells.

Routine 1: Machine cell allocation

```
1. Set up a vector holding the number of machine instances for each machine type i.e. $KTYPES$;
2. Set up a 2-D matrix named $MACHMATRIX$ of size $(NM \times length(KTYPES))$ to hold machine types and machine instances. Allocate '1' when machine instance exists and '0' otherwise;
3. Identify row-column indices (coordinates) from step 2 where '1' exists (using the MatLab function find()); sort these indices in ascending order and store them in a matrix named $MACHCOORD$;
4. Initialize a 3-D matrix named $CELLMATRIX$ of size $(NM \times KMAX \times NC)$ to hold machine type, machine instance and cell number respectively. Allocate machines to cells but do not consider duplicate machine instances order yet;
    (a) Choose cell to fill (store number to avoid filling this cell again);
    (b) Choose machine capacity, within $E_{MIN}$ and $E_{MAX}$ for the cell in step 4(a);
    (c) Choose randomly a machine instance pair from the sorted matrix defined in step 3 (delete pair once choosing, in order to avoid choosing this pair again) and update $CELLMATRIX$ for corresponding cell;
    (d) Repeat the above step until the corresponding chosen cell defined in step 4(a) is filled with a number of machine instances equal to the chosen capacity determined in 4(b);
    (e) Repeat steps 4(a) to 4(d) until all cells are filled.
```

[2]A uniform distribution of random numbers on a specified interval $[a, b]$ is implemented by multiplying the output of $rand(n)$ by $(b-a)$ and then adding a (n is the size of the array required). $rand(n)$ is a MatLab function which returns an n-by-n matrix of random entries.

At the end of this stage each cell accommodates between E_{MIN} and E_{MAX} machine instances, while the k^{th} machine instance of type i is allocated to exactly one cell;

5. *Re-arrange* the elements of the $CELLMATRIX$ from step 4 so that duplicate machines are allocated to lower numbered cells in successive numerical order. The number of allocated machine instances to each cell remains unchanged from step 4, but the machine instances in all cells are considered and re-arranged in order.

Although this description provides an indication of all steps involved in machine cell allocation, it is worth providing some more details for implementing steps 4(b) and 5 respectively.

- **Cell Machine Capacity**

The decision made on the capacity of each of the cells in terms of the number of machine instances that each cell should accommodate is partially random. The method employed is introduced via a simple example where the order of which cell is currently filled (i.e. the first, second etc.) together with the total number of the remaining cells to be filled are taken into account. The example is as follows:

It is assumed that the maximum number of machine instances is equal to 20, i.e. $KTYPES = 20$. Moreover, the number of cells in the system is 4, i.e. $NC = 4$. Also, 4 machines are randomly chosen (uniform distribution parameters were chosen to be in range $[1, length([E_{MIN} : 1 : E_{MAX}])]$ and final result rounded to the nearest integer) to be allocated to the first randomly chosen cell.

When the second cell to be filled is randomly chosen (assume cell 3) the capacity allocated to this cell is not defined randomly. Its capacity can be determined by the formulae that follows:

$$current\ \ cell\ \ capacity =$$
$$round(\frac{sum\ \ of\ \ all\ \ machine\ \ instances\ - \ total\ \ previous\ \ capacity\ \ used}{remaining\ \ cells\ \ to\ \ be\ \ filled\ \ (including\ \ current)}) \quad (5.16)$$

Both total number of remaining cells for allocating machines and total capacity already allocated to previous cells are taken into consideration for current cell. Thus, from (5.16) and given the data described above, the capacity for the second chosen cell will be:

$$round(\frac{20-4}{3}) = 5$$

In a similar way, the capacity for the other two remaining cells can be found. More specifically, for the third cell randomly chosen, the number of machines that could be assigned to it are:

$$round(\frac{20-(4+5)}{2}) = 5$$

For the fourth and final cell the machine capacity will be:

$$round(\frac{20-(4+5+5)}{1}) = 6$$

As can be observed from the example, the total capacity in magnitude is equal to the total number of machine instances in the system, thus all machine instances are accommodated in the existing cells.

- **Machine Instances Rearrangement**

According to constraint (5.10) machines are to be allocated to lower numbered cells in successive numerical order. Just after the completion of step 4 machines are allocated to cells as indicated in the matrix named $CELLMATRIX$. However, machines are placed in a random way without preserving constraint's (5.10) requirements. Before continuing with the rearrangement of machine instances within cells, it is rather useful to check on $CELLMATRIX$'s operation via an example.

$CELLMATRIX$ is binary based of size $(NM \times KMAX \times NC)$ where 1 is placed when the k^{th} instance of machine of type i is allocated in cell q, and 0 otherwise. An illustration of this matrix is given in Figure 5.1. It can be easily observed that cell 4 (currently highlighted) has five machine pairs in it: 2^{nd} machine of type 3, 3^{rd} machine of type 4, 4^{th} machine of type 2, 6^{th} machine of type 5 and 7^{th} machine of type 5.

The strategy employed for rearranging machine instances within cells is provided in Routine 2 as follows:

Routine 2: Rearranging machine instances within cells

- initialise a $CELLMATRIX2$ equal to the size of $CELLMATRIX$;
- initialise a temporary binary matrix, i.e. *temp_matrix* of size $(NC \times KMAX)$ for sorting purposes;
- for all machines whose instances are greater than one
  - find *temp_matrix* for all cells for current machine: store 1 when machine instance of the current machine exists within a cell and 0 otherwise;

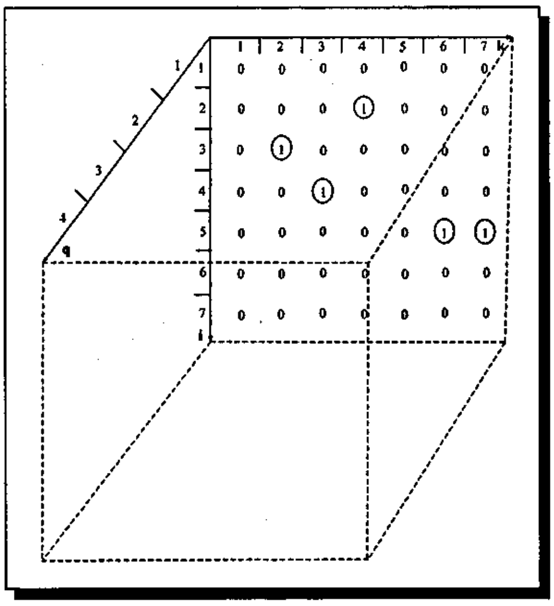


Figure 5.1: Graphical illustration of a 3D matrix named *CELLMATRIX*

```
    – find row-col indices, i.e. cell-machine instance indices (using MatLab function
      find()[3]) in temp_matrix;
    – sort the cell indices in ascending order (instance indices are already sorted in
      ascending order);
    – update CELLMATRIX2 using the sorted indices of cells and corresponding
      machine instances;
    – re-initialise temp_matrix for the next machine;
end for
```

For example assume that matrix *CELLMATRIX*, for every machine instance of type 5 allocated in all cells is of the following form:

$$CELLMATRIX(5,:,:) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \tag{5.17}$$

After employing Routine 2, *CELLMATRIX2* will be produced with rearranged elements as shown in matrix (5.18).

[3][R,C] = FIND(EXPR) returns the row and column indices of the evaluated expression which are non-zero (true elements). Please note that the searching process is column based.

$$CELLMATRIX2(5,:,:) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{5.18}$$

5.4 Stage II: Part Allocation to Machine-Cells

The allocation of parts to machine cells forms the *key* stage of the proposed heuristic algorithm as *ordered part machine sequence* is taken into account imposing a great restriction on the solution strategy to be developed. Each part is tied to its ordered machine sequence, therefore for each part there is only one route to be followed in order for its operation to be complete. Moreover, for each machine in the operation sequence there is a certain utilisation amount to be used from current part. The above restrictions in conjunction with the elements of the objective function, equation (5.1), such as intercellular movements, set-up costs and later revisits of parts to already visited cells, and the remaining constraints of the model will be taken under consideration when designing the allocation of parts to machine cells.

The algorithm proposed for part machine cell allocation will be split into different levels for better presentation. Although each level can stand on its own as it constitutes an independent heuristic algorithm, all levels together operate simultaneously towards the allocation of parts to machines cells. These levels are:

- Initialisation Process;
- Machine Types Identification;
- Cell Sequence Identification;
- Part Allocation.

5.4.1 Initialisation Process

Similar to the allocation of machines to cells where a 3D matrix was formed, another 3D matrix, called $PARTMATRIX$, is also created here for part machine cell allocation purposes. The dimensions represent machine type i, machine instance k and part j respectively (i.e the matrix size is $(NM \times KMAX \times NP)$). Please note that $PARTMATRIX$ is not a binary matrix as each non-zero entry is equal to the corresponding part/machine utilisation.

For illustration purposes of the *PARTMATRIX* an example is provided in Figure 5.2. It is assumed that part's 2 machine sequence is as follows:

$$M_1, M_4, M_5, M_6$$

The part/machine utilisation with respect to the machine sequence is 0.3, 1.2, 0.2 and 0.5 respectively. A possible allocation for part 2 as shown in Figure 5.2, could be as follows:

Part 2 uses 0.3 units of first instance of machine of type 1, 0.2 units of the fifth instance of machine of type 5 and 0.5 units of second instance of machine of type 6. Part 2 is also allocated to more than one instances of machine of type 4 since the utilisation is greater than 1. More specifically a split is carried out forming the utilisations as 0.1, 1, and 0.1 (please note that there are many ways of splitting a part/machine utilisation which is greater than one) where the first, second and third instances of machine of type four are used respectively.

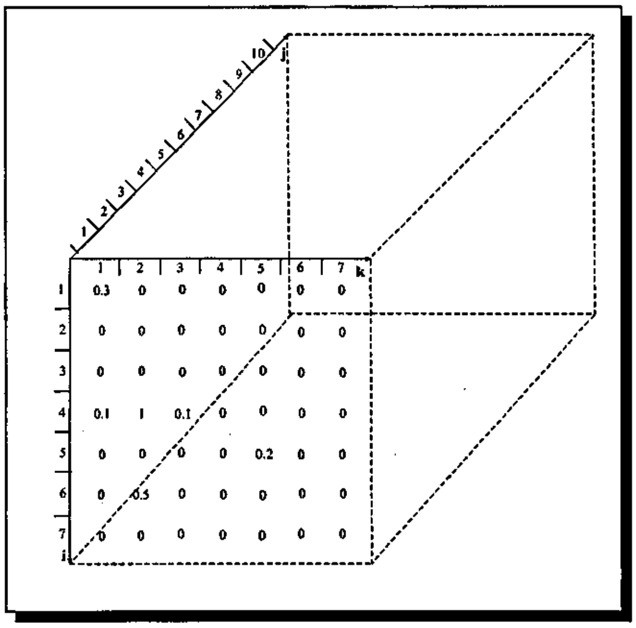


Figure 5.2: Graphical illustration of a 3D matrix named *PARTMATRIX*

PARTMATRIX in conjunction with *CELLMATRIX* are to be used when allocating parts to machine cells. However, in order to have a full reference on the part allocation where allocated parts and the cells used together with the machine instances pairs employed each time can be identified, a 4D binary matrix, named *PCMATRIX*, of size ($NM \times KMAX \times NP \times NC$) is developed. For example, in order to find the machine instances of machine of type i used by part j in all cells the following command will be entered:

$$PCMATRIX(i, :, j, :)$$

For illustration purposes assume the example used before where part 2 was used , i.e. $j = 2$. From Figure 5.2, it can be observed that part 2 uses three instances of machine of type 4. In order to be able to identify the cells (assume four cells in the system) that these machines instances are allocated and used by the current part the following command will be used:

$$PCMATRIX(4, :, 2, :)$$

which will produce a number of vectors each referring to a different cell as listed in Table 5.1. The length of each vector is equal to $KMAX$ (assume that the maximum

Table 5.1: Output for $PCMATRIX(4, :, 2, :)$

V1	0	0	0	0	0	0	0
V2	0	1	1	0	0	0	0
V3	0	0	0	1	0	0	0
V4	0	0	0	0	0	0	0

number of multiple instances belong to part 5 and it is equal to seven). Vector V2 shows part 2 using both the second and third instances of machine of type 4 in cell 2 whereas V3 shows that part 2 is using the fourth instance of machine of type 4 in cell 3.

Every time a part is allocated both $PARTMATRIX$ and $PCMATRIX$ will be updated simultaneously as will be described later.

Moreover, two vectors named *pm_sequence* and *part_moves* are initialised. For storing the machine sequence of each part *pm_sequence* is used, whereas for holding information such as machine type-instance and cell used each time a part (latter also included) is allocated, *part_moves* is employed. The latter will be of special use when calculating the number of later revisits of parts to already visited cells as indicated in the objective function, equation (5.1). Last but not least, similar to the deterministic model, matrices named $UTIL$, $SETUP$, L, $KTYPES$, and $ZTYPES$ are set up to hold part machine utilisation, part machine set-up costs, part machine operation sequences, number of machine instances per machine type and number of operations for each part respectively. More information on all these elements will be provided in later sections.

5.4.2 Machine Types Identification

Before proceeding with the actual allocation of parts to machine cells there is a need to identify the machine types as stored in $CELLMATRIX(i,k,q)$. In order to ease the implementation, the 3D matrix is mapped temporarily onto a 2D matrix which will get hold of machine types and machine instances while the cell index is updated (a "for" loop is used). The `MatLab` function `find()` is used to determine both coordinates (row and column) for machine types and their instances in the temporary matrix, therefore two vectors with indices are returned. However, from these two only the one referring to machine types, row, is needed. Machine types with reference to the cell currently investigated are stored in a 2D matrix named $MACHINENUMBERq$ which is of size $(NC \times E_{MAX})$ (E_{MAX} is used because each cell will not have more than E_{MAX} total number of machines in it).

For clarification purposes an example is provided. Assume that the maximum number of machines allowed for each cell is six, i.e. $E_{MAX} = 6$, total number of cells in the system is equal to four, i.e. $NC = 4$, and total number of machine instances is twenty, i.e. $TMI = 20$. Then a possible allocation of machine instances to cells is provided in $MACHINENUMBERq$, matrix (5.19).

$$MACHINENUMBERq = \begin{bmatrix} 2 & 5 & 6 & 2 & 5 & 0 \\ 1 & 4 & 7 & 2 & 5 & 0 \\ 3 & 4 & 2 & 5 & 5 & 0 \\ 3 & 6 & 4 & 5 & 5 & 0 \end{bmatrix} \tag{5.19}$$

It can be seen from matrix (5.19) that for instance cell 1 contains two machine instances of type 2 and two instances of machine of type 5. Although knowing where multiples of machines are located is important, there is no need to keep a record of them here. What is important is to identify the machine types in each cell as they will be needed later on when part machine sequence will be involved. Also $MACHINEMBERq$ might include some zeros at the tail of each row if not all cells are filled in with the maximum allowable number of machines (e.g. rows one to four). Therefore, in order to remove multiples and zeros appearing in each row two routines, Routine 2 and Routine 3 are developed and described below.

Routine 3: Remove multiple elements from a row vector

1. Set up an input vector, i.e. Vin
2. Set up an output vector i.e. $Vout$;

```
3. Set counter to 1, i.e $N = 1$;
4. while $Vin$ is not empty
    (a) Find the indices in $Vin$ equal to $Vin(1)$ (the indices length is at least one)
    (b) Store the element of first index found in $Vout$;
    (c) Remove this element from $Vin$;
    (d) Accumulate counter $N$;
   end while
```

Routine 4: Remove zero entries from a row vector

```
• Find the indices of zero entries in $Vin$;
    1. if no zero entries found return original vector, i.e $Vout = Vin$;
    2. else remove the zero elements as required.
```

Applying the above routines to eliminate multiples and zero entries to the first row of $MACHINENUMBERq$ the following is obtained:

$$MACHINENUMBERq(1,:) = \begin{bmatrix} 2 & 5 & 6 \end{bmatrix} \tag{5.20}$$

5.4.3 Cell Sequence Identification

The next step, after identifying the machine types included in each of the cells, is to determine the sequence of cells that will be used by each part when part allocation to machine cells commences. Please note that the strategy utilised for cell sequence identification is within the part allocation procedure as will be seen later.

For identifying the cell sequence two approaches were implemented namely:

- Cell sequence relative to part machine sequence
- Continuous cell sequence relative to part machine sequence

details of which are provided next.

• **Approach 1: Cell Sequence Relative to Part Machine Sequence**

The first approach designed for determining the cell sequence is based on organizing cells in descending order having the majority of machines relative to part machine sequence and described in Routine 5 next.

Routine 5: Cell sequence based on majority of machines in cells

```
1. Set up a vector for holding machines for a certain cell i.e. machines_in_cell;
2. Set up a vector for keeping cell and the total number of machines relative to part
   machine sequence, i.e. cell_and_machines;
3. Initialize temporary vector for classifying cells, i.e. temp_cell_seq;
4. for each cell
      (a) From corresponding $MACHINENUMBERq$ remove multiple and zero en-
          tries and determine machines_in_cell;
      (b) Set up a scalar vector, machines_sum, for keeping the total number of machines
          existing in both machines_in_cell and pm_sequence for current cell;
      (c) Compare each entry of machines_in_cell with pm_sequence. When same entry
          encountered update machines_sum.
      (d) Update cell_and_machines vector;
   end for
5. Sort the cells in cell_and_machines in descending order of majority of machines held
   in each;
6. Keep a record of these ordered cells in temp_cell_seq.
```

For example, assume the following machine sequence of a particular part:

$$pm\_sequence = [1 \quad 2 \quad 4 \quad 5 \quad 7]$$

Based on $MACHINENUMBERq$, matrix (5.19), and the procedure described above the cell sequence for the part used when allocation commences will be:

$$temp\_cell\_seq = [2 \quad 3 \quad 1 \quad 4]$$

Please note that cells 1 and 4 have the same total number of machines relative to *pm_sequence*. The way that these are decided to be sorted is arbitrarily chosen. MatLab, however, sorts elements of the "same size" based on their smallest index.

• Approach 2: Continuous Cell Sequence Relative to Part Machine Sequence

The second approach designed for determining a cell sequence is based on identifying a *maximum continuous* machine operation sequence within cells relative to the part machine sequence. During this process a one to one relationship is preserved between machines in cells and part machine sequence. The first step involved concentrates on the identification of any continuous machine sequence by mapping machines in cells to corresponding part machine sequence and it is achieved via Routine 6 described next.

Routine 6: Mapping machines in cells to part machine sequence

```
1. Initialize a 2D matrix, i.e. $ac\_all$ to be equal to empty;
2. for each cell in the system
    (a) Find $machines\_in\_cell$ as defined in first approach (see step 4(a)) in Routine 5;
    (b) Initialize a vector, i.e. $ac$ for keeping an arranged sequence of machines;
    (c) for $i$ equal to 1 up to the length of $pm\_sequence$
        i. if comparison between current element of $pm\_sequence$ and $machines\_in\_cell$
           holds add this element in the same position as indicated in $pm\_sequence$ in $ac$;
        ii. else store 0 in $ac$.
           end if
       end for
    (d) Accumulate $ac$ for current cell in $ac\_all$ matrix (all rows are of the same length
        with the $pm\_sequence$);
   end for
```

For example, assume $pm\_sequence = [1\ \ 2\ \ 4\ \ 5\ \ 6\ \ 7]$ and machines in cells, for cells 1 to 4 are:

$$\begin{array}{ll} cell\ 1: & [1\ \ 2\ \ 4\ \ 3] \\ cell\ 2: & [3\ \ 4\ \ 6\ \ 8] \\ cell\ 3: & [1\ \ 2\ \ 5\ \ 6\ \ 7] \\ cell\ 4: & [1\ \ 2\ \ 4\ \ 5\ \ 3] \end{array}$$

Employing the procedure described above results in *ac_all* of the following form:

$$ac\_all = \begin{bmatrix} 1 & 2 & 4 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 6 & 0 \\ 1 & 2 & 0 & 5 & 6 & 7 \\ 1 & 2 & 4 & 5 & 0 & 0 \end{bmatrix} \tag{5.21}$$

Both cells 4 and 3 complement each other as first four machines of *pm_sequence* are included within cell 4 and the remaining two within cell 3. Although it is obvious for the decision maker that probably the best route for the current part would be to visit first cell 4 and then cell 3, there is a need for a specific mechanism for producing that. Therefore, another step is involved within the current cell sequence procedure where identification of zero entries in *ac_all* is taking place as shown in Routine 7.

Routine 7: Finding positions of zero entries in *ac_all*

```
1. Initialize a 2D matrix, i.e. $zi\_all$ same size as $ac\_all$ with all entries equal to length
   of $(pm\_sequence + 1)$;
2. Find indices (row $i$, column $j$) of zero entries in $ac\_all$;
3. Sort $i$ indices in ascending order using MatLab function sort [4]; return sorted elements
   in $y$ and corresponding indices of the sorted elements in $k$;
4. Take $i$ of $k$ store it in $ik$ ($ik$ the same as $y$) and $j$ of $k$ and store it in $jk$;
5. Set counter to 1, i.e zz=1;
6. while $zz$ is less than or equal to the maximum element of $ik$;
    (a) Find the indices where current $zz$ is equal to $ik$ and store them in $i\_set2$;
    (b) Store the $jk$ elements of indices $i\_set2$ in $zi\_all$;
    (c) Accumulate counter;
   end while
```

Applying the above procedure when part machine sequence is [1 2 4 5 6 7] and *ac_all* of form (5.21) the following is obtained:

[4] $[Y, Yi] = sort(A(:,1),\ 'ascend')$ sorts first column of matrix A in ascending order while returning the sorted elements in Y and the indices of sorted elements in Yi.

$$zi\_all = \begin{bmatrix} 4 & 5 & 6 & 7 & 7 & 7 \\ 1 & 2 & 4 & 6 & 7 & 7 \\ 3 & 7 & 7 & 7 & 7 & 7 \\ 5 & 6 & 7 & 7 & 7 & 7 \end{bmatrix} \qquad (5.22)$$

First column of *zi_all* includes for all cells the positions (indices) of first zero entries encountered in *ac_all*. In other words, for example, in cell 4 the first element recorded in *zi_all* has a value five meaning that the first zero encountered in *ac_all* for the current cell is in position 5, thus the first four elements of part machine sequence which is of the form [1 2 4 5 6 7], i.e. machines 1, 2, 4 and 5 are sequentially included within this cell. The latter implies that the greater the value of the index of the zero elements found in *ac_all* the longer the continuous machine sequence in cells relative to part machine sequence.

The complete description of the algorithm for finding the cell sequence for current part allocation via identifying and determining a maximum continuous cell sequence relative to part machine sequence is provided in Routine 8 described next.

Routine 8: Cell sequence based on max continuous sequence of machines

```
1. Initialize an index referring to a machine in part machine sequence, i.e. $jj = 1$;
2. Initialize temp vector for classifying cells, i.e. temp_cell_seq (similar to Approach 1);
3. while $jj$ is less than or equal to the length of pm_sequence;
   (a) Initialize vector for storing the indices (i.e. cell no.'s) of the non-zero elements
       in ac_all column-wise, i.e. i_set1;
   (b) Find matrix ac_all (refer to Routine 6);
   (c) Find the index(ices) where elements in column $jj$ of ac_all matrix are the same
       with $jj$ machine in pm_sequence and store them in i_set1;
   (d) if $jj$ is equal to the last element of pm_sequence:
         i. Fill cells by adding the cell corresponding to the first index found in i_set1
            in temp_cell_seq;
       else
         i. Determine matrix zi_all (refer to Routine 7);
         ii. Sort the elements of the first column of zi_all in descending order and
             return sorted elements and their corresponding indices in yz1_temp and
            $index1\_temp$ respectively; if only one element found store this element in
            $yz1\_temp$ and add one as its index in $index1\_temp$;
        iii. Take the $i\_set1$ of $index1\_temp(1)$ (cell no.) and update the cell sequence
            $temp\_cell\_seq$;
        iv. Update $jj$ to be equal to $yz1\_temp(1)$ so that search continues from this
            element in the $pm\_sequence$;
      end if
    end while
4. Find all remaining cells not involved in the arranged sequence found so far and add
   them to $temp\_cell\_seq$.
```

Using matrix (5.21) and current part's machine sequence, e.g. [1 2 4 5 6], the following sequence of cells will be obtained via Routine 8:

$$temp\_cell\_seq = [4 \;\; 3 \;\; 1 \;\; 2]$$

The first two cells listed match exactly the decision maker's choice. Remaining unused cells (i.e. 1 and 2), are simply added to the tail of *temp_cell_seq* as step 4 of the algorithm indicates.

In conclusion, the second approach employed organizes the cells in a way that the maximum continuous sequence of machines in cells relative to part machine sequence is preserved. It is worth mentioning that if the first element of a part machine sequence is not included within the cell with the maximum recorded sequence of machines then the first cell to be added into the required cell sequence will be that one holding the first machine so that the ordered sequence is preserved throughout. In contrast with the second approach, the first approach considers only the number of machines in cells relative to the part machine sequence and sorts cells in descending order starting with the cell that holds the majority of them.

Overall, when considering the complete CF model where each part has a specific route attached to it, i.e. a machine operation sequence, approach two seems to be more suitable for current study. However, the decision on which approach to employ will become obvious later on when testing is carried out.

5.4.4 Part Allocation

Once the machines have randomly been allocated to machine cells, the final and *key* stage to be implemented is the allocation of parts to machine cells. A number of phases such as initialisation process, identification of machine types within each cell and identification of cell sequence relative to the part machine sequence performed in previous sections will be recalled when allocating parts to machine cells.

It is worth mentioning, that throughout the allocation procedure the part machine sequence is strictly preserved, thus increasing the complexity of the system.

- **Algorithm Design**

Before proceeding with the description of the pseudo algorithm for allocating parts to machine cells, a short description will follow identifying the key operations involved in the design stage:

1. Sort the parts into ascending order of total processing requirements (for each part sum up all relevant part/machine utilisation amounts and sort them in ascending order);
2. Let j be the next non-allocated part to be allocated;
3. Identify part's j machine operation sequence and determine the sequence of cells for allocating current part (see Approaches 1, 2);
4. Let d be the first machine in current part's machine sequence;
5. Let q be the first cell in sequence;
6. Check if cell q in sequence includes the machine d of part's j machine operation sequence (the latter is examined when Approach 1 is employed and also when Approach 2 is utilised but at later stages of the allocation process). If it is, find all the instances of current machine in candidate cell otherwise, check the next cell in the cell sequence;
7. Let k be the first machine instance found;
8. Check current part/machine utilisation and also remaining capacity of instance k found within current cell;
9. Depending upon the values of both, a number of cases are examined as illustrated in Table 5.2 (assume remaining capacity of a machine instance is equal to a and current part utilisation is equal to b) and part is allocated accordingly;

10. After current part's allocation a number of elements are updated (as will be seen in Routines 10-12). Also current status of part/machine utilisation is checked: if latter is zero current part with reference to a certain machine in its sequence is fully allocated and the process continues with next part in sequence, unless the length of its machine sequence has not been reached yet, i.e. there are still machines for current part to be allocated to, so the process will continue from there. In case that the current part/machine utilisation is not zero then continue allocating current part to another instance of the same machine type till this becomes zero.

Table 5.2: Remaining Capacity Versus Current Utilisation

1. $a=1$	4. $b=1$	7. $a+b=1$	10. $a>b$
2. $a>1$	5. $b>1$	8. $a+b>1$	11. $a=b$
3. $a<1$	6. $b<1$	9. $a+b<1$	12. $a<b$

From Table 5.2, case 1 won't be examined as nothing more can be allocated to a machine instance whose capacity has reached one unit. For the same reason case 2 is deleted. For the rest of the cases, i.e. 3 to 12, a decision was made to combine some of these and a number of combined expressions was derived:

$$\begin{aligned} i:&\quad a<1 \quad \& \quad b>=1 \\ ii:&\quad a<1 \quad \& \quad b<1 \\ iii:&\quad a<1 \quad \& \quad a>=b \end{aligned}$$

Expression i is complete (nothing else can be chosen), whereas for expressions ii and iii a number of other choices exist. Therefore, considering ii and when $a<=b$ cases 7, 8, and 9 are examined in sequence. Similarly for iii cases 7, 8, and 9 are nested and examined separately.

For a thorough description of the allocation of parts to machine cells Routines 9 to 12 are presented below. Please note that Routine 9 constitutes the main part allocation algorithm, whereas the other routines are simply created to split the algorithm into smaller chunks and make it easier to present. For more information on the actual MatLab code involved for the allocation of parts to machine cells see Appendix D.

Routine 9: Part allocation to machine cells

```
• Sort parts into ascending order of total processing requirements and store sequence
  in $UTIL\_sortedi$;
• Assign $UTIL$ to $UTIL\_temp$ initially; $UTIL\_temp$ stands for current utilisation;
• for each part $j$ in $UTIL\_sortedi$
    1. Determine cell sequence, temp_cell_seq, for current part (see Approaches 1 or
       2)
  • for each machine $d$ in pm_sequence
    1. Initialise counter for navigating through cells, i.e. $q2 = 1$
    2. while q2 (cell index) is less than or equal to the length of temp_cell_seq;
        – if current machine, $pm\_sequence(d)$, exists in cell $temp\_cell\_seq(q2)$
            * Initialize vector for keeping machine instances of a specific machine
              type within current cell, i.e. mach_instances;
            * Find mach_instances for current part and cell;
            * Identify all non-zero indices in mach_instances and store them in
              instance_noi;
            * Initialise counter for navigating through machine instances, i.e. $k2 = 1$;
            * while $k2$ less than or equal to length of instance_noi
            (a) if remaining capacity for current machine instance pair less than 1 &
                current part machine utilisation greater than or equal to 1
                Go to Routine 10;
            (b) else if remaining capacity for current machine instance pair less than
                1 & current part machine utilisation less than 1
                Go to Routine 11;
                end if
            (a) if remaining capacity for current machine instance less than 1 &
                remaining capacity for current machine instance greater than or equal
                to current part machine utilisation
                Go to Routine 12;
                end if
            Accumulate iteration counter $k2$;
            end while
          end if
    3. if $UTIL\_temp$ becomes 0 move onto the next machine in sequence;
       Accumulate iteration counter $q2$;
       end while
  end for loops.
```

Routine 10

```
1. $UTIL\_1$ equals 1 minus remaining capacity;
2. Update corresponding $PARTMATRIX$ with $UTIL\_1$;
3. $UTIL\_temp$ equals $UTIL\_temp$ minus $UTIL\_1$;
4. Update $part\_moves$; Update $PCMATRIX$;
5. if $UTIL\_temp$ becomes zero move onto the next machine in sequence.
```

Routine 11

```
• if remaining capacity for current machine instance pair less than or equal to current
  part machine utilisation
    – if remaining capacity plus current utilisation equal to 1
        1. Update $PARTMATRIX$ with current $UTIL\_temp$;
        2. $UTIL\_temp$ equals zero;
        3. Update $part\_moves$; Update $PCMATRIX$;
    – else if remaining capacity plus current utilisation greater than 1
        1. $UTIL\_1$ equals 1 minus remaining capacity;
        2. update corresponding $PARTMATRIX$ with $UTIL\_1$;
        3. $UTIL\_temp$ equals $UTIL\_temp$ minus $UTIL\_1$;
        4. Update $part\_moves$; Update $PCMATRIX$;
        5. if $UTIL\_temp$ becomes zero move onto the next machine in sequence
    – else (remaining capacity plus current utilisation less than 1)
        1. update $PARTMATRIX$ with current $UTIL\_temp$;
        2. $UTIL\_temp$ equal zero
        3. Update $part\_moves$; Update $PCMATRIX$;
      end if
  end if
```

Routine 12

```
• if remaining capacity plus current utilisation equal to 1
    1. Update $PARTMATRIX$ with current $UTIL\_temp$
    2. $UTIL\_temp$ equal zero
    3. Update $part\_moves$; Update $PCMATRIX$;
• else if remaining capacity plus current utilisation greater than 1
    1. $UTIL\_1$ equals 1 minus remaining capacity;
    2. Update corresponding $PARTMATRIX$ with $UTIL\_1$;
    3. $UTIL\_temp$ equals $UTIL\_temp$ minus $UTIL\_1$;
    4. Update $part\_moves$; Update $PCMATRIX$;
    5. If $UTIL\_temp$ becomes zero move onto the next machine in sequence;
• else (remaining capacity plus current utilisation less than 1)
    1. Update $PARTMATRIX$ with current $UTIL\_temp$;
    2. $UTIL\_temp$ equals zero
    3. Update $part\_moves$; Update $PCMATRIX$;
  end if
```

For illustration purposes of the routines above an example will be provided next, which will assist in better understanding of the methodology employed for allocating parts to machine cells when part machine operation sequence is taken into account.

• Illustration of Part Allocation Strategy: Example (II)

In order to be able to demonstrate the proposed strategy for allocating parts to machine cells a data set is randomly generated with a relatively small number of parts and machine types. Assume $NP = 8$, $NM = 4$ and also $TMI = 15$, $ZOPER = 4$, $KMAX = 5$, $E_{MAX} = 6$, $E_{MIN} = 3$ and $NC = 3$. Also part/machine utilisation (i.e. $UTIL$), part/machine set-up cost (i.e. $SETUP$), and parts machine operation sequences (i.e. L) all randomly generated are as follows:

$$UTIL = \begin{bmatrix} 0.6 & 0 & 0.7 & 1.1 & 0.6 & 0.9 & 0.6 & 0 \\ 0.8 & 0.3 & 0.7 & 0.5 & 0.4 & 0.4 & 0 & 0 \\ 0 & 0.8 & 0 & 0.5 & 0.5 & 0.2 & 0.8 & 0.9 \\ 0.4 & 0.4 & 0 & 0.4 & 0.7 & 0 & 0 & 0 \end{bmatrix} \tag{5.23}$$

$$SETUP = \begin{bmatrix} 4.7 & 0 & 1.16 & 2.52 & 0.45 & 4.72 & 3.11 & 0 \\ 1.77 & 2.08 & 4.81 & 2.06 & 0.41 & 1.11 & 0 & 0 \\ 0 & 0.27 & 0 & 3.14 & 0.68 & 2.89 & 0.65 & 4.27 \\ 2.08 & 4.56 & 0 & 0.26 & 3.31 & 0 & 0 & 0 \end{bmatrix} \tag{5.24}$$

$$L = \begin{bmatrix} 1 & 2 & 4 & 0 \\ 2 & 3 & 4 & 0 \\ 1 & 2 & 0 & 0 \\ 4 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 0 \\ 1 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix} \tag{5.25}$$

Moreover, $KTYPES$ and $ZTYPES$ are provided below as follows:

$$KTYPES = \begin{bmatrix} 5 & 4 & 4 & 2 \end{bmatrix},$$
$$ZTYPES = \begin{bmatrix} 3 & 3 & 2 & 4 & 4 & 3 & 2 & 1 \end{bmatrix} \tag{5.26}$$

Since allocation of machine to cells is random, one possible allocation of machine to cells is described via $CELLMATRIX2$ for all cells and shown in (5.27).

$$CELLMATRIX2(:,:,1) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$
$$CELLMATRIX2(:,:,2) = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{5.27}$$
$$CELLMATRIX2(:,:,3) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Moreover, assuming that parts are sorted in ascending order of total processing requirements, $UTIL\_sortedi$ will be of the following form:

$$UTIL\_sortedi = \begin{bmatrix} 8 & 3 & 7 & 2 & 6 & 1 & 5 & 4 \end{bmatrix} \tag{5.28}$$

Applying Routines 9-12 to the initial allocation of machines to cells (see (5.27)), parts are allocated to machine cells and a number of elements are produced presenting this allocation. The first thing to be examined is the actual allocation of parts to machines (i.e the corresponding $PARTMATRIX$) as a certain machine instance pair utilisation amount is employed by each part while both part machine operation sequence and sequence of cells[5] for parts allocation are taken into account. The $PARTMATRIX$ produced for all parts when four machine types and five machine instances (maximum allowable number of instances of a specific machine type) are considered, is shown in (5.29).

[5]Assume for current example that cells are organised based upon their maximum continuous sequence of machines relative to the part machine sequence (i.e. Approach 2).

$$PARTMATRIX(:,:,1) = \begin{bmatrix} 0.3 & 0.3 & 0 & 0 & 0 \\ 0.3 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$PARTMATRIX(:,:,2) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0.4 & 0 & 0 & 0 \end{bmatrix},$$

$$PARTMATRIX(:,:,3) = \begin{bmatrix} 0.7 & 0 & 0 & 0 & 0 \\ 0.7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$PARTMATRIX(:,:,4) = \begin{bmatrix} 0 & 0.2 & 0.4 & 0.5 & 0 \\ 0 & 0 & 0.3 & 0.2 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0.4 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$PARTMATRIX(:,:,5) = \begin{bmatrix} 0 & 0.5 & 0 & 0 & 0.1 \\ 0 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 \\ 0.1 & 0.6 & 0 & 0 & 0 \end{bmatrix},$$

$$PARTMATRIX(:,:,6) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.9 \\ 0 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$PARTMATRIX(:,:,7) = \begin{bmatrix} 0 & 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$PARTMATRIX(:,:,8) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.29}$$

In order to able to see in which cell each part is allocated to, $PCMATRIX$ is presented in (5.30). $PCMATRIX(:,:,j,q)$ presents part's j machine cell q allocation. $PCMATRIX(:,:,j,q)$ is equal to one when a part j occupying a specific machine type - machine instance pair is allocated in cell q, 0 otherwise.

$$PCMATRIX(:,:,1,1)=\begin{bmatrix}1&1&0&0&0\\1&1&0&0&0\\0&0&0&0&0\\1&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,2,1)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,3,1)=\begin{bmatrix}1&0&0&0&0\\1&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},$$

$$PCMATRIX(:,:,4,1)=\begin{bmatrix}0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\1&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,5,1)=\begin{bmatrix}0&1&0&0&0\\0&1&0&0&0\\0&0&0&0&0\\1&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,6,1)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},$$

$$PCMATRIX(:,:,7,1)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,8,1)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,1,2)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},$$

$$PCMATRIX(:,:,2,2)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,3,2)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,4,2)=\begin{bmatrix}0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},$$

(5.30)

$$PCMATRIX(:,:,5,2)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,6,2)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,7,2)=\begin{bmatrix}0&0&1&0&0\\0&0&0&0&0\\1&1&0&0&0\\0&0&0&0&0\end{bmatrix},$$

$$PCMATRIX(:,:,8,2)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\1&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,1,3)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,2,3)=\begin{bmatrix}0&0&0&0&0\\0&0&1&0&0\\0&0&1&0&0\\0&1&0&0&0\end{bmatrix},$$

$$PCMATRIX(:,:,3,3)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,4,3)=\begin{bmatrix}0&0&0&1&0\\0&0&1&1&0\\0&0&0&1&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,5,3)=\begin{bmatrix}0&0&0&0&1\\0&0&0&0&0\\0&0&0&1&0\\0&1&0&0&0\end{bmatrix},$$

$$PCMATRIX(:,:,6,3)=\begin{bmatrix}0&0&0&0&1\\0&0&1&0&0\\0&0&1&1&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,7,3)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix},\ PCMATRIX(:,:,8,3)=\begin{bmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{bmatrix}$$

Moreover, in order to demonstrate for each part all of its related cell movements while its machine operation sequence is preserved, *part_moves* is produced and is shown in (5.31). Each column (starting from the left) represents part, cell, machine type and corresponding machine instance. For example part 6 visits only cell 3 (see (5.30)) where it firstly uses 5th machine of type 1, secondly 3rd machine of type 2 and finally 3rd and 4th machine instances of type 3 as its machine operation sequence also indicates (see (5.25) matrix L for part 6). *part_moves* will be used very efficiently when evaluating the value of the objective function.

$$part\_moves = \begin{bmatrix} 8 & 2 & 3 & 1 \\ 3 & 1 & 1 & 1 \\ 3 & 1 & 2 & 1 \\ 7 & 2 & 1 & 3 \\ 7 & 2 & 3 & 1 \\ 7 & 2 & 3 & 2 \\ 2 & 3 & 2 & 3 \\ 2 & 3 & 3 & 3 \\ 2 & 3 & 4 & 2 \\ 6 & 3 & 1 & 5 \\ 6 & 3 & 2 & 3 \\ 6 & 3 & 3 & 3 \\ 6 & 3 & 3 & 4 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 4 & 1 \\ 5 & 3 & 1 & 5 \\ 5 & 1 & 1 & 2 \\ 5 & 1 & 2 & 2 \\ 5 & 3 & 3 & 4 \\ 5 & 3 & 4 & 2 \\ 5 & 1 & 4 & 1 \\ 4 & 1 & 4 & 1 \\ 4 & 1 & 1 & 2 \\ 4 & 2 & 1 & 3 \\ 4 & 3 & 1 & 4 \\ 4 & 3 & 2 & 3 \\ 4 & 3 & 2 & 4 \\ 4 & 3 & 3 & 4 \end{bmatrix} \tag{5.31}$$

5.5 Stage III: Evaluation of Objective Value and Solution

The objective function, equation (5.1), includes three elements that need to be minimised: number of distinct allocations of parts to cells, part/machine set-up cost and later revisits of parts to already visited cells. While allocation of machine to cells is implemented as described in Stage I and part allocation as presented in Stage II a number of elements are continuously updated. These will help in the evaluation of all three components of the objective function, hence the evaluation of the objective function as a whole.

5.5.1 Total Number of Part/Cell Distinct Allocations

The first procedure to be implemented is for calculating the distinct allocation of parts to cells as shown in Routine 13.

Routine 13: Distinct allocations of parts to cells

```
• Initialise a scalar vector, i.e. $W = 0$, for keeping the total number of distinct allocations of parts to cells;
• Set up a 2D binary matrix, i.e. $W\_jq$ of size $NP \times NC$ for part cell allocation (this will be used when evaluating the objective function);
• for all parts in $UTIL\_sortedi$
    – Find the index (indices) in the first column of $part\_moves$ where current part is; Store these in $I$;
    – Based upon $I$ return the corresponding element(s) existing in second column, i.e actual cell number(s); Store this in $qt$;
    – while $qt$ is not empty
        1. Compare the first element of $qt$ with the whole of $qt$ and return the indices of the same element(s) in $I2$;
        2. Update $W\_jq$ by setting one for current part and first cell in $qt$;
        3. Remove cells already investigated (using $I2$) from $qt$;
        4. Accumulate $W$ as one more part cell distinct allocation is encountered;
      end while
  end for
```

For example, assume the total *part_moves* matrix (5.31) where part currently to be considered is 5. The segment from *part_moves* referring to part 5 is as follows:

$$\begin{bmatrix} 5 & 3 & 1 & 5 \\ 5 & 1 & 1 & 2 \\ 5 & 1 & 2 & 2 \\ 5 & 3 & 3 & 4 \\ 5 & 3 & 4 & 2 \\ 5 & 1 & 4 & 1 \end{bmatrix}$$

When applying Routine 14 the following qt is produced:

$$qt = \begin{bmatrix} 3 \\ 1 \\ 1 \\ 3 \\ 3 \\ 1 \end{bmatrix}$$

Comparing the first element of qt with the rest of the elements a number of indices are returned, i.e. $I2 = [1 \;\; 4 \;\; 5]$. Before removing the corresponding elements from qt, W_jq is updated as follows:

$$W\_jq = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Also W is assigned to 1. The operation for current part continues until qt becomes empty and then the next part in sequence is examined.

5.5.2 Total Part/Machine Set-up Cost

In order to be able to calculate the total part machine set-up cost there is a need to determine in advance the number of machines of type i used by part j in cell q. For the latter Routine 14 is implemented.

Routine 14: Total Number of machines of type i used by part j in cell q

```
• Set up a 3D matrix $S\_ijq$ of size $(NM \times NP \times NC)$ to hold the number of machines
  of type $i$ used by part $j$ in cell $q$
• Set up a 2D matrix, i.e. $S\_ij$ of size $(NM \times NP)$ for keeping the total machines of
  type $i$ used by part $j$
• for all parts
    – for all machines
        * for all cells
            1. Find the sum of machine instances via $PCMATRIX$ and store this in
               $S\_ijq$;
          end for
        * Map 3D matrix, $S\_ijq$, to 2D, $S\_ij$[6] by summing up all machine instances
          of certain type in all cells used by current part;
      end for
  end for
```

From example (II) considered earlier and via Routine 14 S_ijq will be as follows:

$$S\_ijq(:,:,1) = \begin{bmatrix} 2 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix},$$

$$S\_ijq(:,:,2) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$S\_ijq(:,:,3) = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Moreover, S_ij will also be obtained where each entry is produced by summing up for all cells all machine instances of machine type i used by certain part j. The result is

[6]This mapping is only performed in order to be able to define later on the total set-up cost within the objective function (see equation (5.32) presented in page 120).

shown below:

$$S\_ij = \begin{bmatrix} 2 & 0 & 1 & 3 & 2 & 1 & 1 & 0 \\ 2 & 1 & 1 & 2 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 2 & 2 & 1 \\ 1 & 1 & 0 & 1 & 2 & 0 & 0 & 0 \end{bmatrix}$$

5.5.3 Calculation of Part Cell Later Revisits

The third component within the objective function that needs to be determined is the number of later revisits of parts to already visited cells. The strategy employed is described in Routine 15.

Routine 15: Later part revisits estimation

```
• Initialise a vector, i.e. $extra\_moves$ for keeping the total number of revisits for each
  part;
• for all parts in $UTIL\_sortedi$ sequence
    1. Find the segment from $part\_moves$ that corresponds to current part and store
       it in $part\_moves\_temp1$;
    2. Take the second column (i.e. cell column) from $part\_moves\_temp1$ and store
       it in $part\_moves\_temp2$
    3. if length of $part\_moves\_temp2$ is equal to one continue with next part in se-
       quence
       end if
    4. Initialise a scalar vector, i.e. $EXTRA$;
    5. Set up a counter, i.e. $x = 1$;
    6. while $x$ is less than the length of $part\_moves\_temp2$
       (a) for $w$ equal to $x + 1$ up to length $part\_moves\_temp2$
             – if $part\_moves\_temp2$ of $w$ is equal to $part\_moves\_temp2$ of x & the
               distance between w and x is greater than one
               i. Accumulate $EXTRA$;
              ii. Set x equal to $w - 1$;
               end if
           end for
       (b) Accumulate $x$;
    end while
    7. Store $EXTRA$ values for each round in $extra\_moves$;
end for
```

From example (II) and matrix (5.31) assume part 5. When Routine 15 is employed *part_moves_temp*1 and *part_move_temp*2 are formed as follows:

$$part\_moves\_temp1 = \begin{bmatrix} 5 & 3 & 1 & 5 \\ 5 & 1 & 1 & 2 \\ 5 & 1 & 2 & 2 \\ 5 & 3 & 3 & 4 \\ 5 & 3 & 4 & 2 \\ 5 & 1 & 4 & 1 \end{bmatrix}$$

$$part\_moves\_temp2 = [3 \;\; 1 \;\; 1 \;\; 3 \;\; 3 \;\; 1]$$

By applying Routine 15 for current data the number of later revisits counted will be equal to 2 as part returns to both cells 3 and 1 that initially visited.

Given a number of key elements (parameters, matrices, etc.) presented within Routines 13 to 15, an attempt was made to express the objective value, following the pattern of mathematical model's objective function, i.e. equation (5.1), with a formulae which can be seen below.

$$OBJVAL = (M_{j,q} \times sum(sum(W\_jq)) + sum(sum(SETUP. \times S\_ij)) \\ + A_j \times sum(extra\_moves)) \quad (5.32)$$

Please note that $SETUP.{*}S\_ij$ produces an element-by-element product of the matrices $SETUP$ and S_ij. Also the 'sum' feature in MatLab returns the sum of elements. If $B = sum(A)$ and A is a matrix then $sum(A)$ treats the columns of A as vectors, returning a row vector of the sums of each column.

5.6 Summary

This chapter presented a three stage heuristic approach for designing and setting up an initial starting framework representing the CF problem. The first stage involved a preliminary allocation of machines to cells. In the second stage parts were allocated to cells based upon the machine cell allocation and a number of key constraints. The final stage involved mainly the evaluation of the objective value and the corresponding problem solution. The most complex stage of all was the design of the allocation of parts to machine cells since part machine operation sequence was taken into account. This stage had to be designed very carefully in order to obtain a good starting point, which can become the foundation for additional study. More specifically, this heuristic procedure will serve the basis for the design of more advanced searching strategies presented in the next two chapters.

Chapter 6

An Iterative Heuristic Approach for CF

6.1 Introduction

An initial solution strictly following the CF problem requirements and developed via a number of heuristic strategies (see Chapter 5) cannot represent the CF system in its full operation. For this reason other heuristic procedures need to be developed for investigating a significant number of good quality solutions on a different searching level. The initial solution in combination with the latter strategies can prove to be very useful when the decision maker attempts to design a CF system involving data sets of significant sizes, thus incorporating realism into the system.

For the purpose of this chapter, a first attempt was made for proposing an iterative heuristic procedure in the presence of an ordered part machine sequence forming a number of solutions when transitions of machines are considered between cells. At each stage machine cells allocation will be updated and part machine cell allocation will be re-implemented for a better cell system configuration. Moreover, computational results will be presented for a variety of problem sizes.

6.2 Heuristic Components

Before introducing the actual design of the proposed heuristic algorithm, it is useful to recall for a moment the well-known descent local search (LS) heuristic (also known as hill climbing). The descent LS algorithm starts from an initial (maybe randomly generated) solution. Further, the search process continues by performing some sequential transformations of solutions, i.e. making moves from solutions to solutions. A move is applied to the current initial solution in order to get a new solution. The moves are controlled, i.e. decisions about moving to other solutions, or not, from the current one are taken depending upon the qualities of solutions (the values of the objective functions involved). So, if the decision is 'positive', i.e. a better objective value is found, then the current solution is replaced by the neighbouring one, which will be used as a 'starting point' for the subsequent trials; otherwise the search continues from the current solution. The whole process continues until the current solution found at each stage becomes locally optimal, which means that no other better solution exists within the neighborhood of the current solution.

The central idea of the proposed algorithm will follow the operation of a descent LS heuristic where an iterative form, with certain strategies built in it, will be employed enhancing the searching process. Before going into more details about the design of the algorithm it is important to identify the type of transitions which will be involved in the current approach for investigating a number of solutions within a pre-specified number of iterations.

Two types of moves will be considered in the search procedure: (a) a single move, and (b) an interchange move. The single move is an operation that moves a machine instance pair, (i, k), from its current cell q (source cell) to a new cell q' (destination cell).

The interchange move is an operation which consists of two independent single moves. If a machine instance pair (i, k) is moved from its source cell q to another cell q' (first single move), then another machine of type i'[1] of instance k' will be moved from the destination cell q' of the first move to the source cell q of the first move (second single move) in exchange. Thus the two moves generated are $((i,k),\ q')$ and $((i',k'),\ q)$.

6.3 Heuristic Algorithm Design

The algorithm proposed based on the local search operative principle (see for instance [GYZ02]) will be split into different phases for better illustration. These phases namely are as follows:

- Initialisation Process;

[1]Please note that the machine of type i' in destination cell q' could happen to be of the same type as the machine of type i in the source cell q. However, these machines will be different instances as each instance is unique in the system. Also the way that this exchange might improve the current solution is down to the candidate part/machine requirements.

- Iterative Procedure;
- Ending Process.

6.3.1 Initialisation Process

In order to present the iterative procedure, certain elements need to be initialised, therefore some additional information/notation will be used and is presented below. As stated already the notation used in Chapter 5 will be adopted here as well.

SOL_temp[2]	current system solution
$OBJVAL\_temp$	current value of objective function
$INIT\_OBJVAL$	initial objective value
$INIT\_SOL$	initial solution
$CELLMATRIX2\_temp$	current allocation of machines to cells
$PARTMATRIX\_temp$	current utilisation allocation of parts to machines
$PCMATRIX\_temp$	current allocation of parts to machines cells (used for evaluating part machine set-up cost)
$part\_moves\_temp$	current part machine cell allocation relative to part machine sequence
W_jq_temp	current distinct allocation of parts to cells
$BEST\_COST$	best objective value found so far
$BEST\_SOL$	best solution found so far
$BEST\_CELLMATRIX2$	best machine cell allocation
$BEST\_PARTMATRIX$	best part machine allocation
$BEST\_PCMATRIX$	best 4D matrix presenting part machine cell allocation
$BEST\_part\_moves$	array recording best part machine cell allocation relative to part machine sequence
$BEST\_W\_jq\_temp$	best distinct allocation of parts to cells
$CELLMATRIX\_temp0$	used as starting point when objective value does not improve
$PARTMATRIX\_temp0$	as above respectively
$PCMATRIX\_temp0$	as above respectively
$part\_moves\_temp0$	as above respectively

The initialisation stage involves storing certain elements before the iterative approach

[2] SOL_temp is a cell array with three matrices stored in it: W_jq_temp (current distinct allocation of parts to cells), S_ij_temp (current number of machines of type i used by part j) and $extra\_moves\_temp$ (current total number of revisits for each part). In addition, $BEST\_SOL$ and $INIT\_SOL$ are the best and initial values of the objective function respectively.

commences. Besides recording best cost and corresponding solution, a number of temporary matrices created will be given the values found in the initial solution in order to be able to navigate the search space by knowing, for instance, the allocation of machines to cells together with the allocation of parts to machines and the exact path that each part follows within cells when its sequence is preserved. Initially these matrices are recorded as the best found so far as well. Please note that within the iterative procedure all values either best or temporary will be continuously updated as will be seen later.

For the starting point however, best single values together with best and temporary matrices are initialised and described in Routine 1.

Routine 1: Initialisation phase for Heuristic Algorithm

- initialise $BEST\_COST$ to be equal to $INIT\_OBJVAL$;
- initialise $BEST\_SOL$ to be equal to $INIT\_SOL$;
- initialise $CELLMATRIX2\_temp$ to be equal to $CELLMATRIX2$;
- initialise $CELLMATRIX2\_temp0$ to be equal to $CELLMATRIX2\_temp$;
- initially assume that $BEST\_CELLMATRIX2$ is equal to $CELLMATRIX2\_temp$;
- initialise $PCMATRIX\_temp$ to be equal to $PCMATRIX$;
- initialise $PCMATRIX\_temp0$ to be equal to $PCMATRIX\_temp$;
- initially assume that $BEST\_PCMATRIX$ is equal to $PCMATRIX\_temp$;
- initialise $PARTMATRIX\_temp$ to be equal to $PARTMATRIX$;
- initialise $PARTMATRIX\_temp0$ to be equal to $PARTMATRIX\_temp$;
- initially assume that $BEST\_PARTMATRIX$ is equal to $PARTMATRIX\_temp$;
- initialise $part\_moves\_temp$ to be equal to $part\_moves$ (as determined in the initial solution);
- initialise $part\_moves\_temp0$ to be equal to $part\_moves\_temp$;
- initially assume that $BEST\_part\_moves$ is equal to $part\_moves\_temp$;
- initialise W_jq_temp to be equal to W_jq;
- initially assume that $BEST\_W\_jq$ is equal to W_jq_temp;
- initialise W_jq_temp0 to be equal to W_jq_temp;

6.3.2 Iterative Procedure

The iterative procedure is the main phase of the heuristic algorithm where investigation of better solutions is sought. It mainly consists of two approaches each representing single and interchange transitions of machine instance pairs. Both of these processes are independent from each other as the algorithm has been designed such that when a single transition takes place for a machine instance pair an interchange is not considered and vice versa. The decision of which transition type to consider is based upon the cell machine capacity of the candidate chosen cells where the transition is going to take place.

The algorithm begins by considering the parts in the system and more specifically the part that causes the majority of the intercellular movements in the initial solution. Given the distinct allocation of parts to cells (W_jq), as produced from the initial starting solution (see Chapter 5), parts are ordered in descending order relative to the number of intercellular movements. This decision was made based on the fact that the distinct allocations of parts to cells in the objective function (see equation (5.1)) is identified as the most important element when trying to minimise the objective function. The better the allocation of parts to cells the better the value of the objective function in the system. The values of the two remaining elements of the objective function, i.e. total part/machine set-up cost and total revisits of parts to already visited cells mostly depend upon the distinct allocation of parts to cells.

Moreover, with reference to the current part in operation, the segment corresponding to its cell machine instance pair allocation relative to its machine sequence is found in *part_moves_temp* and stored in a vector named *qik*. Each machine instance pair in the latter, is a candidate for moving it from the cell currently in use, i.e. source cell, to the remaining cells, i.e. destination cells, visiting them one at a time. For each of these pairs either a single or interchange transition takes place and new solutions are investigated.

Before continuing with the description of the two transition types included within the heuristic approach, a foundation framework for the iterative procedure based on the overall discussion above is presented next in Routine 2.

Routine 2: Iterative Heuristic Framework

```
• Initialise i_part equal to zero;
• while i_part less than or equal to $\omega$ ($\omega$ defined in section 6.3.3 - Ending Process)
• Sort the parts in descending order of the intercellular movements caused, and return
  their indices in $wi$ and the actual parts in $wy$;
  • for each part in $wi$
    – Set $i$ (iterations index) equal to zero;
    – while $i$ is less than $\lambda$ ($\lambda$ defined in section 6.3.3 - Ending Process)
        * Find the segment for current part corresponding to cell no $q$. machine type
          $i$, machine instance $k$ in $part\_moves\_temp$ and store it in $qik$;
        * for c1 equal to one up to the length of $qik$
            1. Assign all cells in a vector named $all\_cells$;
            2. Take the first row in $qik$, find the cell where the first machine instance
               pair is allocated and store it in a vector named $cc$;
            3. Delete this from $all\_cells$;
            4. for $c2i$ equal to one up to the length of $all\_cells$ (all remaining cells)
             (a) Store the cell corresponding to $c2i$ index in $c2$;
             (b) if capacity of machines stored in $c2$ (destination cell) cell is less than
                 $E_{MAX}$ & capacity of $qik(c1,1)$ (source cell) is less than or equal to
                 $E_{MAX}$ & greater than $E_{MIN}$
                 · Single move commences;
                 else if capacity of machines stored in $c2$ (destination cell) cell is
                 greater than $E_{MIN}$ & equal to $E_{MAX}$ & capacity of $qik(c1,1)$
                 (source cell) is greater than $E_{MIN}$
                 · Interchange move commences;
                 end if
               end for
          end for
        * Accumulate $i$;
      end while
  end for
  Accumulate $i\_part$;
  end while
```

• **Single Move**

The approach adopted for a single transition within the search procedure can be outlined by the following steps:

1. single transition execution;
2. update of relevant elements;
3. reallocation of parts to machine cells;
4. evaluation of objective value;
5. update on the best values if objective value improves;

Item four has been described extensively within section 5.5, thus only some extra comments will be provided here. However, for the remaining steps more details are presented next.

More specifically, when a single move is taken into account a candidate machine instance pair i.e. (i, k) of a part will be moved from a source cell, i.e. q, where this machine instance pair is currently allocated, to a destination cell, i.e. q' (first cell in the sequence of the remaining cells). While this happens a number of updates will be considered for the system, with the first update taking place on the current machine cell allocation, i.e. $CELLMATRIX2\_temp$, as shown below:

$$CELLMATRIX2\_temp(i,k,q') = 1 \quad and \quad CELLMATRIX2\_temp(i,k,q) = 0$$

Moving a machine instance from a current cell to a different one, won't only affect the current part's situation but it will also have an effect on other part(s) occupying the same machine instance within the current cell. As already mentioned, part machine operation sequence forms a key constraint in the system, therefore, a machine of a specific type might be included within the sequence of other parts, besides the current one. For this reason when a single transition takes place, *part_moves_temp* is in question as it keeps a record of all parts machine instance cell allocation relative to each part's machine operation sequence. Moreover, for similar reasons with the *part_moves_temp*, *PCMATRIX_temp* will be affected as well.

Since an update on machine cell allocation has been already examined, a re-allocation of parts to machine cells is performed at this stage. Via the latter, updates are obtained immediately for both *PCMATRIX_temp* and *part_moves_temp* (see section 5.4) and possibilities of getting a better solution are highly increased. It is worth mentioning that initially, the algorithm was designed in such a way that updates to both *part_moves_temp* and *PCMATRIX_temp* were taking place after the transition without any reallocation of parts to machine cells. Although this update was operational there were less possibilities for the cost to be improved when compared with the value of cost produced when reallocation of parts to machine cells was occurring. Part reallocation acts as extra processing to the already allocated machines to cells, thus offering better chances for an improved output.

Having both machine cell allocation and part machine cell allocation updated, an evaluation of the value of the objective function occurs following the approach described in section 5.5. If the current objective value $OBJVAL\_temp$ is better than the best found so far, i.e. $BEST\_COST$, then the latter takes the value of $OBJVAL\_temp$ and best solution, $BEST\_SOL$, is replaced by current solution, SOL_temp. Also a number of other elements are stored before moving onto the next iteration and are shown below:

- $BEST\_CELLMATRIX2$ takes the value of $CELLMATRIX2\_temp$;
- $BEST\_PARTMATRIX$ takes the value of $PARTMATRIX\_temp$
- $BEST\_PCMATRIX$ takes the value of $PCMATRIX\_temp$;
- $BEST\_part\_moves$ takes the value of $part\_moves\_temp$.

Moreover, an update on another set of elements, whose role is to hold the best values in order to use them as a reference point from where the search will continue when objective value does not improve, takes place and is illustrated below:

- update $CELLMATRIX2\_temp0$ with $BEST\_CELLMATRIX2$;
- update $PARTMATRIX\_temp0$ with $BEST\_PARTMATRIX$
- update $PCMATRIX\_temp0$ with $BEST\_PCMATRIX$;
- update $part\_moves\_temp0$ with $BEST\_part\_moves$.

As already stated if the objective value does not improve the search continues from the situation where the best solution was found. Therefore all current/temporary values are replaced by the "0" temporary values shown above. For example, $CELLMATRIX2\_temp$ takes the value of $CELLMATRIX2\_temp0$ and so on for the rest of the elements. By doing this, the search process changes direction from a not very promising area to the point where the best solution was found.

- **Interchange**

The strategy employed for an interchange transition is similar to the single as it consists of two single transitions performed within two cells. When the interchange move occurs reallocation of parts to machine cells takes place together with the evaluation of the value of the objective function and updates on certain elements similar to the single transition.

It is worth presenting the process of the interchange transition as it has some special characteristics. As already mentioned, if a machine instance pair, (i, k), belonging to the sequence of the current part, is moved from source cell q to a destination cell q' then another machine instance pair (i', k') will be moved from the destination cell q' (of the first single move) to a source cell q. The question arising at this point is which machine instance pair to send back to the source cell as there might be machine instance pairs used by the current part as its sequence indicates.

For this reason a specific strategy was employed in order to have a donation of a machine instance pair from the destination cell back to source which is not required by current part. This is done in order to avoid unnecessary intercellular movements caused by the current part. The first step involved is to identify all machine instance pairs (including those used by the current part, if any) in destination cell q' via $CELLMATRIX2\_temp$ and store them in a vector named *machine_pairs_temp* (first index refers to machine type whereas second to its instance). Once this is achieved the segment corresponding to current part in *part_moves_temp* that holds information about machine pairs cell allocation is also identified and the machine pairs currently located in destination cell q' are marked and stored in a vector named *qik_receiver*. Later a comparison takes place between *qik_receiver* and *machine_pairs_temp*. Common pairs found are deleted from *machine_pairs_temp* so that no more elements used by current part exist in the latter. The first element found in *machine_pairs_temp* is donated to the source cell of the first single move. In case that *machine_pairs_temp* is empty, i.e. all machine instance pairs in destination cell are also used by the part in question then the process is interrupted, and a continuation from the outer loop where transition to a different cell (the next in sequence of *all_cells*) for the current machine instance pair is considered.

After this operation, the interchange transition leads to an update in the allocation of machine instance pairs to cells as follows:

$$CELLMATRIX2\_temp(i, k, q') = 1 \quad and \quad CELLMATRIX2\_temp(i, k, q) = 0$$

$$CELLMATRIX2\_temp(machine\_pairs\_temp(1, 1), machine\_pairs\_temp(1, 2), q) = 1$$

$$CELLMATRIX2\_temp(machine\_pairs\_temp(1, 1), machine\_pairs\_temp(1, 2), q') = 0$$

The above moves are implemented when the objective value improves. If the objective value does not improve, the next iteration commences but before the "0" temporary values are updated in a similar way to the single transition.

It is worth mentioning that in the majority of cases, interchange transitions are considered more for implementation rather than single. This is happening because of the available capacity of both source and candidate destination cells.

6.3.3 Ending Process

The algorithm stops when a maximum number of iterations has been reached. The latter could be determined based upon certain data and parameters involved within the heuristic pseudo code (see Routine 2 where all required outer loops are presented). An attempt was made to derive a formulae determining the total number of iterations required. This can be seen in equation (6.1).

$$Total\ \ Heuristic\ \ Iterations = \omega \times \{\lambda \times \sum_{j=1}^{NP}[length(qik(:,1)) \times (NC-1)]\} \quad (6.1)$$

The number of iterations depend upon the length of *qik* [3] which corresponds to the current part's cell machine instance allocation, the number of destination cells involved and the total number of parts in the system.

The above elements are also included within two loops the duration of which is defined explicitly via parameters λ and ω. The value of λ refers to the number of times the machine operation sequence of a certain part is considered, whereas ω to the number of times the parts listed in descending order relative to the intercellular movements caused are taking into account.

After certain experimentation it was decided for the majority of problems the values for ω and λ to be one and two respectively. ω set to one implies that the parts are considered only once and with the descending ordered sequence obtained from the initial solution. λ set to two entails the examination of possible transitions for current part with specific machine instance pair cell allocation relative to part machine sequence two times.

Before continuing with the testing of the heuristic approach it is useful to discuss the data sets generated since no available problem instances that could match the current

[3]The length of *qik* can be greater than the length of machine sequence denoted in *ZTYPES* for each part as multiple cells might be visited given a specific part machine operation sequence.

mathematical model requirements, by providing values of all the necessary parameters, can be found in the existing literature.

6.4 Generation of Data Sets

The proposed heuristic approach will be tested using a number of problem sets. The data elements required for each problem are: number of parts (NP) and number of machines (NM) involved, part/machine utilisation ($UTIL$), part/machine set-up costs ($SETUP$), number of operations required to produce each part ($ZOPER$), part machine operation sequence (L), number of multiple machine instances required by each machine ($KTYPES$), total number of machine instances (TMI), cells to be formed (NC), maximum (E_{MAX}) and minimum (E_{MIN}) number of machines allowed in each cell.

A number of data sets for small, medium and large data sets (twenty nine in total) were used in the test. The number of machines and parts involved in each of the data sets were adapted from problems previously used in the literature (except for problem instances twenty seven to twenty nine whose size was on purpose chosen to be four times six, in order to consider some very small problem instances). Tables 6.1 and 6.2 (pages 133 and 134) show the reference for each problem, the number of parts and the number of machines included in the system for medium to large and small data sets respectively.

Please note that, although, some of the problem sizes are used more than once they differ as their data each time are hypothetical and randomly generated. More specifically, part/machine utilisation, part/machine set-up costs and part machine operation sequences are randomly generated, whereas the number of multiple machine instances required by each part is found once the part/machine utilisation matrix is formed. Concerning the number of cells to be created in the system together with the upper bound, i.e the maximum number of machines allowed in each cell, are determined as soon as the total machine instances in the system are known given a certain problem size. The number of operations for each part is determined prior to the random data generation by the decision maker. For the purpose of this study the latter has been decided to take the value five for the majority of problem instances, i.e. a part can have up to five machines in its operation sequence in order to be fully processed. Please note that for problem instances twenty seven to twenty nine the number of operations was set to four since only four machine types are included in the system.

Table 6.1: Size Sources for Medium and Large Data Sets

Problem	Size Source	No. of Machine Types (NM)	No. of Parts (NP)
1.	[ASV97]	10	19
2.	[FFW06]	7	10
3.	[Sta85]	14	24
4.	[Sta85]	30	50
5.	[VN92]	15	30
6.	[VK86]	30	41
7.	[KKV86]	23	20
8.	[CR86a]	8	20
9.	[CM82]	10	15
10.	[CH81]	9	9
11.	[CH81]	9	9
12.	[KN82]	36	90
13.	[Kin80]	14	24
14.	[BC91]	20	35
15.	[ACGV91]	16	43
16.	[DW80]	12	19
17.	[Bur89]	16	43
18.	[MSW72]	27	27
19.	[ARS97]	26	37

It is worth providing at this stage some more information on the random data generation. For example, for each machine type required by each part, the utilisation amount was generated using a uniform distribution with parameters [0.0, 1.2] and rounded to the nearest integer. The resulting output was the creation of the $UTIL$ matrix of size $(NM \times NP)$. Also the set-up cost of each machine to be used by a certain part was also generated using a uniform distribution with parameters [0.00, 6.00] and rounded to the nearest integer, where the matrix $SETUP$ was formed of size $(NM \times NP)$ similar to $UTIL$.

For generating the machine operation sequence for each part in the system, the matrix L was initialised of size $NP \times max(ZTYPES)$. For identifying each part the machine operations involved, the indices of the non-zero elements in $UTIL$ matrix working column-wise were found and stored. Before adding the actual elements (corresponding to these indices) for each part as they appear to matrix L, an extra manipulation was carried for each machine sequence in order to perform some kind of perturbation and obtain a randomly generated sequence of machines. For doing the latter a `MatLab`

Table 6.2: Size Sources for Small Data Sets

Problem	Size Source	No. of Machine Types (NM)	No. of Parts (NP)
20.	[SK93]	6	8
21.	[SK93]	6	8
22.	[SK93]	6	8
23.	[SK93]	6	8
24.	[FFW06]	5	7
25.	[FFW06]	5	7
26.	[FFW06]	5	7
27-29.	N/A	4	6

function named *circshift*()[4] was employed. *shiftsize* was decided to be of a range type whose values were also randomly generated where two consecutive shifts were performed ensuring a random sequence. For more details on the actual code developed for the generation of random data involved, see Appendix C.

6.5 Computational Results for the Heuristic Approach

Testing the heuristic algorithm is very important in order to be able to identify its operational and computational advantage when compared with the output produced via the mathematical solver, i.e. XPRESS-MP, when both its behaviour and required CPU time are examined especially when large scale data sets are taken into account. Via this process, an opportunity is offered for discussing, for example, the behaviour of the initial starting solution, designed prior to the heuristic to act as a basic input into the latter, and also commenting on the actual heuristic and the number of iterations involved within the iterative procedure. Please note, that the heuristic algorithm presented in this study was coded in MatLab and run on a personal computer (Genuine Intel(R) 1.66 GHz, 1.00 GB of RAM), whereas the optimum or best known integer programming solutions, which required much longer runs, were obtained from the XPRESS-MP general purpose mathematical programming solver running on a Linux machine (Intel (P4 Xeon) 3GHz, 1.00 GB of RAM) and accessed remotely. It is also worth noting that for the initial solution which is fed into the heuristic, the second

[4]*circshift*: shift array circularly. More specifically, this function has the form:

$$B = circshift(A, shiftsize)$$

which implies: circularly shift the values in the array A by *shiftsize* elements. *shiftsize* is a vector of integer scalars where the N^{th} element specifies the shift amount for the N^{th} dimension of array A. If an element in *shiftsize* is positive, the values of A are shifted down (or to the right). If it is negative, the values of A are shifted up (or to the left).

approach (see section 5.4.3) designed to identify a cell sequence preserving a maximum continuous sequence of machines relative to part machine sequence is employed here since better results are produced.

For all problem instances involved the minimum number of machines in the system is defined prior to any work to be equal to four except for problem instance twenty seven whose value is set to three and for the last two small problem instances whose value is set to two. As the latter two mentioned problems are very small a way is needed to guarantee that at least three cells will be created so that both XPRESS-MP solver and heuristic approach to be of interest in the current study.

The medium and large data sets will be examined first concentrating on the computational results via which a number of elements will be discussed further such as: the behaviour of both the iterative procedure and the initial starting solution, the number of iterations considered in the system and the processing CPU times.

6.5.1 Computational Results for Medium and Large Sized Problems

The computational results for medium and large data sets are presented in Table 6.3 (page 137). The elements determined are: number of cells in the system, total number of machine instances, the best known objective value obtained (including the optimum when reached) via XPRESS-MP and the corresponding CPU time. In a similar way the best value of the objective function together with the value of the initial objective value obtained from the heuristic approach are also recorded. In addition, the required CPU time for only one run of the heuristic algorithm is also recorded. Please note that a number of runs [5] are tried for the heuristic algorithm which approximately take the same time, however only the results from the best run for each instance, including CPU time, are shown in the table. Moreover, a comparison is made between the value of the initial objective function and the best found and the percentage of improvement/deviation from the initial is included. In addition, the percentage deviation of each solution obtained via the heuristic approach from the optimum or best known obtained via XPRESS-MP is indicated. Moreover, the mean CPU time and the mean deviation values together with the mean number of heuristic algorithm runs are found and presented at the bottom of the table. Finally, the number of parts and machine

[5]Each run of the heuristic algorithm produces a different CF configuration since it is based on a random generated initial solution. As already stated in Chapter 5, machines are initially allocated to cells randomly and then parts are allocated to machines cells following a certain part before the iterative heuristic procedure commences, thus a different machine cell allocation leads to a different CF configuration.

types involved in each problem are also listed within Table 6.3 for completeness.

As can be seen from Table 6.3, and when XPRESS-MP is employed, the optimum solution is only found in two problem instances (two and sixteen) whereas for the rest of the instances the best known solutions [6] were recorded as no optimum values were found within a certain time limit. In current study this limit was set to be equal to thirty hours.

On the contrary, the results from the heuristic algorithm were more promising since for almost half of the problem instances the percentage deviation of the heuristic from the best known solutions was small in magnitude and the required CPU time for most of them did not exceed six hundred seconds overall (problem twelve is currently excluded, as been the largest data of all needs a substantial CPU time in order for the iterative procedure to end). More specifically, some small deviations in magnitude obtained are three, five, seven, eight and nine.

Moreover, the deviation percentage between the initial objective value obtained, when the CF initial solution was employed, and the best cost found, via the iterative process, is for the majority of problems substantially significant meaning that a large increment in the value of objective function is taking place via the iterations. This implies that via the iterative heuristic approach a number of machine instance pair transitions is considered with respect to the number of iterations involved leading to the investigation of better solutions. The trend on the objective values involved will become clearer in the section where the behaviour of the heuristic approach is examined.

In addition, it is worth commenting on problems fourteen, eighteen, and nineteen which can be classified as large sized problems whose percentage deviation is estimated to be three, three and seven percent respectively from the best known values and required CPU time less than four hundred seconds.

[6]Please note that when the best known solutions were recorded the optimality gap indicated via XPRESS-MP it was still large which does not necessarily imply that solutions were still far from the optimum as lots of nodes could turn out to be of no interest. Thus it was decided not to tabulate the gap as it was large.

Table 6.3: Problem Data and Computational Results for Medium & Large Sized Problems

						XPRESS-MP		Heuristic Algo.					
Prob.	NM	NP	NC	TMI	E_{MAX}	Obj.	CPU time	Init. Obj.	Best Obj.	Dev.†††	CPU time (secs)	Runs	Dev.††
1.	10	19	5	40	9	444.13†	> 30 hrs	664.89	484.93	27%	47.281	4	9%
2.	7	10	5	20	6	219.92*	141 secs	312.35	243.51	22%	4.9375	5	11%
3.	14	24	5	37	8	383.78†	> 30 hrs	590.05	428.4	27%	34.188	10	12%
4.	30	50	8	99	13	1100.02†	> 30 hrs	1697.8	1207	29%	758.48	9	10%
5.	15	30	7	47	7	582.71†	> 30 hrs	948.13	659.99	30%	123.5	10	13%
6.	30	41	7	94	14	893.28†	> 30 hrs	1452.8	972.26	33%	529.09	6	9%
7.	23	20	6	42	8	390.51†	> 30 hrs	645.63	411.51	36%	47.625	4	5%
8.	8	20	5	31	7	333.82†	> 30 hrs	606.36	395.72	35%	25.094	11	19%
9.	10	15	5	30	7	302.25†	> 30 hrs	529.77	342.82	35%	20.578	7	13%
10.	9	9	5	29	6	242.13†	> 30 hrs	424.3	285.14	33%	18.578	12	18%
11.	9	9	5	28	6	269.11†	> 30 hrs	424.93	304.74	28%	17.219	5	13%
12.	36	90	10	147	17	1441.62†	> 30 hrs	2853.2	1912.4	33%	2839.6	15	33%
13.	14	24	5	45	10	508.82†	> 30 hrs	822.92	557.96	32%	68.063	5	10%
14.	20	35	7	64	11	732.37†	> 30 hrs	1090.6	754.7	31%	174.41	2	3%
15.	16	43	9	65	11	799.08†	> 30 hrs	1307.5	862.51	34%	281.42	3	8%
16.	12	19	5	29	7	293.85*	196 secs	386.65	322.79	17%	14.344	8	10%
17.	16	43	8	61	10	713.97†	> 30 hrs	1147.3	851.39	26%	182.88	12	19%
18.	27	27	8	76	10	763.51†	> 30 hrs	1212.8	787.65	35%	354.89	2	3%
19.	26	37	8	64	10	644.41†	> 30 hrs	1125	689.24	39%	203.11	5	7%

Mean Rounded Values			
Dev.	CPU	Runs	Dev.
31%	302 secs	7	12%

* Optimum solution.

† Best known solution.

†† Percentage deviation of each solution from the optimum or best known solution, obtained via XPRESS-MP, and produced as follows:

$Dev. = round((\frac{Heur.\ Best\ Obj. - XPRESS\text{-}MP\ Best\ Obj.}{XPRESS\text{-}MP\ Best\ Obj.}) \times 100)$

††† Heuristic Percentage of improvement: Init. Obj. vs. Best Obj. as follows:

$Dev. = round((\frac{Init.\ Obj. - Best\ Obj.}{Init.\ Obj.}) \times 100)$

Some additional observations, involve problem instances eight and seventeen whose deviation from the best known solution for both is nineteen percent. Although there are other problems which are of a similar size and of similar resulting deviation, these two have a similarity in their behaviour for a certain reason. More specifically, for both of these problems the number of machine types included are eight and sixteen, whereas their complementary parts are recorded to be twenty and forty three respectively. Therefore there are only a few machine types to be used by many parts, or in other words the majority of the machine types have a high probability to be included within the machine operation sequence of each part. The latter could cause a difficulty to the iterative procedure to converge to a value close to the best known as at each time that a transition takes place for a machine instance pair of a certain part, a number of other parts using this machine instance pair is also affected since the cell allocation of the latter could change causing possibly an additional intercellular movement. This might also occur in other problem instances with similar resulting deviations from the best known value found.

• Heuristic Iterative Procedure Behaviour

The heuristic approach commences once the initial random solution is obtained. Within the actual iterative process a number of transitions, either single or interchange, are taking place for each part in the system. During this procedure the objective value evolves with respect to the specified number of iterations, and a number of significant fluctuations to the former is observed. At iteration zero the objective value is assumed to be equal to the value of the objective function obtained via the initial solution. For illustrating the trend on the objective value, problem instance nineteen on Table 6.3 (page 137) is employed and the results are displayed in Figure 6.1 (page 139).

Although this problem involves a significant number of iterations, an attempt to display the objective function values corresponding to the iterations is made and the result is shown on Table 6.4 (page 140). Please note that some of the iterations are not shown due to the lack of space.

Both Table 6.4 and Figure 6.1 show that the number of iterations involved is 1188. The best objective, as recorded in Table 6.3, has a value of 689.24 units which was first found at iteration number 925 and also in other iterations later on. In Figure 6.1 this value is the closest to the iterations axis. It is also worth commenting on the value of the initial objective value recorded at iteration zero and the level of improvement through the iterative procedure. The deviation between initial and best objective found

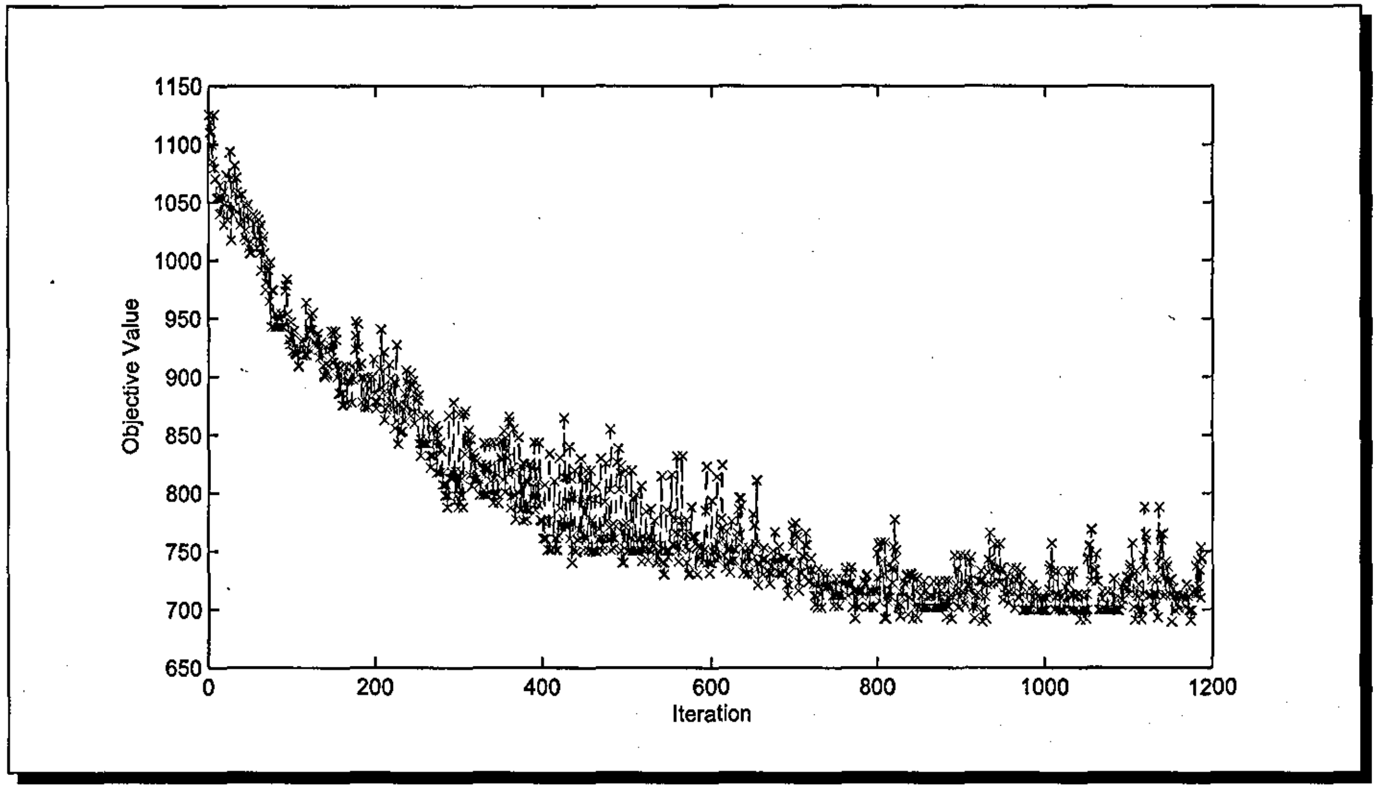


Figure 6.1: Problem instance 19: Objective Function Value Evolution

Table 6.4: Objective Function Value Evolution for Problem 19

Iter.	Obj.	Iter.	Obj.	Iter.	Obj.	Iter.	Obj.	Iter.	Obj.	Iter.	Obj.	Iter.	Obj.	Iter.	Obj.	Iter.	Obj.	Iter.	Obj.
0	1125	48	1010.6	96	931.66	144	910	192	873.18	240	891.48	900	715.84	948	735.18	996	709.24	1141	764.9
1	1109.8	49	1005.8	97	927.57	145	923.04	193	888.85	241	867.75	901	713.85	949	707.82	997	699.24	1142	726.84
2	1119.8	50	1015	98	931.66	146	926.66	194	898.4	242	882.28	902	705.85	950	713.03	998	699.24	1143	712.65
3	1110.8	51	1005.8	99	946.57	147	938.84	195	881.8	243	902.28	903	703.85	951	717.82	999	699.24	1144	741.11
4	1099.4	52	1010.6	100	921.66	148	916.66	196	899.4	244	892.28	904	723.85	952	727.82	1000	709.24	1145	722.65
5	1085.1	53	1039.4	101	921.66	149	936.66	197	899.4	245	892.28	905	746.41	953	717.82	1001	699.24	1146	736.84
6	1125	54	1031.2	102	939.74	150	912.15	198	914.73	246	896.29	906	713.85	954	727.82	1002	709.24	1147	724.78
7	1079.2	55	1020.8	103	920.66	151	938.65	199	887.23	247	859.6	907	719.73	955	714.99	1003	713.03	1148	712.65
8	1070.2	56	1037.8	104	919.66	152	909.65	200	877.97	248	869.6	908	715.46	956	704.99	1004	716.82	1149	712.65
9	1070.2	57	1016.3	105	918.48	153	931.83	201	872.32	249	889.6	909	719.73	957	715.24	1005	734.35	1150	725.66
10	1053	58	1010.1	106	923.39	154	899.65	202	877.97	250	879.6	910	739.73	958	725.24	1006	737.82	1151	689.24
11	1054.3	59	1034.6	107	909.48	155	909.65	203	877.97	251	879.6	911	725.46	959	725.24	1007	727.82	1152	718.02
12	1054	60	1026.5	108	908.35	156	885.79	204	881.98	252	883.61	912	744	960	735.24	1008	756.47	1153	711.37
13	1039.8	61	1005.8	109	923.62	157	907.79	205	906.37	253	842.19	913	701.67	961	714.24	1009	732.59	1154	699.24
14	1064.7	62	1029.8	110	930.53	158	885.79	206	890.4	254	832.19	914	701.67	962	714.24	1010	699.24	1155	699.24
15	1040.8	63	990.69	111	923.5	159	907.97	207	940.24	255	846.46	915	691.67	963	700.49	1011	700.24	1156	712.25
16	1054.7	64	1015.8	112	921.34	160	875.79	208	889.85	256	842.19	916	711.67	964	725.24	1012	710.24	1157	704.61
17	1053	65	1020	113	918.05	161	885.79	209	886.07	257	842.19	917	711.67	965	725.24	1013	711.59	1158	712.65
18	1030.8	66	1005.8	114	931.79	162	875.79	210	920.52	258	866.85	918	701.67	966	735.24	1014	713.41	1159	700.37
19	1059.2	67	996.84	115	918.35	163	904.44	211	862.32	259	842.19	919	713.54	967	710.24	1015	732.59	1160	710.37
20	1049.1	68	974.16	116	918.35	164	896.79	212	872.32	260	842.19	920	723.54	968	710.24	1016	699.24	1161	710.37
21	1073.6	69	980.69	117	963.09	165	908.97	213	871.8	261	846.46	921	723.54	969	710.24	1017	700.24	1162	710.37
22	1034.8	70	990.69	118	918.35	166	877.07	214	877.07	262	842.19	922	723.54	970	732.37	1018	710.24	1163	710.37
23	1046.4	71	996.33	119	933.26	167	896.79	215	887.07	263	842.19	923	733.08	971	712.41	1019	711.59	1164	712.65
24	1043.8	72	990.69	120	923.01	168	896.87	216	910.08	264	866.85	924	723.62	972	723.41	1020	713.41	1165	711.42
25	1069.2	73	964.16	121	939.91	169	904.44	217	872.32	265	832.19	925	689.24	973	699.24	1021	698.52	1166	703.42
26	1093.1	74	998.07	122	939.91	170	896.79	218	898.32	266	822.19	926	699.24	974	699.24	1022	698.52	1167	701.42
27	1016.5	75	942.4	123	952.09	171	908.97	219	877.59	267	832.19	927	699.24	975	699.24	1023	709.24	1168	721.42
28	1070.7	76	974.16	124	939.91	172	877.07	220	878.07	268	832.19	928	710.37	976	699.24	1024	719.24	1169	711.42
29	1038.7	77	974.16	125	954.82	173	896.79	221	872.32	269	832.19	929	691.67	977	704.15	1025	702.98	1170	711.42
30	1045	78	974.16	126	929.91	174	896.87	222	895.33	270	855.2	930	720.37	978	715.15	1026	722.98	1171	699.24
31	1069.1	79	942.4	127	930.33	175	923.04	223	855.06	271	842.19	931	732.42	979	699.24	1027	732.59	1172	699.24
32	1081.1	80	944.4	128	929	176	934.66	224	894.06	272	858.19	932	730.8	980	699.24	1028	699.24	1173	690.24
33	1040.8	81	954.13	129	926.39	177	946.84	225	926.77	273	817.4	933	742.42	981	699.24	1029	700.24	1174	699.24
34	1070.7	82	942.4	130	932.01	178	903.94	226	862.32	274	846.94	934	765.43	982	699.24	1030	710.24	1175	714.61
35	1058.7	83	952.4	131	936.92	179	944.66	227	842.19	275	853.55	935	721.42	983	704.15	1031	711.59	1176	714.06
36	1055	84	942.4	132	923.01	180	925.01	228	876.07	276	817.4	936	739.69	984	715.15	1032	713.41	1177	717.3
37	1029.8	85	942.4	133	916.35	181	908.3	229	876.06	277	817.4	937	740.65	985	720.97	1033	732.59	1178	713.03
38	1030.8	86	944.4	134	927.35	182	898.65	230	852.19	278	843.4	938	722.68	986	699.24	1034	699.24	1179	717.3
39	1055.7	87	954.13	135	929.53	183	910.83	231	852.44	279	835.84	939	732.68	987	712.97	1035	700.24	1180	737.3
40	1056.7	88	942.4	136	918.35	184	877.93	232	871.49	280	807.4	940	755.69	988	709.24	1036	710.24	1181	713.03
41	1035.7	89	952.4	137	909.43	185	898.65	233	853.32	281	817.4	941	732.68	989	714.98	1037	711.59	1182	741.57
42	1047.2	90	942.4	138	901.65	186	898.65	234	876.33	282	807	942	734.68	990	710.89	1038	713.41	1183	719.56
43	1019.5	91	945.57	139	899.65	187	875.79	235	885.37	283	798	943	723.18	991	699.24	1039	699.24	1184	733.35
44	1025.6	92	973.57	140	925.18	188	873.18	236	861.69	284	798	944	723.18	992	699.24	1040	699.24	1185	710.04
45	1016.5	93	978.48	141	910.65	189	877.97	237	905.37	285	807	945	723.18	993	699.24	1041	699.24	1186	753.35
46	1037.5	94	983.57	142	901.65	190	892.72	238	895.37	286	788	946	756.19	994	709.24	1042	691.24	1187	743.35
47	1047.4	95	953.57	143	903	191	899.33	239	874.37	...	...	947	733.18	995	699.24	...	....	1188	744.75

for problem instance nineteen is about thirty nine percent which is quite significant. Within the iterative procedure a number of fluctuations in the objective values are observed with the most significant happening when after finishing with the current part and all the possible transitions for the machine instance pairs belonging to its machine operation sequence, the next part in the sequence is considered. When a new part is considered with another machine operation sequence the future moves could be more promising for investigating unexplored areas since new machine instance pairs are now under consideration. The latter can be more efficient if each part's machine operation sequence is in its majority different from sequences of other parts in the system. In the current study the latter cannot be guaranteed since all data are randomly generated, which of course makes the system realistic.

• **Heuristic Approach vs. Initial Solution**

At this stage it is important to refer to the initial solution and how this affects the heuristic algorithm's operation and the best solution found. The strategy employed for the generation of the initial solution consists of the allocation of machines to cells the allocation of parts to machine cells and the evaluation of the value of the corresponding objective function. As can be seen for problem nineteen in Table 6.4, the initial value is 1125. Feeding this solution into the iterative heuristic results in a best cost of magnitude 689.24 units. The trend of the objective value as shown in Figure 6.1 could change if the number of iterations are of different size. For example some more repetition of values might occur if the inner loop lasts for longer and also some new values might be produced as additional areas might be investigated. Overall though, this won't affect much the final best cost as a value of similar or of the same magnitude with the previous will be obtained. However, if the initial solution changes, the picture produced from the heuristic approach will be different. For illustration purposes of the latter see the objective value fluctuation in Figure 6.2 (page 142) when a different initial solution is generated and fed into the iterative procedure. Also the number of iterations for the inner and outer loop are defined to be of the same size as before. Please note that the total number of iterations might differ slightly from the total number as shown on Table 6.4, and this is happening due to the different part machine cell allocation stored in *qik* examined within the inner loop.

As shown in Figure 6.2 the best cost produced is 700.85 units whereas the initial cost is 1064.2 units. Moreover, the deviation from the best known solution is recorded to be nine percent. Overall, a different initial solution produces a different cell formation configuration. Since the initial solution is randomly generated a few trials are needed till the iterative procedure in conjunction with its input part machine cell allocation work efficiently and the least possible deviation from the best known solution is found.

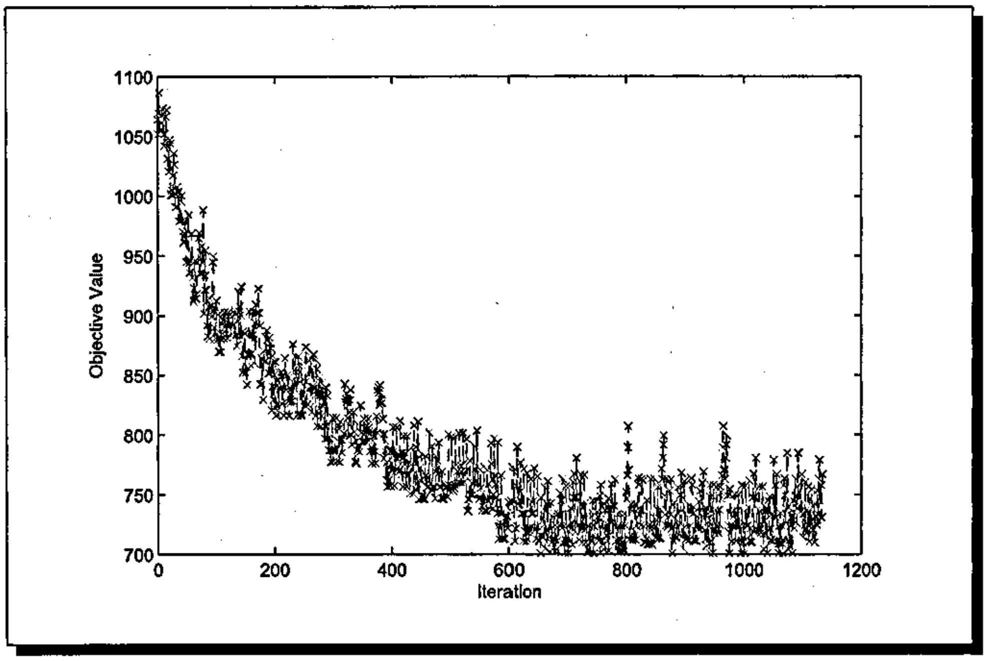


Figure 6.2: Problem 19: Objective Function Value Evolution when Initial Solution Differs

- **Additional Trends in the Objective Values**

For more information on the overall trend of the objective values for other problem instances, see Figures 6.3 and 6.4 (pages 143 and 144 respectively). Figure 6.3 refers to objective values fluctuation of problem instance eighteen and Figure 6.4 to problem instance six. Due to the fact that both problems, eighteen and six, involve large scale data a significant number of iterations will be involved. As can be seen from Figure 6.3 the total number of iterations for problem instance eighteen is 1750, whereas from Figure 6.4 the number of iterations involved for problem instance six are more than 1908. For both problems the cost of the initial solution starts at a high magnitude level and then as the iterations progress it gradually begins descending, arriving at the lower cost level (best cost). The latter is obtained almost half way through the iterations involved. Moreover, the fluctuations in the value of the objective function continue indicating that the transitions considered lead to the investigation of many areas with new solutions, from which only those producing better costs are only implemented.

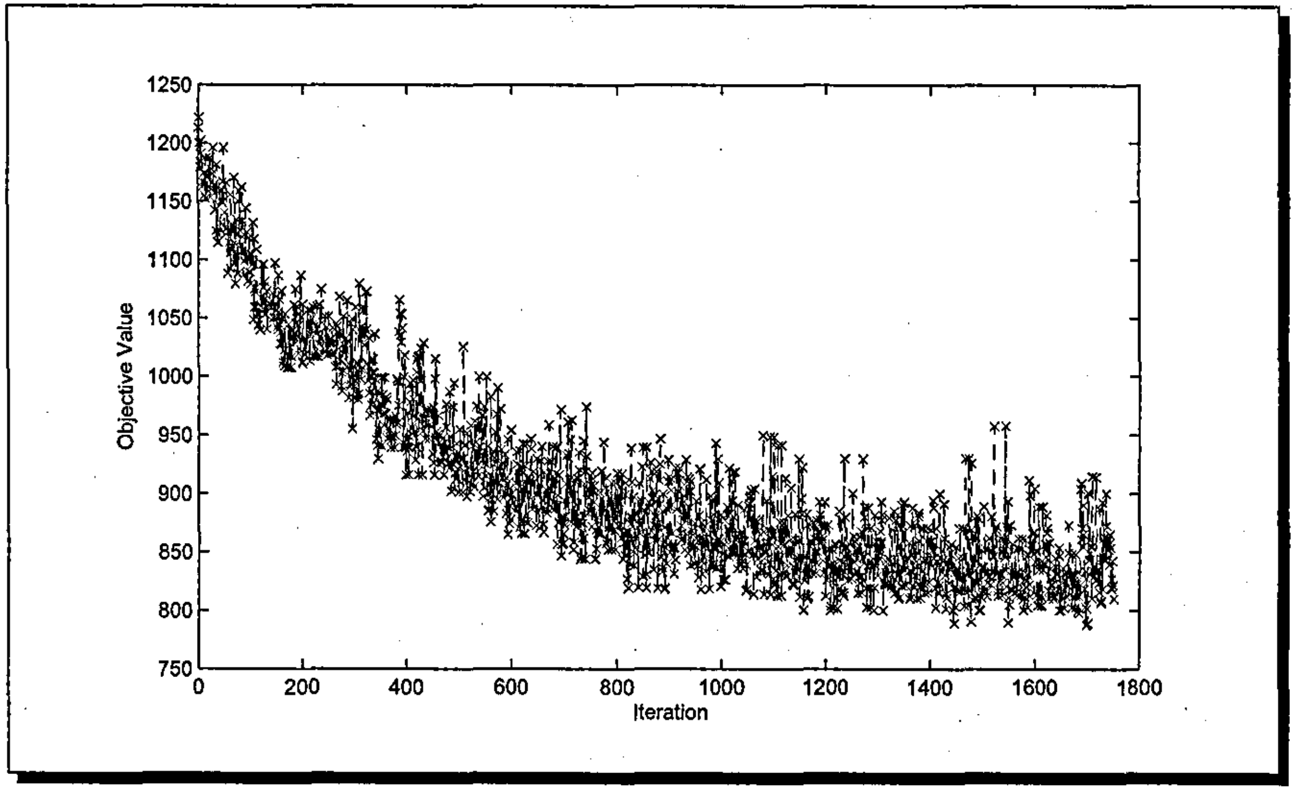


Figure 6.3: Problem instance 18: Objective Function Value Evolution

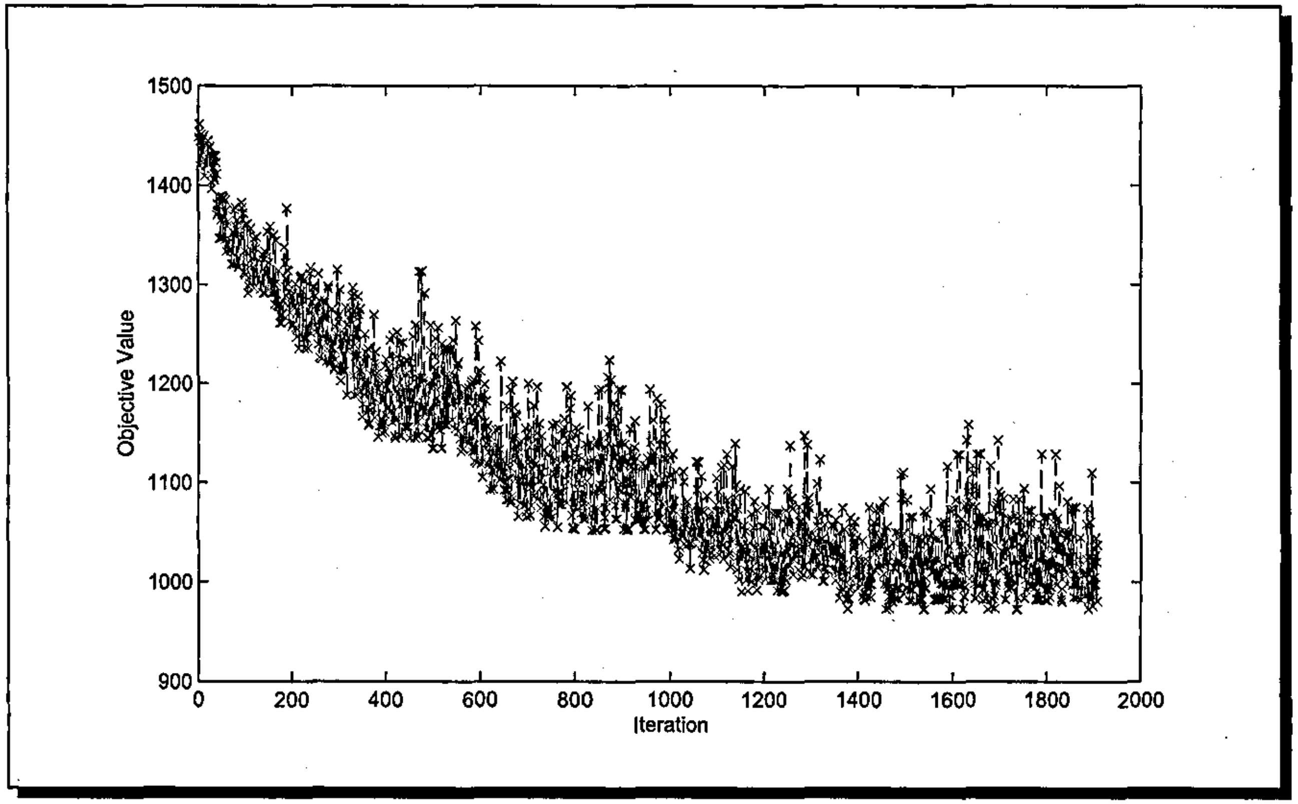


Figure 6.4: Problem instance 6: Objective Function Value Evolution

6.5.2 Computational Results for Small Sized Problems

For the small sized problem instances, presented earlier in Table 6.2, their corresponding computational results are shown in Table 6.5 (page 146). In a similar way to the large problem instances a number of elements are included here as well such as: number of parts and machines types, number of cells, total number of machine instances and the maximum value of machines allowed in each cell. For the mathematical solver the best solution found and its required running time are displayed. For the heuristic approach, values like the initial and the best cost found, together with the required CPU time for the best run [7] are also recorded. Finally, the mean values for some of the elements as in large problem instances are added here for completeness.

An important observation that could be made, and which is something to be expected since small problems are employed at this stage, is that for all of them XPRESS-MP finds the optimum value. For almost half of the problems the latter is obtained in less than a hundred seconds, whereas for the rest a significant amount of CPU time is required. For example, problem instance twenty needs more than twenty three hours to find the optimum for a data size of six machines and eight parts. On the other hand, the heuristic approach finds the optimum in two data sets of small size and the mean value of best solution obtained overall deviates by eight percent from the optimum. Moreover, the average value of the required CPU time within the iterative procedure does not exceed forty seven seconds.

Additional comments can be made regarding the overall improvement in the value of the initial objective function made within the iterative procedure. The mean deviation for the value of the best objective function, found via the heuristic approach, from the initial cost is twenty four percent. This implies that in the majority of cases the level of improvement is significant. For example, for problem instance twenty one, forty five percent is the deviation of the best cost from the initial value of the objective function, achieved in approximately four seconds of CPU time.

The ideal for the heuristic approach would be for the majority of cases to reach the optimum solution. Although the data sets involved here are not large, the aim is unreasonable since the part machine operation sequence is included adding a significant constraint to the system. As stated before, but here it is easier to observe since the

[7]In contrast with the medium and large sized problem instances, only a few runs were attempted for the small problem instances. More specifically, an average of three to four runs were generated for each and the CPU time of the best run was recorded.

Table 6.5: Problem Data and Computational Results on Small Problems

						XPRESS-MP		Heuristic Algo.				
Problem	NM	NP	NC	TMI	E_{MAX}	Opt.*	CPU time	Init. Obj.	Best Obj.	Dev.††	CPU time (secs)	Dev.†
20.	6	8	5	22	6	213.70	23.80 hrs	257.85	231.71	10%	8.2969	8%
21.	6	8	4	19	6	143.04	71 secs	282.97	155.58	45%	3.7813	9%
22.	6	8	5	23	6	204.25	3210 secs	314.96	211.63	33%	8.625	4%
23.	6	8	5	28	6	229.09	4167 secs	354.61	244.5	31%	11.75	7%
24.	5	7	3	17	6	153.36	52 secs	177.59	164.7	7%	2.7031	7%
25.	5	7	4	20	6	183.14	2871 secs	289.71	201.56	30%	5.2188	9%
26.	5	7	3	17	6	153.19	25 secs	195.85	164.7	16%	2.9219	8%
27.	4	6	3	13	5	117.82	1 sec	155.76	132.69	15%	1.6563	13%
28.	4	6	3	10	4	113.26	16 secs	157.73	113.26	28%	1.1875	0%
29.	4	6	3	9	4	103.08	1 sec	140.05	103.08	26%	1.0313	0%
										Mean Rounded Values		
										Dev.	CPU	Dev.
										24%	47 secs	7%

* Optimum solution obtained via XPRESS-MP.

† $Dev. = round((\frac{Best\ Obj. - Opt.}{Opt.}) \times 100)$

†† TS Percentage of improvement: Init. Obj. vs. Best Obj. as follows:

$Dev. = round((\frac{Init.\ Obj. - Best\ Obj.}{Init.\ Obj.}) \times 100)$

parts and the machines involved are only a few, every time a transition takes place within the iterative procedure the majority of the parts get affected leading to an update of their cell machine instance pair allocation. Additionally, part reallocation takes place after the latter update so that a better reconfiguration of the system could be obtained.

For illustration purposes on the trend of the objective values involved, while transitions take place, problem instance twenty will first be considered. Table 6.6 shows the objective function values corresponding to the heuristic iterations and Figure 6.5 (page 148) this data set plotted.

Table 6.6: Objective Function Value Evolution for Problem Instance Twenty

Iteration	Objective	Iteration	Objective	Iteration	Objective	Iteration	Objective
0	257.85	28	245.68	56	277.6	84	231.71
1	341.43	29	277.6	57	258.21	85	320.11
2	280.89	30	258.21	58	238.68	86	284.03
3	304.64	31	258.12	59	244.76	87	231.71
4	241.71	32	242.94	60	259.62	88	286.03
5	268.08	33	297.32	61	265.08	89	283.98
6	289.46	34	265.08	62	257.08	90	283.14
7	257.73	35	257.08	63	287.46	91	289.61
8	335.63	36	287.46	64	233.71	92	245.64
9	262.83	37	241.82	65	245.68	93	321.51
10	271.74	38	240.9	66	259.79	94	289.61
11	233.71	39	262.28	67	264.71	95	245.64
12	282.25	40	231.71	68	256.2	96	321.51
13	231.71	41	241.71	69	279.57	97	266.64
14	245.68	42	277.33	70	248.72	98	281.65
15	259.79	43	245.68	71	269.94	99	267.83
16	264.71	44	277.6	72	256.57	100	262.64
17	256.2	45	258.21	73	248.72	101	301.53
18	279.57	46	258.12	74	269.94	102	252.83
19	262.64	47	242.94	75	256.57	103	231.71
20	301.53	48	297.32	76	260.53	104	289.52
21	252.83	49	241.82	77	265.78	105	267.79
22	231.71	50	240.9	78	285.13	106	231.71
23	289.52	51	262.28	79	289.61	107	289.52
24	267.79	52	265.08	80	245.64	108	267.79
25	231.71	53	257.08	81	321.51		
26	241.71	54	287.46	82	320.11		
27	277.33	55	245.68	83	284.03		

As can be seen the total number of iterations involved is only 108 and the best value of the objective function is first found at iteration number 13. Also the deviation of the best value from the initial value is ten percent. The values of the objective functions range between the best value to others of higher magnitude of the initial value of the objective function. The latter can easily be seen in Figure 6.5.

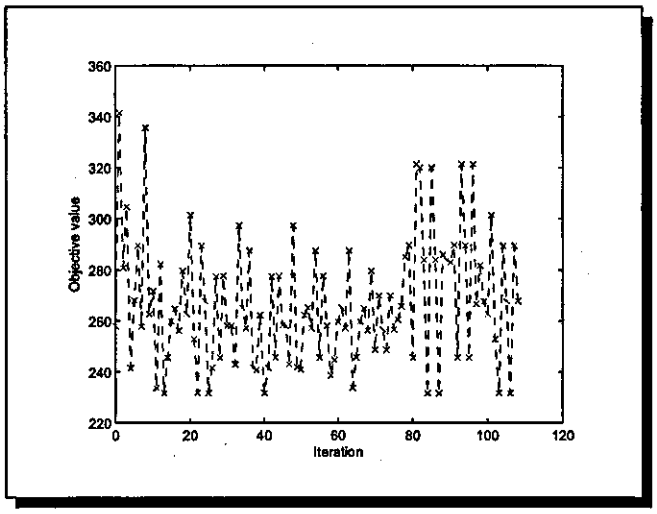


Figure 6.5: Problem instance 20: Objective Function Value Evolution

6.6 Concluding Remarks - Heuristic Algorithm Limitations

Although the heuristic approach has proved to be very effective for a variety of different sized problem instances, it has certain limitations. A number of conclusions including specific heuristic procedure restrictions are drawn at this point and listed below.

- Overall the iterative heuristic approach produce good solutions, in reasonable computational times, with deviations of small magnitude from the best known solutions for the majority of the problem instances as shown in Tables 6.3 and 6.5 respectively;

- In order to obtain a deviation value of small magnitude from the best known or optimum solution a number of running trials are needed for the heuristic approach, as shown especially for medium and large scale data sets (Table 6.3). The latter implies that the iterative heuristic approach won't produce a good solution unless the randomly generated starting solution, i.e. the allocation of machines to cells, for the current problem 'suits' the iterative procedure thus, leading to the investigation of the smallest deviation value within the pre-specified iteration limit for the heuristic approach;

- The reason for the above is due to the heuristic algorithm operation. The iterative procedure follows a certain path of exploration, i.e. at each step, where a new transition is taken into account, the iterative procedure continues from the best solution found so far without allowing a number of other areas which might not be very promising within the search space to be investigated, thus the search is limited;
- The iterative heuristic procedure could get stuck to a local optimum especially when a large sized problem with complex part machine operation sequence is taken into account;
- Last but not least, the part machine operation sequence has been identified as a key constraint within the mathematical model in previous chapters. Its operation within the iterative heuristic approach fulfills the latter, since every operation depends upon this sequence and certain elements need to get updated, e.g. the part machine cell allocation relative to part machine sequence is updated everytime a transition, either single or interchange is implemented.

6.7 Summary

This chapter presented an iterative heuristic approach for the CF problem. The central idea of the heuristic was mainly based on the operation of a descent algorithm where a number of solutions was obtained via single or interchange transitions of machine instance pairs among cells. At each stage the value of the objective function was evaluated; if this was better than the global best then the iterative procedure continued from current solution else the global best was recalled. The computational results for this iterative procedure when a number of problem instances were generated proved to be very promising, however certain limitations were identified. In order to overcome these limitations another approach, based on the principles of an already established methodology, will be proposed for the CF and presented next in Chapter 7.

Chapter 7

A Tabu Search Algorithm in the Presence of Part Machine Sequence

7.1 Introduction

In the current chapter an extension to the heuristic algorithm described in Chapter 6 will be proposed for searching the space on a different level. In the last twenty years, a new kind of approximate algorithm has emerged which tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring the search space. These methods are nowadays commonly called *metaheuristics.* The term *metaheuristic,* first introduced by Glover [Glo86], comes from the composition of two Greek words; *heuristic,* which derives from the verb "heurísko" (ευρίσxω) that means "to search" and *metá* (μετά) which means "beyond on a higher level". The employment of a metaheuristic algorithm will prove to be promising and very effective for the CF problem.

7.2 Metaheuristics

In general, there is no commonly accepted definition of the term metaheuristic. However, some researchers in the field tried to propose a definition a few years ago. Two definitions of interest are quoted below.

Osman and Laporte [OL96] wrote:

"A metaheuristic is formally defined as an iterative process which guides a

> subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."

Voß *et al.* [VMOR99] stated:

> "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution at each iteration. The subordinate heuristics may be a high (or low) level procedures, or a simple local search, or just a construction method."

Over the past two decades, metaheuristics have been particularly popular and effective in obtaining solutions to the CF problem. Some of the most popular metaheuristic approaches employed, as already presented in Chapter 2, are Simulated Annealing (SA), Genetic Algorithms (GAs), Tabu Search (TS) and Ant Colony Optimisation (ACO).

Common fundamental properties which characterize all the above approaches are as follows:

- Metaheuristics are strategies that "guide" the search process;
- The goal is to efficiently explore the search space in order to find (near) optimal solutions;
- Techniques implemented could range from simple local search procedures to complex learning processes;
- They may incorporate mechanisms to avoid getting trapped in local minima;
- The basic concepts of metaheuristics permit an abstract level of description;
- Metaheuristics are not problem specific.

For the purpose of this research Tabu Search method is chosen for the following reasons [Glo86]:

- Tabu search is a meta-heuristic that has the ability to overcome local optimality;
- This strategy unlike simulated annealing has no stochastic elements;
- Tabu search combines the aggressiveness of descent methods and the diversity (the ability to explore the solution space extensively) of simulated annealing;

- Tabu search seems to have more success than GAs in building and improving solutions when there are many feasible solutions;
- Tabu search allows considerable freedom to implement refinements of this method; there is no standard way of implementing tabu search;
- Tabu search can be embedded within an existing model formulation by amending and not altering the system's requirements;
- Last but not least tabu search methods have been applied successfully for finding optimum or near optimum solutions to a number of combinatorial optimisations problems such as planning and scheduling, telecommunications, parallel computing, transportation routing. For a thorough list of applications the reader is referred to Glover and Laguna [GL97].

7.2.1 Tabu Search

Tabu search is one of the most successful metaheuristics for application to combinatorial problems and it can be viewed as a metaheuristic superimposed on another heuristic. The basic ideas of TS were introduced by Glover [Glo86]. A description of the method and its concepts can be found in Glover and Laguna [GL97], Glover [Glo89a], and [Glo89b]. The basic idea of TS is the explicit use of search history, both to escape local minima and to implement an explorative strategy.

In general terms, a TS algorithm is an iterative search that starts from some initial feasible solution randomly set and determined by the decision maker strictly following all problem specifications/constraints. Further, it attempts to determine a better solution in the manner of a hill-climbing algorithm similar to the policy of LC, described in Chapter 6, but goes beyond that paradigm. At each step the tabu method functions with only one "current configuration" (at the beginning, any solution), which gets to be updated at each iteration. In each iteration, the "mechanism of a passage" of a configuration, called x, to the next one called y, comprises the following two stages:

- the construction of the *neighbor* set of x, i.e. the set of all accessible configurations from only one elementary movement of x (if a problem is too large only a subset of neighbors is defined). Let N_s be the set of these neighbors;
- the evaluation of the objective function f for each configuration belonging to N_s. A configuration y can succeed in the series of solutions built by the tabu method if the configuration of N_s takes the minimal value f. If for a number of iterations

f does not improve then a configuration y is selected as the current best even if $f(y) \geq f(x)$; the latter operation is adopted by the tabu method in order to avoid f getting trapped in a local minimum.

Moreover, TS employs a *short term memory* which maintains the information about the past step of the search and uses it to create and exploit better solutions. The following description presents the short term memory's operation via its components.

- Accepting a best configuration created from a move at any one time there is a significant risk to return to a configuration already retained at the time of a precessing iteration, which generates a cycle. In order to prevent cycling, it requires updating at each iteration a list of prohibited moves, the tabu list (TL). TL stores all the tabu moves that cannot be applied to the current solution i.e. it stores m movements $(y \rightarrow x)$, which are the opposite of the last m movements $(x \rightarrow y)$ carried out.

- The number of iterations a move is kept to be tabu is called the tabu tenure (TT). In other words, TT is the length of the TL. TT controls the memory of the search process and can be either fixed or it can vary.

- Another key issue of TS arises when a move under consideration to be implemented has been found to be tabu. The tabu status of a move is not absolute, but can be overruled if certain conditions are met, expressed in the form of an aspiration criterion (AC). If appropriate AC is satisfied the move will be accepted despite tabu classification. Roughly speaking, AC is designed to override tabu status if a move is "good enough" [Glo89a]; e.g. if the objective value of a candidate move, which happens to be tabu, is better than the current best stored then tabu status of the candidate will be overruled and accepted as the current best.

- Last but not least, there may be several possible stopping conditions for the search. For example, a common criterion for terminating a tabu search algorithm is when the maximum allowed number of iterations is reached or an optimal solution is found.

A flowchart of a "simple" tabu search algorithm can be seen in Figure 7.1.

Additional mechanisms, named *intensification* and *diversification* are often implemented to also equip the algorithm with a *long term memory* [GL97], [BR03]. This process does not exploit further the temporal proximity of particular events, but rather the frequency of their occurrence over a longer period. The intensification consists in

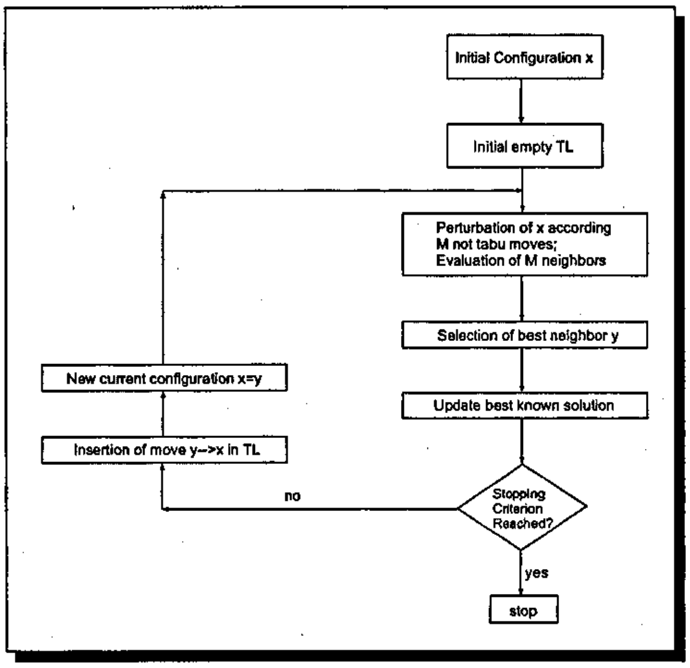

Initial Configuration x
Initial empty TL
Perturbation of x according M not tabu moves; Evaluation of M neighbors
Selection of best neighbor y
Update best known solution
Stopping Criterion Reached?
yes
stop
no
Insertion of move y-->x in TL
New current configuration x=y

Figure 7.1: Flowchart of a "simple" tabu search algorithm

looking further into the exploitation of certain areas of the solution space, identified as particularly promising ones. Diversification is on the contrary the periodic reorientation of the search to explore areas rarely visited until now.

The basic form of TS where short term memory is only employed comprises fewer parameters of adjustment which makes it easier to use. In contrast, long term memory mechanisms bring a notable complexity.

- **Tabu Search Illustration**

Consider an example with a minimum spanning tree (MST)[1] and additional constraints [Glo90] as shown in Figure 7.2. The graph consists of five nodes, hence its spanning trees consist of four edges. The cost of the edges is indicated next to the name of the edges i.e *a*, *b*, etc. The neighborhood solution is defined by edge exchange moves where an added edge to the tree is added to the TL and remains tabu for two iterations and is then removed from TL. Thus TT is equal to 2. The AC criterion selected to override tabu status is the one that allows the current move to include a tabu edge if

[1]Given a connected, undirected graph, a spanning tree of that graph is a subgraph which is a tree and connects all vertices (nodes) together. A minimum spanning tree is a spanning tree with weight less than or equal to the weight of every other spanning tree.

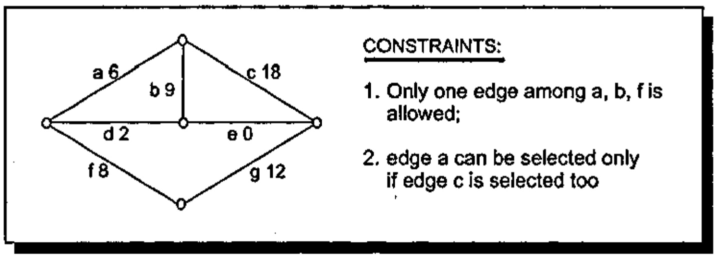

a 6
b 9
c 18
d 2
e 0
f 8
g 12
CONSTRAINTS:
1. Only one edge among a, b, f is allowed;
2. edge a can be selected only if edge c is selected too

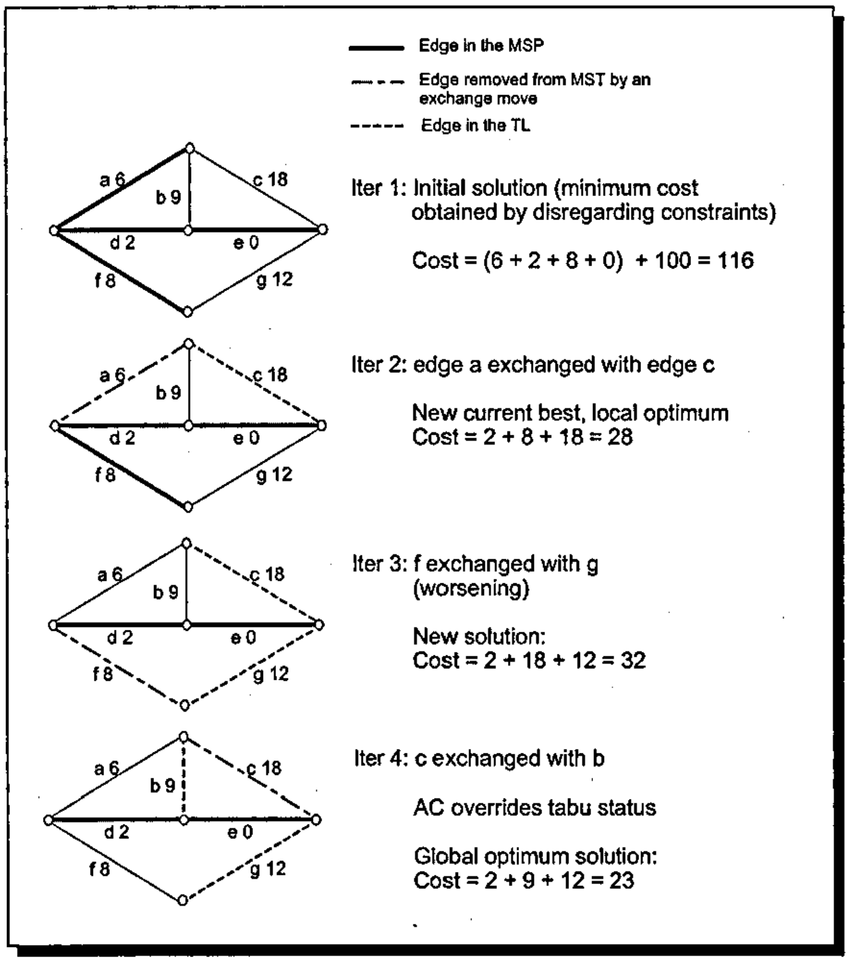
Figure 7.2: TS model: Minimum Spanning Tree

the resulting tree is better (cost improves) than the best tree produced so far. The cost of the tree consists of two elements: the sum of the edge costs for the tree plus a penalty of 50 for a single constraint violation.

Figure 7.3 below illustrates the tabu search operation. At iteration one an initial solution is presented which violates both of the constraints indicated in Figure 7.2 of the system thus the solution is infeasible. At iteration two a best cost change is produced by adding edge *c* and dropping *a*. This eliminates the violation of both constraints reducing the penalty from 100 to 0, while increasing the remaining component cost from 16 to 28. Also edge *c* is added to the tabu list, therefore not allowing it to be dropped for two iterations to follow.

Among the remaining moves the best selected is by adding edge *g* and dropping *f* as shown at iteration three. The selected move is the most attractive as no violation of constraints occur. The cost of the tree worsens indicating that the current tree is a local optimum, since no available moves lead to a better solution. Edge *g* now joins edge *c* in becoming tabu.

At this stage the best of the available moves is to add edge *b* and drop *c*, a move that normally would be disallowed since *c* is tabu. However, the move satisfies the AC by producing a tree with the better cost obtained so far, and consequently the move is selected as indicated at iteration 4. Edge *b* joins edge *g* to constitute the two most recently added edges and hence is designated tabu. The current tree is a new local optimum and also the new current best. Although further iterations are not shown here, the tabu search algorithm will continue functioning in a similar way until a desired iteration cut off will be reached.



Edge in the MSP

Edge removed from MST by an exchange move

Edge in the TL

Iter 1: Initial solution (minimum cost obtained by disregarding constraints)

Cost = (6 + 2 + 8 + 0) + 100 = 116

Iter 2: edge a exchanged with edge c

New current best, local optimum
Cost = 2 + 8 + 18 = 28

Iter 3: f exchanged with g (worsening)

New solution:
Cost = 2 + 18 + 12 = 32

Iter 4: c exchanged with b

AC overrides tabu status

Global optimum solution:
Cost = 2 + 9 + 12 = 23

Figure 7.3: Minimum Spanning Tree: Iterations 1 to 4

7.3 Tabu Search and Short Term Memory

The cell formation problem has been described extensively in previous chapters, and a comprehensive description on forming an initial random solution, when part machine sequence is taken into account, has been presented in Chapter 5. This solution will be fed into the TS algorithm, via which better solutions will be investigated. As already mentioned, TS can either employ short term memory and be characterised as a 'simple' searching process, or long term memory and become more complex. For the current work, short term memory will be utilised and act as the main framework for the TS development. Moreover, a number of technical decisions will be made together with some strategies which will be formulated in order to improve the system's efficiency while the current memory framework is underlined. Before proceeding with the actual development of the TS algorithm, a number of its key elements and tools need to be presented. The specifications for some of these tools won't vary much from their default definition, however certain aspects need to be reconsidered in order to meet the system's requirements.

Please, note that the TS algorithm for CF is implemented using `MatLab`, similarly to the initial starting solution software utilisation. Also the notation employed in Chapter 5 will be preserved here together with some additional which is needed for the TS implementation. This new notation will follow later in this chapter.

7.3.1 Main TS Components

The identification process of the basic elements for TS when short term memory is employed goes in parallel with the actual CF problem as has been already presented (see Chapter 3). A key constraint like part machine operation sequence has taken a primal role for the design of part allocation in machine cells while forming an initial starting solution for the TS. The latter in conjunction with the model's operation and its parameters will also play an important role in the formulation of the short term memory and its components. Some of these main concepts for TS and short term memory components are described in the sections that follow next.

• **Definition of a Feasible Solution**

In the context of the current CF problem, a feasible solution consists of an assignment of machines to cells and an assignment of parts to machine cells when part machine operation sequence is taken into account. As already described in Chapter 5, while machines are allocated to cells certain elements are stored, holding information about the capacity used for each cell and to which cell each machine is allocated. This calculation provides all the information needed for the next step which is the allocation of parts to machine cells. The latter as the most complex and key stage for the formation of the solution involves a significant number of parameters and matrices in order to store information while every part (starting with the part having the majority of processing requirements), following strictly its machine sequence, is allocated to machines in cells. After parts are fully allocated, which is indicated by the fact that a temporary part/machine utilisation matrix becomes empty, the calculation of the relevant costs within the objective function can be carried out. The assignment of parts to cells allows for the calculation of the number of distinct cells used by each part and also the later revisits of parts to already visited cells. Finally, depending upon the number of machines of a specific type used by a part in a particular cell the machine set-up costs can also be determined leading to the total cost calculation for CF.

- **Neighborhood Generation - Moves Definition**

In TS methods, each iteration of the search process focuses on finding a good neighborhood solution with better quality than the current solution. The neighborhood of a current solution is defined as a set of feasible solutions that can be reached by a transition move.

In the TS process, theoretically speaking, the neighborhood of each solution contains all solutions to be explored. However, the size of the neighborhood is determined by several considerations and it can be subject to change because: either the problem is too big for all neighboring solutions to be found, or some of the neighboring solutions are eliminated as their corresponding moves are forbidden (see tabu list definition, page 159). In any case, the size of the neighborhood should allow a thorough search within the neighborhood so that promising regions are investigated.

Similar to the design of the heuristic algorithm in Chapter 6, two types of transitions, either single or interchange, will be used for TS to generate neighboring solutions for each configuration in the system.

Each neighborhood will comprise of solutions produced by moves of machine instance pairs belonging to a certain part. When a different machine instance pair is taken into account another neighborhood with a certain set of solutions will be generated. From each neighborhood a solution will be chosen, based on some criteria as will be explained later, to become the current one from where the search will continue and move onto another neighborhood of solutions generated by another configuration, i.e. another machine instance pair. This process continues until the number of iterations is reached. In order to illustrate the neighborhood generation operation Figure 7.4 is produced where only a small number of generated neighborhoods is shown, i.e. only a few number of iterations is assumed. The dotted line represents the solutions chosen from neighborhood to neighborhood and the actual flow of the search. Once a solution is chosen from current neighborhood another set of neighboring solutions is generated for a different machine instance pair and a solution is chosen. This process continues till the iteration limit is reached. Assume that from each neighborhood the solution indicated by the letter x and the corresponding neighborhood as a subscript is implemented. Moreover, the starting solution is produced by the random initial solution strategy as shown in Chapter 5.

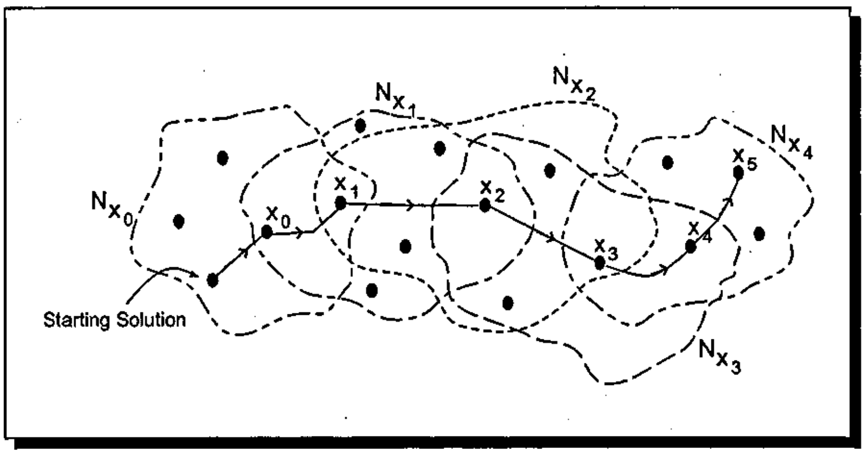


Figure 7.4: TS Neighborhood Generation

As stated already, all solutions generated are produced from either single or interchange moves. Both transitions employed follow the specifications presented in Chapter 6. However, for a complete description of the two transition types and the solutions generated within the tabu search algorithm see the design of the tabu search procedure presented in section 7.4.

- **Short Term Memory: Aspiration Criterion**

Before describing the tabu list operation it is worth mentioning the aspiration criterion (AC) and its value [GL97]. When certain conditions are satisfied, it is desirable to override the tabu status and allow tabu moves to be candidates. The AC employed here resembles the typical and customarily used global form of aspiration by objective, where the tabu status of a move is overridden when the move improves the best value found so far.

- **Short Term Memory: Tabu List & Tabu Tenure**

The short term memory employed is the history of recent moves (recency memory) and aims primarily at preventing moving back to these moves, avoiding being trapped at local optima or causing cycling. The most common implementation of the short term memory is based upon the storage/update of the move attributes that were recently visited.

As already mentioned for each machine instance pair a neighborhood will be generated with solutions produced by considering moving the current machine instance pair either by performing single or interchange transition. At this stage of the search process, where no moves are implemented yet, a temporary tabu list referring to current neighborhood and named *tl_tempx* is created. Elements such as cell assignment, current machine instance pair and tabu tenure referring to each transition involved within current neighborhood are stored in *tl_tempx*. These elements involve both the reverse and forward attributes of the current move. For each newly added entry (row) in *tl_tempx*, the maximum tabu tenure, i.e. T_{MAX} is also recorded. Moreover, each row within the temporary tabu list has another entry added in the first position to denote the neighboring number solution generated within current neighborhood. For example, for the first solution generated within a specific neighborhood the temporary tabu list will have the entity one stored as the first entry for all the rows involved listing both forward and reverse attributes for current solution. This will be used for identification purposes when moves within the current neighborhood will be considered for implementation later on. In order to be able to illustrate the temporary tabu list's operation an example is presented next.

Assume in the first instance that a *single move* is considered for the machine of type i and instance k from a source cell $c1$ to a destination cell $c2$, as shown in Figure 7.5, generating the first neighboring solution within the current neighborhood. The temporary tabu list will be updated and the following entries will be added as shown in matrix (7.1). Please note that *neighbor* 1 is assigned the value one.

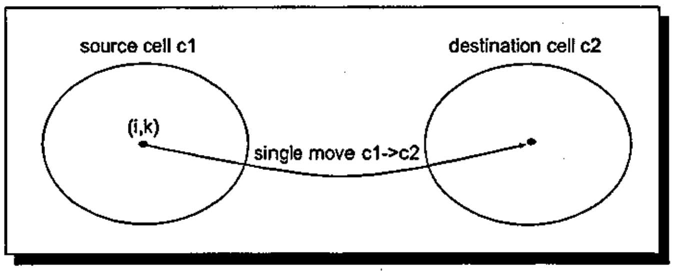


Figure 7.5: Single Transition

$$tl\_tempx = \begin{bmatrix} neighbor\ 1 & c1 & i & k & T_{MAX} \\ neighbor\ 1 & c2 & i & k & T_{MAX} \end{bmatrix} \qquad (7.1)$$

Later on, assume that for the same machine instance pair another neighbor solution will be generated when an interchange move is to be taken into account. More specifically, a machine of type i and instance k is moved from source cell $c1$ to destination cell $c2'$ and another machine of type i' and instance k' is moved from its source cell $c2'$ (i.e. destination cell of first move) to the destination cell $c1$ (source cell of the first move) as shown in Figure 7.6.

After the generation of the second solution within the current neighborhood for configuration (i, k) the tabu list will be updated and will have the form of the matrix (7.2). Please note here also that *neighbor* 2 is assigned the value two.

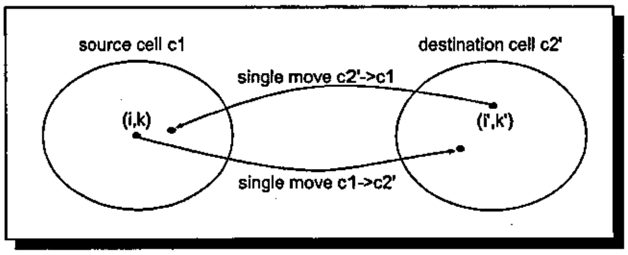


Figure 7.6: Interchange Transition

$$tl\_tempx = \begin{bmatrix} neighbor\,2 & c1 & i & k & T_{MAX} \\ neighbor\,2 & c2' & i & k & T_{MAX} \\ neighbor\,2 & c2' & i' & k' & T_{MAX} \\ neighbor\,2 & c1 & i' & k' & T_{MAX} \end{bmatrix} \qquad (7.2)$$

Please note, that for either single or interchange moves, odd entries in the tabu list represent the reverse of the move and even entries the forward.

In a similar way, the remaining neighboring solutions for the machine of type i, instance k, when source cell is $c1$, will be generated and the temporarily tabu list will be updated accordingly.

Once the entire neighborhood or a subset of it (depending upon the size of the problem) is generated, the moves are starting to be considered for implementation and another tabu list named *tl* (permanent this time) is initialised with three entries in it: cell assignment, machine instance pairs, and tabu tenure. Tabu tenure remains fixed during the entire search, therefore all attributes will remain tabu-active for the same number of iterations as the search goes along. However when a certain move becomes admissible a certain update will take place in *tl* as will be seen later on.

For a theoretical description on the decision of a candidate move to become admissible for implementation when short term memory is employed see Figure 7.7 (page 162).

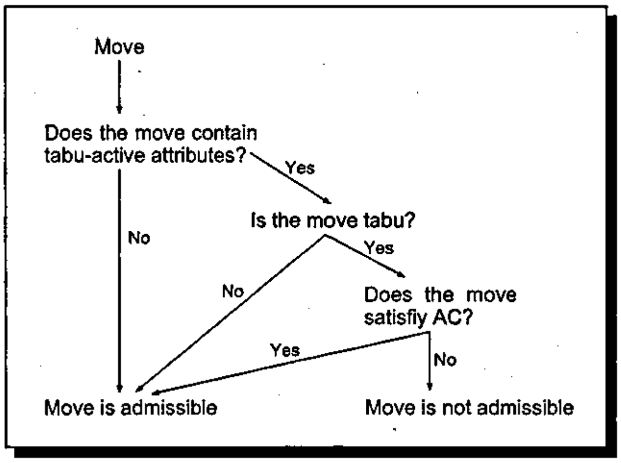


Figure 7.7: Tabu Decision Tree

But how will a certain solution be selected from the neighboring ones for the current study? Initially, all the solutions found within a neighborhood are sorted in ascending order of their corresponding objective values. If the solution with the smallest objective value happens to be less than the best value found so far then its corresponding move will become admissible for implementation no matter what the tabu status is of its related attributes because of the aspiration criterion. If this is not the case, then the procedure will continue locally, within current neighborhood, in search of the move to implement. The solution with the least worst objective value will be examined first to check whether the tabu status of the corresponding move is still active. If the latter is happening then the search will continue examining the solution that follows next in the sorted sequence and so on till a move that can become admissible is found.

It is worth presenting in more details the operation of the tabu list and see the interaction between *tl_tempx* and *tl*. Steps 1 to 3, below, describe the update on the tabu list *tl* when a move becomes admissible since its corresponding value satisfies the aspiration criterion, i.e. objective value for current solution is better than the best found so far.

1. Find in *tl_tempx* the segment referring to the current admissible move, i.e. all move attributes (forward and reverse), and also the segments referring to the

remaining non-implemented moves;

2. Check *tl*:

   (a) if it is empty add for the admissible move both reverse and forward attributes as found in *tl_tempx*;

   (b) else, add in *tl* both reverse and forward attributes of the admissible move found in *tl_tempx* as above, and also check the forward attributes of the remaining moves if they exist in *tl* from previous transitions. If they do, i.e. they are still tabu active, leave the entries as they are with their current tabu tenure; Otherwise place them in *tl* with tabu tenure equal to T_{MAX}. The latter is happening so that re-consideration of these will be avoided for a substantial time interval as these moves are of no interest for the near future;

3. For completion of processes 2 (a) and 2 (b), subtract one unit from tabu tenures of the other entries in tabu list and remove those entries whose tabu tenure is zero.

If the AC is not satisfied, then the search will take place locally, as already stated, considering to implement the best solution in the neighborhood caused by a move whose forward attribute [2] is not tabu. If the tabu list is empty then the best solution within the neighborhood can become admissible for implementation. In this case both attributes, reverse and forward of the current move, are added into the tabu list *tl* with tabu tenure equal to T_{MAX} similar to the process described above. Also for the remaining segments of the non-admissible moves their forward moves are also added to the tabu list *tl* and their tabu tenure is set to T_{MAX} unless they are already in *tl* so no update takes place. Also step 3 above is implemented last as well.

In the case where the tabu list is not empty then the search continues exploring all solutions in the neighborhood (starting from the least worst) whose forward attribute is non-tabu and can become admissible. Once this solution is found the tabu list is updated as shown above in steps 2 (a) and 2 (b) and 3.

In case that all solutions in the neighborhood still have tabu active forward attributes the search process will continue with any of the following three solutions depending upon the nature of the data set employed:

[2]Please note that when an interchange move is encountered only the forward attribute of the first single move, i.e. from source to destination cell, is examined.

1. current least worst found locally;
2. current worst found locally;
3. best solution found so far.

- **Stopping Criterion**

The tabu search procedure stops when a predetermined number of iterations has been reached; or the solution has not been improved after a certain number of consecutive iterations [Glo90]. In the current study the iterative tabu search process is terminated after a certain number of iterations is reached as will be seen in section 7.4.3. But before, the actual TS algorithm is presented next.

7.4 Tabu Search Algorithm Design

The actual design of the tabu search forms the key stage for the investigation of better solutions for CF when larger data sets are to be taken into account. In order to build this algorithm all of its components, as described in the previous section, are assembled together within an iterative procedure aiming to search the neighboring space. Moving transitions, either single or interchanges, are included within the iterative procedure, and some additional heuristic approaches are proposed within the latter to produce a better and a more efficient algorithm.

Similarly to the heuristic algorithm presented in Chapter 6 the tabu search proposed here will be split into different phases for better illustration. These phases are as follows:

- Initialisation Process;
- Iterative Procedure;
- Termination.

7.4.1 Initialisation Process

In order to present the iterative procedure, certain elements need to be initialised with most of them having been specified in previous chapters. More specifically, and for the need of the tabu search algorithm the initialisation presented in Routine 1 will be employed.

Routine 1: Initialisation phase for TS

- initialise $BEST\_COST$ to be equal to $INIT\_OBJVAL$;
- initialise $BEST\_SOL$ to be equal to $INIT\_SOL$;
- initially assume that $BEST\_CELLMATRIX2$ is equal to $CELLMATRIX2\_temp$;
- initially assume that $BEST\_part\_moves$ is equal to $part\_moves\_temp$;
- initially assume that $BEST\_W\_jq$ is equal to W_jq_temp;
- initialise $CELLMATRIX2\_temp$ to be equal to $CELLMATRIX2$;
- initialise $part\_moves\_temp$ to be equal to $part\_moves$;
- initialise W_jq_temp to be equal to W_jq.

It is also worth commenting on the creation of an additional 4D matrix, named $CELLMATRIX2\_4D$. This matrix will be used in order to be able to store for current generated neighborhood its associated solutions and more specifically the allocation of machines to cells. The size of the matrix is $(NM \times KMAX \times NC \times length(all\_cells))$. The first three dimensions match the size of $CELLMATRIX$ initially formed in Chapter 5, whereas the last refers to the number of the solution, within current neighborhood, that is currently explored.

7.4.2 Iterative Procedure

The iterative procedure is the main phase of the TS algorithm where investigation of better solutions close to optimum are sought. It mainly consists of two different stages. The first stage involves the consideration of a machine instance pair belonging to a certain part's machine operation sequence and its transition to other cells where either a single or an interchange transition takes place. The latter, as the heuristic approach presented in Chapter 6, depends upon the capacity of the associated source and candidate destination cells. Each transition entails the creation of a unique solution within the neighborhood. Then a part reallocation is performed and finally the evaluation of the objective value takes place. Moreover, as described previously a temporary tabu list is updated within this stage.

The second stage within the iterative procedure comprises the main stage within the search process since the principles of the tabu search algorithm are adopted and decisions are made concerning the direction of the search. More specifically, once the neighborhood has been generated from the first stage, a process commences searching which move within the current neighborhood to actually implement.

Similarly to the heuristic algorithm the tabu search algorithm begins by considering the parts in the system and more specifically the part that causes the majority of the intercellular movements in the initial solution. Given the distinct allocation of parts to cells (W_jq), as produced from the initial starting solution (see Chapter 5), parts are ordered in descending order relative to the number of intercellular movements. The reasons for doing this were presented in section 6.3.2.

Moreover, with reference to the current part in operation, the segment corresponding to its cell machine instance pair allocation relative to its machine sequence is found in *part_moves_temp* and stored in vector *qik*. Each machine instance pair in the latter, is a candidate for moving it from the cell currently in use, i.e. source cell, to the remaining cells, i.e. destination cells, investigating for current configuration the associated neighborhood.

The pseudo code presented next, in Routines 2-4, includes the main steps and relevant loops involved within the iterative tabu search approach. For more information on the actual code involved see Appendix E. Please note that Routine 3 is briefly described as the transitions types either single or interchange have already been presented within section 6.3.2.

Routine 2: TS Iterative Framework

```
• Initialise a tabu list, i.e. $tl$, to be empty;
• Initialise $i\_global$ to zero;
• while $i\_global$ less than $\theta$ ($\theta$ defined in section 7.4.3 - Termination)
    1. Order the parts in descending order of the intercellular movements caused; Return the sorted parts in $wi$ and number of intercellular movements for each sorted part in $wy$;
    2. Initialise a counter, i.e $counter\_part$, for choosing part;
    3. Initialise $i\_part$ to zero;
    4. while $i\_part$ less than ($\varphi \times length(wi)$) ($\varphi$ defined in section 7.4.3 - Termination)
        – if $counter\_part$ is greater than or equal to $length(wi)$ then $counter\_part$ becomes equal to zero; end if
```

- Current part index, i.e. $j2$, becomes equal to $counter\_part + 1$;
- Initialise i_source to be equal to zero
- Initialise a counter, i.e. $counter\_source$, for choosing source cell index;
- Initialise a counter, i.e. $counter\_objval$, for counting the iterations that the obj. value does not improve;
    * while i_source less than $(\chi \times length(wi))$ (χ defined in section 7.4.3 - Termination)

    (a) if for α[3] iterations the objective value does not improve, i.e. $counter\_objval \geq \alpha$ then break the current loop and continue with another part;
    $CELLMATRIX2\_temp$ equals $BEST\_CELLMATRIX2$;
    Update part_moves_temp and W_jq_temp;
    Continue with next part in the sequence;
    end if

    (b) For actual part, i.e. $wi(j2)$, find the corresponding segment in $part\_moves\_temp$ and store it in qik;

    (c) if $counter\_source$ is greater than or equal to $length(qik)$ then counter becomes equal to zero; end if

    (d) The index of the row of qik, i.e. $c1$, is now is equal to $counter\_source+1$;

    (e) The actual source cell is: $source\_cell = qik(c1, 1)$;

    (f) Assign all cells in a vector named all_cells;

    (g) Take the first row in qik, find the cell where the first machine instance pair is allocated and store it in a vector named cc;

    (h) Delete this from all_cells;

    (i) Initialise a 4D matrix named $CELLMATRIX2\_4D$;

    (j) Initialise a vector, i.e. $OBJVALM$, for storing the objective values of all neighbors in current neighborhood of length equal to one up to the length of all_cells;

    (k) Initialise the temporary tabu list, i.e. tl_tempx equal to empty;

    (l) Continue with Routine 3: Neighborhood Generation;

    (m) Continue with Routine 4: Selection of Admissible Move from the generated neighborhood in Routine 3;

    (n) $counter\_source$ becomes equal to $c1$;

[3] α is a parameter whose value varies depending upon the problem used each time.

```
            (o) Accumulate $i\_source$;
              end while $i\_source$
          – $counter\_part$ becomes equal to $j2$;
          – Accumulate $i\_part$;
        end while $i\_part$
    5. Accumulate $i\_global$;
  end while $i\_global$
```

Routine 3: Neighborhood Generation

```
1. for $i\_dest$ equal to one up to the length of $all\_cells$
    • Store the cell corresponding to $i\_dest$ index in $dest\_cell$;
    • if capacity of machines stored in $dest\_cell$ (destination cell) cell is less than
      $E_{MAX}$ & capacity of $qik(c1,1)$ (source cell) is less than or equal to $E_{MAX}$ &
      greater than $E_{MIN}$
        – Single move commences as presented in section 6.3.2;
        – Reallocate parts to machine cells for current transition;
        – Evaluate current Objective Value named $OBJVAL\_tempx$;
        – Store objective value corresponding to current $i\_dest$ in $OBJVALM$;
        – Update $tl\_tempx$;
      else
        – Interchange move commences as presented in section 6.3.2;
        – Reallocate parts to machine cells for current transition;
        – Evaluate current Objective Value named $OBJVAL\_tempx$;
        – Store objective value corresponding to current $i\_dest$ in $OBJVALM$;
        – Update $tl\_tempx$;
      end if
```

Routine 4: Selection of Admissible Move

```
1. Sort the objective values, found in $OBJVALM$ in ascending order. Store the latter
   in $OBJVALM\_sorted$ and their corresponding indices in $OBJVALM\_sortedi$;
2. Sort also the $CELLMATRIX2\_4D$ using the $OBJVALM\_sortedi$ indices and
   return a 4D matrix named $CELLMATRIX2\_4D\_sorted$;
```

3. Find the segments in tl_tempx corresponding to each generated $OBJVALM\_sortedi$ and sort them accordingly;

4. if local best, i.e. $OBJVAL\_sorted(1)$, less than $BEST\_COST$;

   - Update $CELLMATRIX2\_temp$ with the corresponding first solution stored in $CELLMATRIX2\_AD\_sorted$
   - Perform part reallocation to receive updated $part\_moves\_temp$ & W_jq_temp when machine cell allocation is equal to $CELLMATRIX2\_temp$;
   - Update global best value, i.e. $BEST\_COST$ with local best;
   - Update $BEST\_CELLMATRIX2$ with $CELLMATRIX2\_temp$;
   - Update $BEST\_part\_moves$ with current, i.e. $part\_moves\_temp$;
   - Update $BEST\_W\_jq$ with current W_jq_temp;
   - Update tabu list (see section 7.3.1);

   else if local best neighboring solution worse than global best

   - Accumulate *counter_objval*;
   - if tabu list tl is not empty
     - Find those solutions with reference to their corresponding sorted objective values whose forward moves are in the tabu list and store them in a vector named *collect_notimpl*;
     - Identify those not in tl; Store them in a vector named *imp_candidatei*;

     else the candidate to implement is the first with the least worse objective value; end if
   - if *imp_candidate* is not empty, i.e. there is at least one candidate solution to become admissible
     - Choose the smallest of all (i.e. with the min. objective value) and update $CELLMATRIX2\_temp$;
     - Perform reallocation of parts to receive updated $part\_moves\_temp$ and W_jq_temp when machine cell allocation is equal to $CELLMATRIX2\_temp$;
     - Update tabu list tl (see section 7.3.1);

     else if *imp_candidate* is empty, i.e. all related attributes for all moves are tabu active perform one of the following:

```
        – Update $CELLMATRIX2\_temp$ with
          $CELLMATRIX2\_4D(:,:,:,length(OBJVAL\_sortedi))$ (current worst);
        – Update $CELLMATRIX2\_temp$ with
          $CELLMATRIX2\_4D\_sorted(:,:,:,1))$ (current least worst);
        – Update $CELLMATRIX2\_temp$ with $BEST\_CELLMATRIX2$ (current best);
        – For any of the above three cases chosen perform part reallocation with the
          corresponding machine cell allocation to receive the updated $part\_moves\_temp$
          and $W\_jq\_temp$;
    end if
end if
```

It is worth commenting on a few things in the above routines. In a similar way with the design of the heuristic algorithm, in Chapter 6, when a machine instance is moved from a source cell to a different one both current part's machine cell allocation and other part's situation might get affected because of the existence of the part machine operation sequence. For this reason any time a move is considered and before evaluating the value of the objective function part reallocation is performed, as shown in Routine 3, which outputs the updated *part_moves_temp* and also increases the chances of receiving a better output value. Within routine 4 however, when a move becomes admissible part reallocation is only performed for reasons of receiving the updated *part_moves_temp* and *W_jq_temp* both to be used in the iteration that follows. Please note that the latter does not change the CF configuration since for each of the cases examined the updated machine cell allocation, after a certain move was considered, is revisited and acts as the basic input to the part re-allocation process.

7.4.3 Termination

The tabu search algorithm will terminate when a pre-specified total number of iterations is reached. As already seen in the routine 2, three loops are considered whose duration depends upon the values of parameters θ, φ and χ and the length of wy which is equal to the total number of parts employed each time. The outermost loop concentrates on the total number of parts and their order to be investigated. Within the nested loop, a part is chosen and within the innermost loop transitions of the machine instance pairs relative to current part's machine operation sequence between source and destination cells are taking place.

An attempt was made in order to illustrate with a formulae the total number of iterations involved as shown in equation 7.3.

$$Total\ \ TS\ \ Iterations = \theta \times \{(\varphi \times length(wy)) \times [(\chi \times length(wy)) \times length(NC-1)]\} \quad (7.3)$$

7.5 Summary

For investigation of a higher searching strategy possibly more promising than the heuristic approach described in Chapter 6 an algorithm based on a metaheuristic approach such as TS was of need. This chapter presented a TS algorithm for the CF problem for investigating a significant number of solutions given an initial heuristically based random solution. The type of memory employed was short term memory whose components such as tabu list, tabu tenure, aspiration and stoping criteria were designed to reflect on the CF's requirements and specifications. Within the iterative procedure two types of transitions, single and interchange, were considered for the creation of feasible solutions. Both transitions were designed to be independent from each other, however, both working towards the exploration of better solutions. For examining and criticising the tabu search algorithm's effectiveness, testing will be carried out in Chapter 8 with an additional comparison between the latter and the heuristic approach for better clarification on the performance and the computational effectiveness of both.

Chapter 8

Tabu Search Computational Results

8.1 Introduction

This Chapter presents a number of computational results for the metaheuristic approach, proposed in Chapter 7, where small, medium and large scale problem sizes, the same with those generated in Chapter 6, are employed.

Similarly to the heuristic approach and in order to have a clear picture on the tabu search algorithm and its operation a thorough description will be provided examining its iterative behaviour, the efficiency of the initial solution within the latter and the quality of the solutions obtained together with the computational times required.

Finally, a comparison between the heuristic approach and the tabu search algorithm will be presented for completeness.

Please note that similar to the heuristic approach, the tabu search algorithm was also coded in `MatLab` and run on a personal computer (Genuine Intel(R) 1.66 GHz, 1.00 GB of RAM).

8.2 Computational Results for the TS Algorithm

The medium and large data sets will be examined first concentrating on the computational results and then analysing a number of elements such as: the behaviour of the iterative procedure, the tabu search initial starting solution and the tabu list size involved.

8.2.1 Computational Results for Medium & Large Sized Problems

The computational results for medium and large data sets are presented in Table 8.1 (page 174). In a similar way to the presentation of the heuristic results, in Chapter 6, the elements determined here for each problem instance when the tabu search iterative procedure is employed are: number of cells in the system, total number of machine instances, the best known objective value obtained via the mathematical solver and the corresponding CPU time. Moreover, the best value of the objective function obtained via TS and the required CPU time together with the value of the initial objective value are also included. Additionally, the deviation of TS from the optimum or best known solution obtained via XPRESS-MP is included together with the percentage of deviation of the best TS solution from the initial solution which was randomly generated. Finally, the mean values of both deviation percentages together with the mean value for the processing times are presented at the bottom of the table. Also, the number of parts and machine types involved in each problem are also listed within Table 8.1 for completeness.

The results presented in Table 8.1 are very good compared to integer programming methods proving that the tabu search algorithm is very effective in obtaining rapidly high quality solutions. More specifically, the mean deviation value of TS algorithm from the best known objective value is ten percent. Moreover, the required CPU time for most of the problem instances does not exceed the two thousand seconds overall (problem twelve is currently excluded, as the largest data set of all needs a substantial CPU time in order for the iterative procedure to end).

Table 8.1: Problem Data and Computational Results for Medium & Large Problem Sizes

						XPRESS-MP		Tabu Search (TS)				
Prob.	NM	NP	NC	TMI	E_{MAX}	Obj.	CPU time	Init. Obj.	Best Obj.	Dev.†††	CPU time** (secs)	Dev.††
1.	10	19	5	40	9	444.13†	> 30 hrs	623.73	481.74	23%	102.52	8%
2.	7	10	5	20	6	219.92*	141 secs	339.96	242.83	29%	18.953	9%
3.	14	24	5	37	8	383.78†	> 30 hrs	562.36	428.4	24%	133.97	12%
4.	30	50	8	99	13	1100.02†	> 30 hrs	1642.3	1178.8	28%	1945.5	7%
5.	15	30	7	47	7	582.71†	> 30 hrs	928.48	641.92	31%	333.3	10%
6.	30	41	7	94	14	893.28†	> 30 hrs	1643.6	968.3	41%	1206.2	8%
7.	23	20	6	42	8	390.51†	> 30 hrs	667.14	410.51	38%	275.11	5%
8.	8	20	5	31	7	333.82†	> 30 hrs	541.48	385.99	29%	75.938	16%
9.	10	15	5	30	7	302.25†	> 30 hrs	501.87	332.3	34%	58.563	10%
10.	9	9	5	29	6	242.13†	> 30 hrs	387.07	271.05	30%	30.422	12%
11.	9	9	5	28	6	269.11†	> 30 hrs	407.35	298.65	27%	33.969	11%
12.	36	90	10	147	17	1441.62†	> 30 hrs	2872.9	1755.2	39%	21405	22%
13.	14	24	5	45	10	508.82†	> 30 hrs	872.71	557.22	36%	168.83	10%
14.	20	35	7	64	11	732.37†	> 30 hrs	1068.6	743.19	30%	507.52	1%
15.	16	43	9	65	11	799.08†	> 30 hrs	1283.6	860.51	33%	825.67	8%
16.	12	19	5	29	7	293.85*	196 secs	406.6	318.01	22%	55.422	8%
17.	16	43	8	61	10	713.97†	> 30 hrs	1051.3	810.97	23%	742.47	14%
18.	27	27	8	76	10	763.51†	> 30 hrs	1223.5	816.07	33%	720.03	7%
19.	26	37	8	64	10	644.41†	> 30 hrs	1131.3	671.61	41%	670.11	4%

Mean Rounded Values		
Dev.	CPU	Dev.
31%	1543 secs	10%

* Optimum solution.

** CPU time needed when only one run of the algorithm was attempted. Problem 12 is currently excluded see remark 4 Section 8.3.

† Best known solution.

†† Percentage deviation of each solution from the optimum or best known solution, obtained via XPRESS-MP, and produced as follows:

$$Dev. = round\left(\left(\frac{TS\ Best\ Obj. - XPRESS{-}MP\ Best\ Obj.}{XPRESS{-}MP\ Best\ Obj.}\right) \times 100\right)$$

††† TS Percentage of improvement: Init. Obj. vs. Best Obj. as follows:

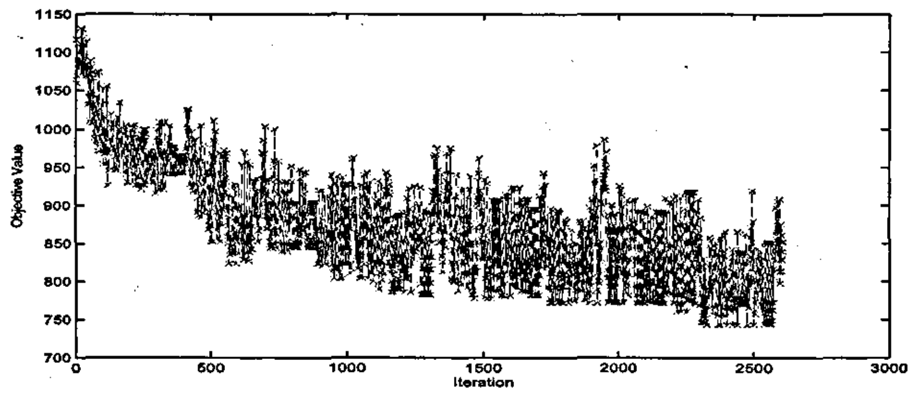
$$Dev. = round\left(\left(\frac{Init.\ Obj. - Best\ Obj.}{Init.\ Obj.}\right) \times 100\right)$$

- **Tabu Search Iterative Procedure Behaviour**

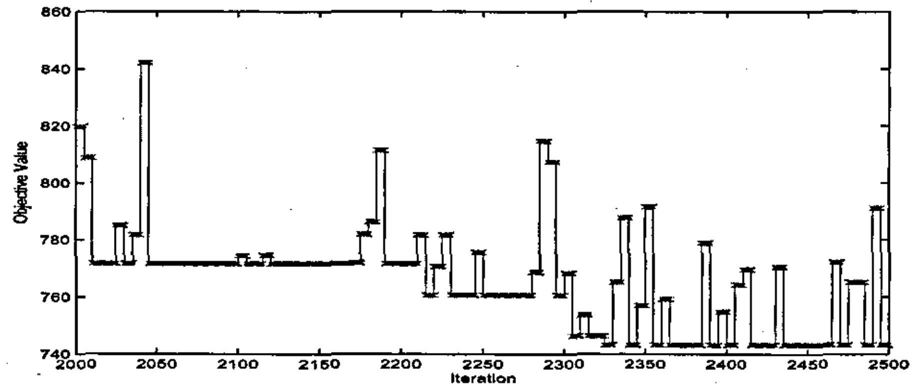
As stated in Chapter 7 and before the commencement of the tabu search, the value of the objective value becomes equal to the value of the initial solution that is fed into the system. Within the iterative procedure and for each machine instance pair belonging to the current part's machine instance cell situation, which is relative to the part machine operation sequence, its neighborhood is generated and a number of solutions with their corresponding objective values are obtained. Later on, a decision is made in order to decide which move to implement from all the possible included within the neighborhood. From the set of solutions the least worse solution, that is not tabu active, becomes admissible for implementation unless the AC criterion is met. The procedure continues with the next machine instance pair in sequence for the current part till it reaches the iteration limit (i.e. the value of χ) where the next part in sequence is taken into account.

In order to illustrate the behaviour of the TS iterative procedure problem instance fourteen is employed where the deviation of the TS from the best known value is one percent. Figure 8.1(a) shows the fluctuation of the objective values with respect to the iterations involved. Please note, that in the latter all objective values including those produced from the non-admissible moves in each generated neighborhood are displayed in order to have a clear picture of the TS and how it evolves within the search space. Also Figure 8.1(b) presents a portion of the trend on the objective values of Figure 8.1(a) only for the implemented moves between iterations 2000 and 2500. It can be seen from Figure 8.1(a) that the total number of iterations needed is approximately 2600. Moreover, the best objective value of magnitude 743.19 units, as specified on Table 8.1, is first obtained at iteration number 2326 and then appears again at later iterations.

Although a rapid descent flow of the objective values is shown in Figure 8.1(a) a significant number of solutions is investigated in the search space increasing the possibilities of receiving a better TS deviation from the best known objective value. Moreover, since the search does not always continue from the best solution found, unless the AC is met, leads to the investigation of areas that might not sound very promising but could lead later on to very useful results when certain moves are implemented. Moreover, the deviation between the initial objective value and the best cost obtained has magnitude 30% which is very significant, thus the iterative tabu search algorithm operates effectively.



(a) Objective Function Value Evolution: All Moves



(b) Implemented Moves: Portion of Figure 8.1(a) between Iterations 2000-2500

Figure 8.1: Problem instance 14

• Tabu Search vs. Initial Solution

As already discussed in Chapter 7, the initial solution is randomly generated and fed into TS before the iterative procedure begins. From Table 8.1 can be seen that for problem instance fourteen the value of the initial objective value is 1068.6, which is produced with a certain allocation of machines to cells, whereas the value of the best cost obtained via the TS, which is based on the initial solution, is 743.19 units. If a different initial allocation of machines to cells is fed into the system a different output will be produced, however its value won't differ much from the first obtained. The reason for the latter is the neighborhood generation for each candidate machine instance pair where an investigation of many areas/solutions is taking place increasing the possibilities of receiving good and stable results.

In order to illustrate the above, Figure 8.2 (page 178) is presented where a different initial solution is generated for problem instance fourteen and the iterative procedure is run only once. Please note that both the total number of iterations and the value of tabu tenure remain the same.

From Figure 8.2 it can be seen that the total number of iterations is approximately 2600, same as before. The deviation of the TS from the best known solutions is 3%, when the value of the initial solution is 1134.1 and the best cost found is 752.37 units.

• Tabu Search: Tabu Tenure Size

There are no general guidelines to determine the optimal sizes of tabu tenures. In practice the search process should be constructed with different values of tabu tenures. From the computation of the large scale problems, it is observed that for sizes suggested by Glover and Laguna [GL97] which are five to ten, the evaluated solutions did work well in solving the current problems. As stated in Chapter 7 the length of tabu tenure which is utilised for each element in the tabu list is to be of fixed size for each problem instance. The values used most for all problem instances range between five and fifteen.

For the objective value evolution of problem instance fourteen as shown in Figures 8.1(a) and 8.2 the size of the tabu tenure employed was equal to ten. What will happen if the value of the latter changes? Assume problem instance fourteen with tabu tenure equal to fifteen, same initial solution, which was used to produce Figure 8.2, and also the same total number of iterations. The result is shown in Figure 8.3 (page 179).

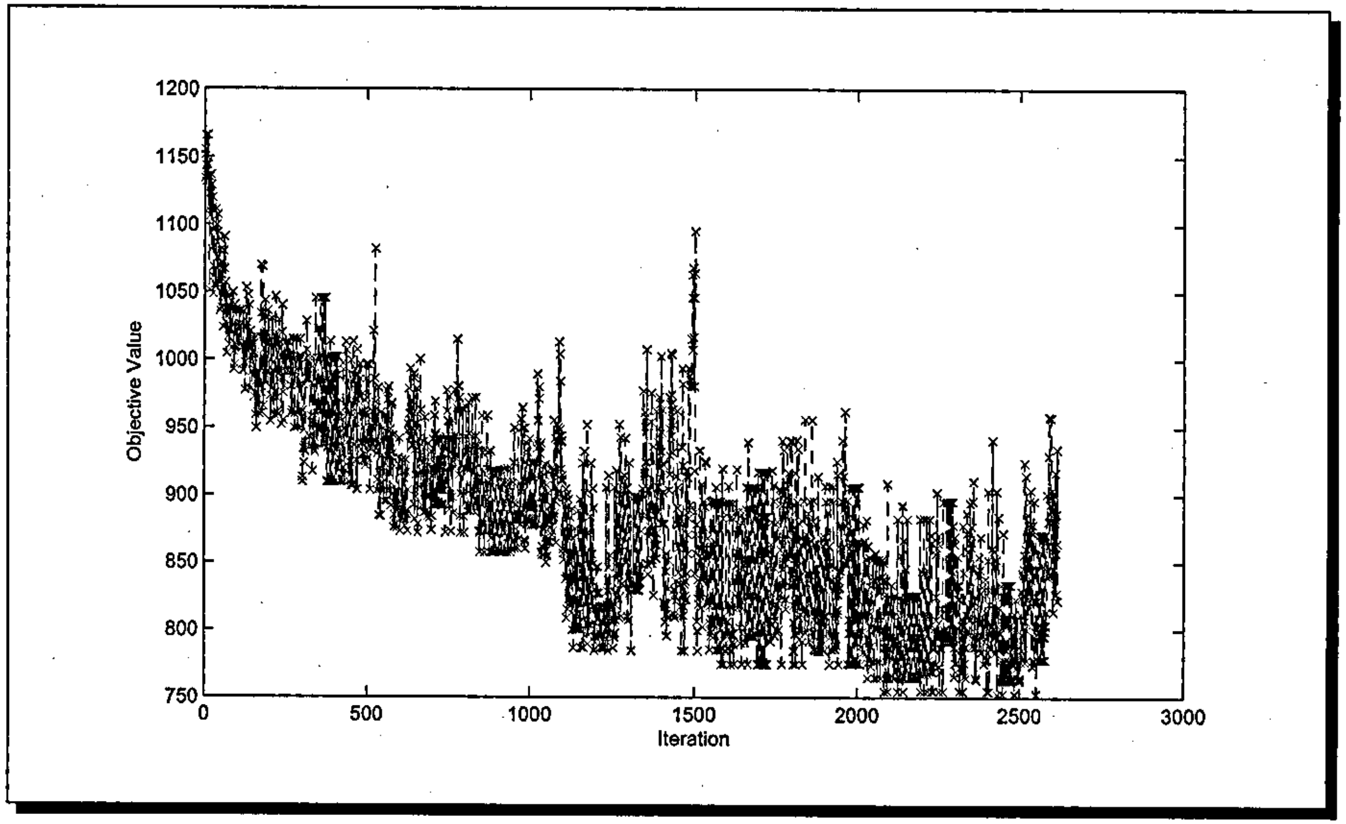


Figure 8.2: Problem instance 14: Objective Function Value Evolution when Initial Solutions Differs

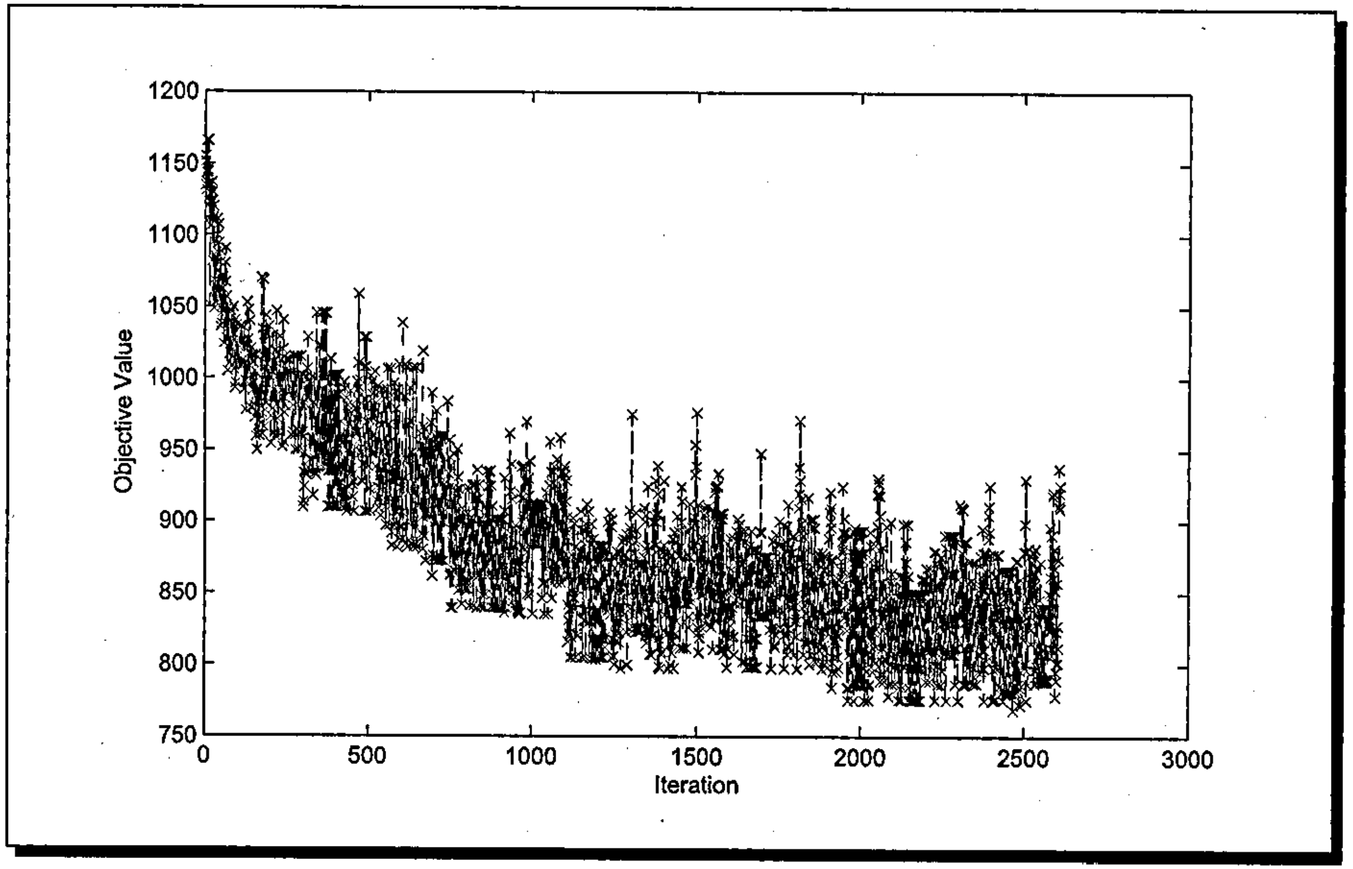


Figure 8.3: Problem instance 14: Objective Function Value Evolution when TT=15

The output generated in Figure 8.3 is different from the one shown in Figure 8.2 as the estimated deviation is five percent and the best cost encountered 768.26 units. Thus a different value of tabu tenure produces different evolution on the objective value with respect to the iterations and different TS deviation from the best known value. After a number of experiments it was decided that for medium to large problems the value of tabu tenure should be ten.

- **Additional Objective Value Fluctuations**

For the purpose of this section problem instance twelve will be employed as it happens to be the largest of all with 147 total number of machines and 90 parts. The deviation produced via TS from the best known value is of magnitude twenty two percent with tabu tenure equal to twelve and total number of iterations increased significantly from the rest of the problem instances. The fluctuation on the objective value can be seen in Figure 8.4 (page 181).

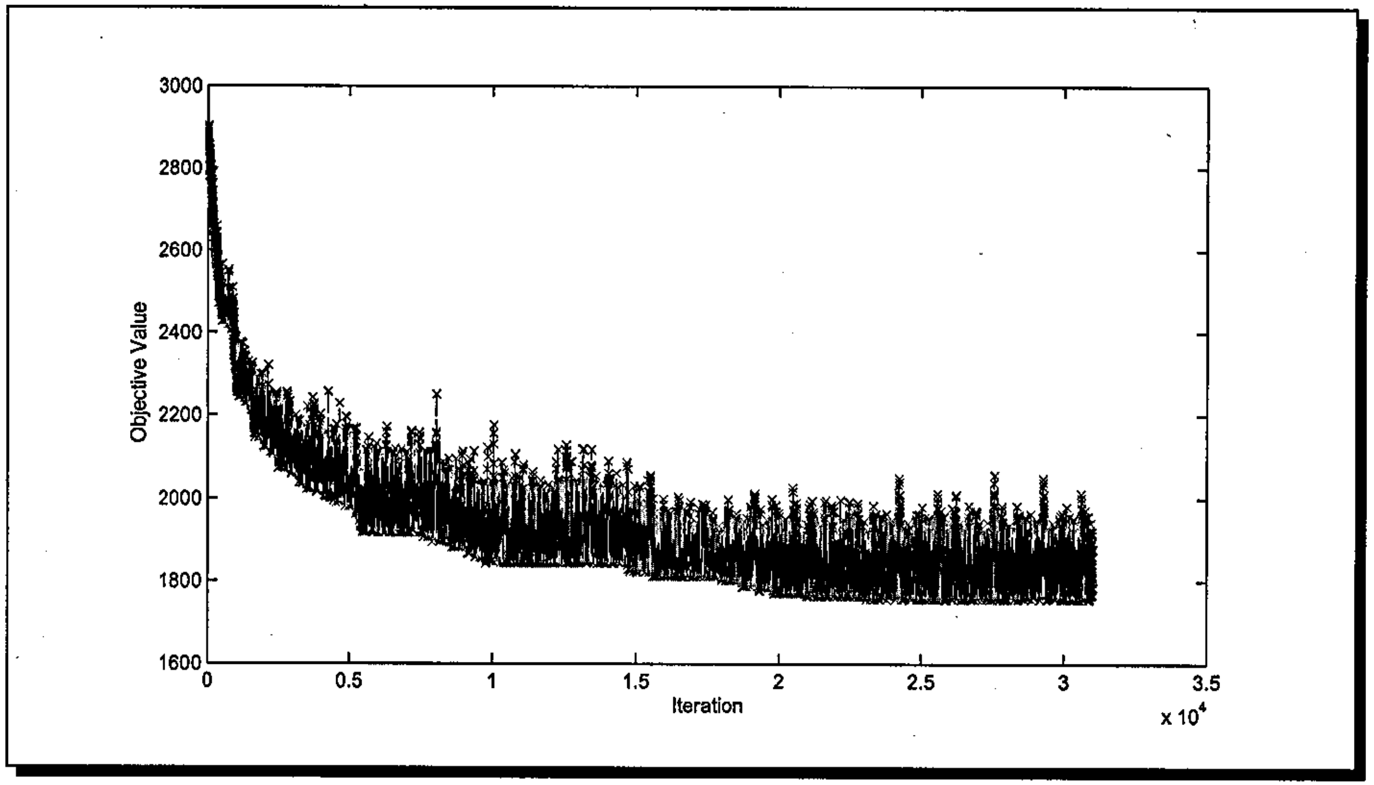


Figure 8.4: Problem instance 12: Objective Function Value Evolution

8.2.2 Computational Results for Small Sized Problems

In a similar way with the heuristic approach that was tested with small sized problem instances, the tabu search algorithm is tested here with the same problem instances and the results are shown in Table 8.2 (page 183). It is worth noting that the mean deviation value of the tabu search from the best known solution is six percent, whereas the mean deviation of the best cost found from the initial value is twenty seven percent. Moreover, the mean processing time is only ten seconds. Please note also that similarly with the heuristic approach, the optimum value is found for two of the problem instances when tabu search was employed. Moreover, all results were obtained with only one trial, i.e the algorithm was run only once.

In order to illustrate the operation of the tabu search algorithm, problem instances twenty two and twenty three will be employed and the evolution on their objective values will be examined. Figure 8.5 (page 185) shows the trend on the objective value for both problems when total number of iterations needed is 366 and 357 respectively.

Figure 8.5(a) shows that the best value of the objective value for problem twenty two is found only once at iteration number 344, whereas from Figure 8.5(b) the best value for problem twenty three is first found at iteration number 170 and to later iterations as well.

For both problems, twenty two and twenty three, the value of the tabu tenure used is equal to five. If the value of tabu tenure changes the evolution of the objective value should differ. In order to illustrate the latter, assume that for both problems the value of tabu tenure is set up equal to ten. Please note that the initial solutions initially fed into the system are used here as well. The new evolution on the objective value for both problems, can be seen in Figure 8.6 (page 186). The deviation of TS from the best known solution is recorded to be fourteen percent for problem twenty two and nine percent for problem twenty three. Both values are much greater than the deviation values initially found as shown in Table 8.2. In conclusion, for small problems, tabu tenure values close to five seem to be more suitable. The latter is true, since within a small problem the candidate moves are only a few and the more restricted these are the more difficult it is for the algorithm to converge to a value close to optimum when a simulation is run.

Table 8.2: Problem Data and Computational Results for Small Problems

						XPRESS-MP		Tabu Search (TS)				
Problem	NM	NP	NC	TMI	E_{MAX}	Opt.*	CPU time	Init. Obj.	Best Obj.	Dev.††	CPU time** (secs)	Dev.†
20.	6	8	5	22	6	213.70	23.80 hrs	328	231.69	29%	15.766	8%
21.	6	8	4	19	6	143.04	71 secs	235.87	154.35	35%	7.875	8%
22.	6	8	5	23	6	204.25	3210 secs	294.06	211.63	28%	15.422	4%
23.	6	8	5	28	6	229.09	4167 secs	334.81	237.53	29%	17.297	4%
24.	5	7	3	17	6	153.36	52 secs	181.37	161.23	11%	16.813	5%
25.	5	7	4	20	6	183.14	2871 secs	265.38	200.59	24%	11.172	9%
26.	5	7	3	17	6	153.19	25 secs	212.11	164.7	22%	6.6875	8%
27.	4	6	3	13	5	117.82	1 sec	195.47	132.69	32%	4.7344	13%
28.	4	6	3	10	4	113.26	16 secs	156.73	113.26	28%	4.0313	0%
29.	4	6	3	9	4	103.08	1 sec	142.06	103.08	27%	3.625	0%

Mean Rounded Values		
Dev.	CPU	Dev.
27%	10 secs	6%

* Optimum solution obtained via XPRESS-MP.

** CPU time needed when only one run of the algorithm was attempted.

† $Dev. = round((\frac{Best\ Obj. - Opt.}{Opt.}) \times 100)$

†† TS Percentage of improvement: Init. Obj. vs. Best Obj. as follows:
$Dev. = round((\frac{Init.\ Obj. - Best\ Obj.}{Init.\ Obj.}) \times 100)$

8.3 Tabu Search Algorithm vs. Heuristic Approach

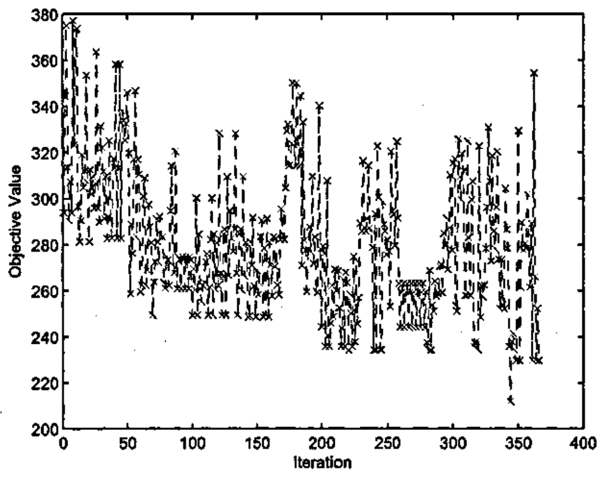
As already presented in Chapter 6, although the proposed heuristic algorithm is effective as the results recorded for a variety of problem sizes are very good, it has certain limitations in its operation as described in section 6.6. On the other hand the tabu search algorithm, proposed in Chapter 7, is currently proved to be better in most cases and more effective both in its functionality and in the quality of the solutions produced. At this point a number of concluding remarks can be drawn summarising the basic operations for the developed tabu search algorithm via which its advantages against the heuristic approach will be identified. These are described below as follows:

1. The results of testing when medium and large scale problems employed showed that the tabu search procedure is very effective and the quality of the solutions generated adequate. The mean deviation value of the tabu search best cost from the best known solution obtained with XPRESS-MP was found to be of magnitude equal to ten which was better than the mean value produced via the heuristic approach which was twelve. Comparing the results for all problem instances for both the TS and the heuristic approach it can be observed that for the majority of them smaller deviations were obtained when TS was employed. Only for problem instance nineteen the heuristic approach performed slightly better than the TS algorithm since the deviation produced was of smaller magnitude.

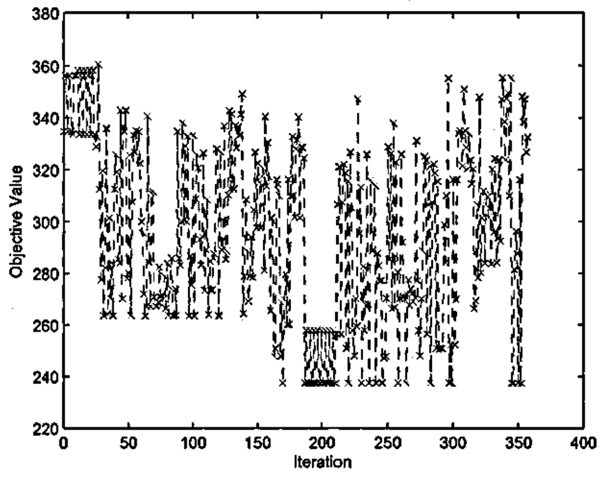
2. For the small sized problems the results gave also a good indication of the tabu search algorithm's effectiveness and its advantage over the heuristic approach. More specifically, only six percent was the mean deviation value of the best cost from the optimum solution found, whereas seven percent was the mean value for the heuristic.

3. Within the TS iterative procedure the mean deviation value of the best cost from the initial value of the objective function is significantly high. The latter indicates that within the iterative procedure and the neighborhood generation for each candidate machine instance pair, exploration of many areas occur implying the investigation of many non-visited solutions. In achieving the latter the utilisation of short term memory and more specifically the use of a tabu list played an important role.

4. An important point in the operation of the tabu search iterative procedure is its robustness when an initial solution is fed into it. More specifically, once the initial solution is generated the iterative procedure commences and it finishes when the total number of iterations is reached producing a certain output. If

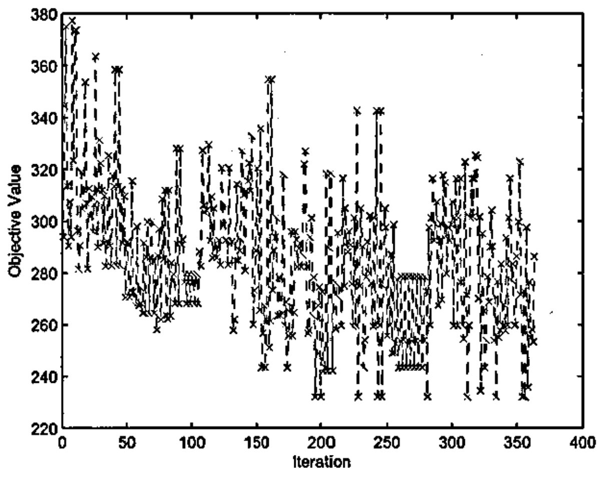


(a) Problem 22

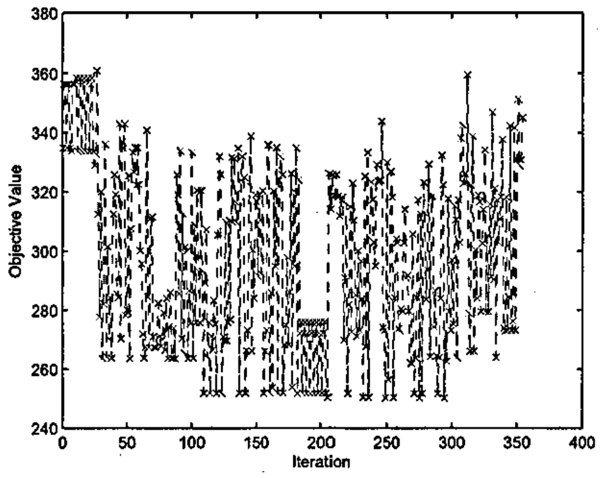


(b) Problem 23

Figure 8.5: Objective Value Evolution when $TT = 5$



(a) Problem 22



(b) Problem 23

Figure 8.6: Objective Value Evolution when $TT = 10$

another initial solution is fed into the iterative procedure, another solution will be produced whose deviation from the best known value won't differ much from the deviation obtained via the first attempt. Overall, different runs with different initial solutions produce solutions whose deviation values don't differ much in magnitude from each other; something that is not happening in the heuristic algorithm. In case, of course, that the problem is too big, e.g. problem instance twelve, multiple runs might be needed, with the same or different initial solution each time, till suitable adjustments to either the number of iterations or the value of the tabu tenure employed will be made in order for a good solution to be obtained.

5. Considering the number of iterations within the TS iterative procedure their total number depends upon the size of the problem employed each time. More specifically, the majority of the parameters within the loops, as already described in section 7.4.3, are defined in relation with the number of parts involved each time; As stated earlier, if the problem is too big then adjustment in the loop parameters is needed.

6. The processing time required for each problem for either the TS or the heuristic approach is mainly affected by the nature of the iterative procedure, the size of the problem employed and the total number of iterations specified. It would be reasonable though for the tabu search algorithm to need more computational time than the heuristic approach since neighborhood generation is taking place at each step. Indeed, as seen already, the mean CPU time needed for TS when large scale data sets were employed was 1543 seconds, whereas for the heuristic only 302 seconds. The computational requirements for both were not so different when small sized data sets were taken into account.

7. After a number of experiments the tabu list size was decided to be fixed of magnitude five to fifteen depending upon the size of the problem tested each time. For small problem instances the value of the latter was kept relatively small, i.e. five or similar, whereas for bigger problems the value of tabu tenure needed to be bigger, i.e. ten or greater in order to force the system to move onto unexplored areas. Overall, the value of tabu tenure affects the exploration of the search space, thus different values of it produce different objective value fluctuation.

8. A few comments on the part machine operation sequence. When designing the initial solution, part machine sequence was one of the most important constraints

to be included and to be taken into account for almost all the heuristic approaches implemented within the initial search. Also a number of elements, such as, which cell to choose to start allocating certain part (i.e. cell sequence identification based on maximum continuous sequence of machines relative to part machine sequence), were designed to meet this constraint, and hence make the system more effective. Later in the design of the tabu search, the part machine operation sequence acted as a key element since with every transition of a specific machine instance pair certain updates were taking place for the part machine cell allocation relative to the part machine sequence of all parts involved. Although, the part machine operation sequence made the initial solution more complex and the tabu search more difficult to converge to a value very close to the optimum after exploring a number of areas in the search space, it incorporated realism and produced a very practical CF system.

9. Finally, a point can be made concerning both heuristic approach and tabu search algorithm. The use of both is very useful for medium and large scale data sets. Especially, the tabu search algorithm is very efficient for large sized problems. Although for the small problem instances both approaches produce good results, the mathematical solver finds the optimum solutions for all problems in reasonable computational times, thus it is more preferable for these type of problems.

From the concluding remarks one important point that could arise concerns the operation on the tabu search and how its output could be improved when a re-run is attempted with an initial solution fed into it is not totally random but the actual output from the heuristic approach whose deviation value from the best known value happens to be of smaller magnitude than TS.

To examine the above problem instance eighteen will be employed whose deviation value obtained when TS is employed happens to be worse (Table 8.1, page 174) than the deviation value obtained via the heuristic approach (Table 6.3, page 137).

For problem instance eighteen, when the output from the simpler heuristic approach is fed into the TS iterative procedure the deviation obtained is of magnitude three percent, instead of seven that it was initially. More specifically, the iterative procedure starts from an initial objective value of magnitude 787.65 and fluctuates as can be seen in Figure 8.7(a) (page 190). The best value found happens to be equal to the initial, i.e. 787.65, that the TS started with. To compare the latter with the trend on the objective value when the deviation initially obtained was seven percent consult Figure

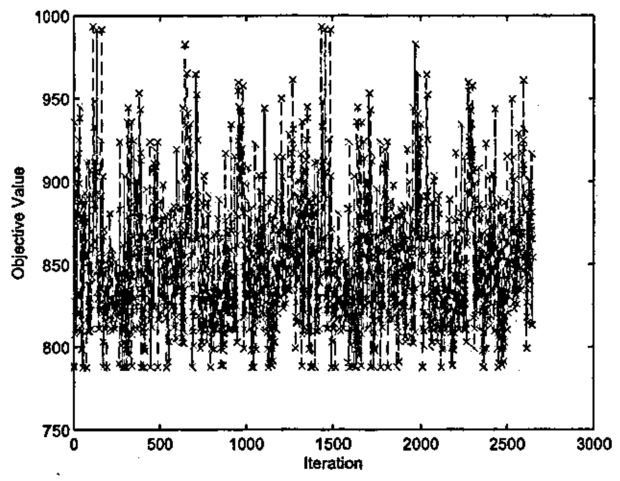
8.7(b) (page 190).

Overall, some improvement can be achieved in the deviation value obtained via TS when the simpler heuristic approach acts as an intermediate stage and its final solution is fed into the TS. On the other hand, the TS cannot achieve more improvement than its initial solution fed into it, for this particular problem, and this could be due to the part machine operation sequence. The latter seems to constrain the system to a great extent (see remark eight), thus not allowing the algorithm to converge more towards the best known value obtained via the mathematical solver.

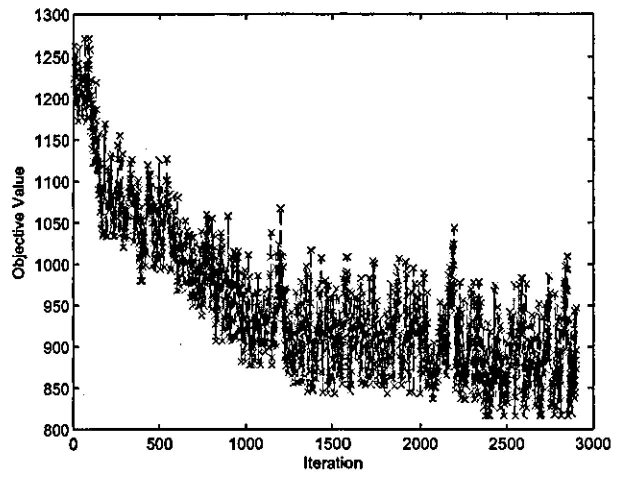
Finally, some additional comments will be made concerning remark four in order to illustrate the robustness of the tabu search algorithm over the simpler heuristic approach when the same randomly generated initial solution is fed into both, thus not allowing more than one trials to be executed. The initial solution used for problem instance one that produced within only one running trial the result shown in Table 8.1 will be used here as input to the heuristic approach. Please note that the deviation obtained for TS was of magnitude 8% and the values of both the initial solution and the best cost found were 623.73 and 481.74 units respectively. The latter can be seen in Figure 8.8(a) (page 191).

When the same initial solution is fed into the heuristic strategy the deviation produced from the best known value is 15 percent, which is much greater than the deviation initially produced from the heuristic approach (see Table 6.3) and also significantly bigger than the output generated via the TS as shown in Figure 8.8(a). For better illustration on the produced output see Figure 8.8(b) (page 191), which shows the evolution on the objective values involved. Please note that the initial solution has a value of 623.73 units, whereas the best cost found is of magnitude 510.57.

The comparison made, on the basis of employing the same initial solution for both strategies, verifies the robustness of the proposed tabu search and also its effectiveness over the simpler heuristic approach.

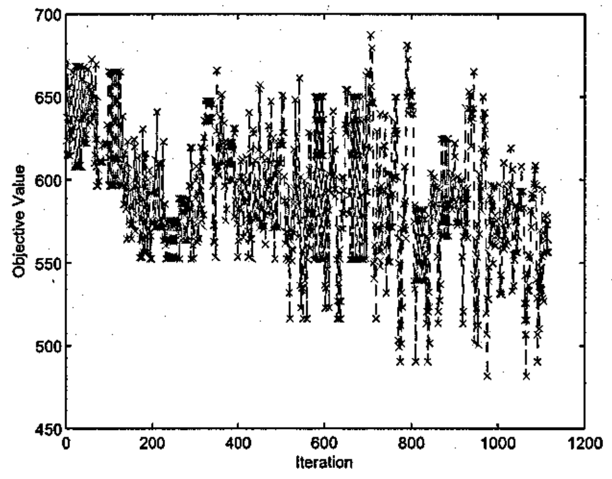


(a) Problem 18 with Heuristic's Output as Input

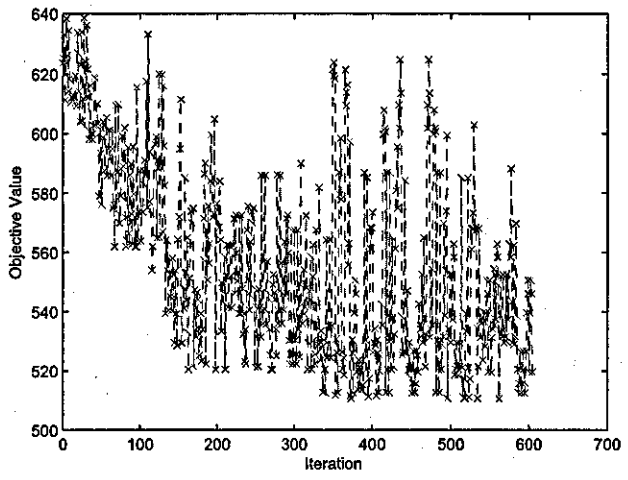


(b) Problem 18 with Random Input

Figure 8.7: Objective Value Evolution for Problem Instance 18



(a) Problem 1: Tabu Search Output



(b) Problem 1: Heuristic Output

Figure 8.8: Tabu Search Algorithm vs. Heuristic Approach

8.4 Summary

This chapter presented the computational results for the tabu search algorithm when a number of problem instances small, medium and large were taken into account. The majority of problem sizes was adapted from the literature with their additional parameters randomly generated. A number of elements were examined such as the behaviour of the iterative procedure including the number of iterations specified within the system, the initial random solution and how this affected the result when fed into the tabu search and finally the tabu list size. Also the fluctuation of the objective values within the iterative procedure was checked and a number of graphs produced to illustrate the trend in the objective value with respect to the number of iterations involved. Finally, several conclusions were drawn evaluating the tabu search effectiveness and also identifying its operational and computational advantage towards the heuristic approach.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

Cellular Manufacturing (CM) is an application of Group Technology and has emerged in the early seventies as a promising alternative of a manufacturing system, linking the advantages of both jobbing and mass production approaches applied at that time. As the name states itself CM concentrates on the design and operation of manufacturing cells, i.e. Cell Formation (CF), where parts are grouped into families and machines into cells. A number of surveys have been conducted and a number of benefits have been identified for CM. The clear benefits are reduced set-up and throughput times, reduced finished goods inventory and work-in-progress, reduced material handling cost, simplified flow of products, scheduling and improved quality.

A number of production based methods have been used for designing cellular manufacturing systems such as clustering techniques, graph partitioning approaches, mathematical programming methods, heuristic and metaheuristic approaches, and fuzzy methodologies. The main objective of clustering techniques is to group either objects or entities or attributes into clusters such that individual elements within a cluster have a high degree of "natural" association among themselves and a very little "natural association" between clusters. Graph partitioning approaches employ graph or network representation for the CF problem, where machines and/or parts are treated as vertices and the processing of parts as edges connecting these nodes. Both clustering and graph partitioning approaches share a common limitation since no production data reflecting CF real systems can be encountered at the design stage. The latter is overcome when mathematical programming models are modelled consisting of certain objective functions and constraints where parts are grouped into families and machines into cells simultaneously. Due to the nature of the combinatorial complexity of the CF systems, obtaining optimal solutions for large scale systems when mathematical programming methods are employed can be infeasible. For this reason heuristic approaches have been proposed for finding solutions close to optimum. Also metaheuristic algorithms have been developed where search is carried out on a higher level following the disciplines of an established methodology, i.e. tabu search. Finally, fuzzy theory has been utilised where fuzziness is taken under consideration via a number of methodologies.

Although, the literature for cell formation is very extensive, there are still areas of great research interest such as the development of a sophisticated mixed integer mathematical programming model representing a real and a practical CF system. The latter can further be studied with a number of advanced methodologies in order to prove the system's efficiency and stability in real situations when uncertainty is taken into account and large scale systems are utilised.

This thesis re-visits the idea of intercellular movements in CF via mathematical programming and concentrates further on the inclusion and design of other attributes, non-addressed in the current literature, so that a more realistic CF system could be developed which could be useful for the production planners. Fuzzy theory is investigated thoroughly so that a certain type of uncertainty could be measured via a traditional mathematical programming software when certain types of membership functions and aggregation operators are taken into account. Some very interesting results are obtained when fuzzy models obtained are tested and compared with the deterministic model. In addition, this thesis work concentrates on the development of an efficient initial three stage heuristic approach addressing the model in its full operation when entities like part machine utilisation amounts, multiple machine instances and part machine operation sequences are taken into account leading to a novel design for the CF problem when compared with the existing literature. Moreover, the initial methodology for generating good starting solutions is fed into an iterative heuristic approach proposed later on in this research work. The latter is tested with a significant number of data sets and the results obtained although prove the heuristic algorithms' effectiveness they also identify a few operational limitations. For overcoming any problems encountered, a metaheuristic approach and more specifically a tabu search algorithm is proposed and the iterative heuristic algorithm is extended to adapt on the principles of the tabu search. The latter proves to be suitable and very effective for the CF problem producing very good results even when large scale data sets are taken into account.

More importantly, the tabu search algorithm proves to be stable and robust while exploring the search space extensively, and be significantly better in its operation when compared with the simpler iterative heuristic algorithm.

In general, cell formation mathematical programming models concentrated on a number of production data such as part type production volume, processing times, machine capacity, tooling etc. In the author's knowledge no work in the existing literature examined the CF system when part machine operation sequence and part machine utilisation amounts and multiple machines of the same type were taken into account as key constraints while the objective function involved the minimisation of the distinct allocation of parts to cells, the part/machine set-up costs and the later revisits of parts to already visited cells. This model was built in steps by considering first, within the objective function, the distinct allocation of parts to cells, later the latter plus the machine set-up costs and finally all previous elements together with the later revisits of parts to already visited cells. This was done in order to be able to identify the differences in the model's operation when set-up costs and finally part machine operation sequence were added in a CF system where only the intercellular movements feature was taken into account. In conclusion, the better the distinct allocation of parts to cells is, when part/machine utilisation is taken into account, the better usage of part/machine set-up costs and improved configuration on the later revisits of parts to already visited cells.

Given the complete form of the mixed integer programming model with part machine operation sequence in there, the author concentrated her research work on employing fuzzy theory in order to be able to 'handle' the uncertainty appearing when trying to define the maximum number of machines allowed in each cell. Please note that although in the current system many other elements could also be examined for their fuzziness such as set-up costs and utilisation amounts, the former is chosen based upon the model's main operation which is the creation of cells with a specific number of machines in them. Moreover, the goal was also assumed to be fuzzy and the model obtained had a symmetrical form since both constraints and goal were treated in the same way. For the fuzzy incorporation within the model, triangular and linear membership functions were considered. For further manipulation and transformation of the fuzzy model to a deterministic one solvable by a mathematical solver three aggregation operations were employed: 'min', 'fuzzy and' and 'min-bounded-sum'.

All fuzzy models were assessed via XPRESS-MP with a number of randomly generated data sets where the results produced were very promising taking into account that no trials were needed till a good solution was obtained like in the deterministic case. The three operators were compared, with best been identified the 'fuzzy and' operator because of the quality of the solutions obtained, and less worse the 'min-bounded-sum' because of the limited amount of CPU time needed. The 'min' operator did not perform well despite its frequent use in the literature. Moreover, the bigger the data set employed the more useful the fuzzy models were, since the tolerance value of the constraints became bigger. Although the latter is true not very large scale models could be considered as the mathematical solver for the fuzzy models needed an excessive amount of time to produce an output because of the combinatorial complexity of CF.

The inability to address greater problem instances for the CF model led to the development of a heuristic algorithm via which solutions close to optimum even for large scale data sets could be obtained. Prior work to the development of the heuristic approach was the implementation of an efficient randomly generated initial solution where all goals and constraints of the mathematical model had to be fulfilled. The design of the initial solution was a three stage approach where machines had to be randomly allocated to cells first, parts had to be allocated to machine cells next and finally the evaluation of the objective value for the current solution obtained had to carried out. The allocation of machines to cells consisted further of two phases; the random selection of the capacity for the currently chosen cell and the random selection of the machine instance pair to be allocated to the chosen cell.

The allocation of parts to machines cells was the most complex phase to be designed since the machine operation sequence for each part together with the part/machine utilisation had to be taken into account. For allocating part to cells, parts were organised in ascending order of their total processing requirements such that the part with the maximum demand in utilisation would be allocated first. Having obtained the current part, candidate cells for allocation were organised in a way that a maximum continuous sequence relative to part machine sequence was preserved assisting the allocation process. The first cell in sequence was further explored and the machine instances of current interest were found together with their available capacity. The values of the latter together with the part/machine utilisation were combined and a number of cases had to be examined in order for the allocation to commence. The allocation was finishing when a temporary utilisation matrix holding the remaining of utilisation units from each step was equal zero. Overall, the allocation of parts to machine cells had to be designed very carefully in order to obtain a good initial solution ready to be employed by the iterative heuristic approach designed next.

The proposed heuristic algorithm was the first attempt to produce good solutions for a variety of data sets when randomly generated initial solutions were fed into it. For generating a number of solutions an iterative procedure was formed within which the parts were initially sorted in descending order of their intercellular movements. For each part a number of transitions for the machine instance pairs belonging to its machine cell sequence, relative to the current part's machine operation sequence, was explored. Two transition types either single or interchange were considered for each machine instance pair together with a part reallocation before the evaluation of current objective value was taking place. In case that the objective value produced was better than the best found so far, the current solution was stored as the best encountered and the procedure was continued from there; otherwise the iterative procedure continued from the situation where the best solution had been found.

The heuristic approach was tested with a variety of data sets, small medium and large, which were randomly generated. In order to examine the heuristic's behaviour a number of elements were determined such as the deviation of the heuristic approach from the best known solution, the processing time and the deviation of the heuristic output from the value of the objective function of the solution initially fed into the system. Overall, the iterative heuristic procedure produced good solutions in terms of the deviation values obtained and the CPU times needed, however certain limitations were identified. More specifically, a lot of running trials were needed till the algorithm was able to produce good results. The latter was happening as the iterative procedure depended mostly on the form of the initial solution and less on the transitions happening within, thus overall it was not either stable or robust. Moreover, via the transitions a certain path of exploration, following each part's machine operation sequence, was carried out with a number of movements repeating very often causing the algorithm to get stuck to local minima and not been able to escape especially when larger problem instances were taken into account.

In order to overcome the heuristic algorithm's limitations and to obtain a more effective approach an extension was proposed which was based on a higher level strategy, i.e. a metaheuristic which explores the search space very efficiently and finds (near) optimum solutions. In the open literature a number of metaheuristic algorithms exist with the most popular such as Simulated Annealing (SA), Genetic Algorithms (GAs), Ant Colony Optimisation (ACO) and Tabu Search (TS). Both GAs and ACO share a common characteristic as they start searching the space not from a single point but from a population of points in parallel which is not very useful for the CF problem since only one randomly generated initial solution was created. Moreover, both SA and TS search the space starting from a single point, however TS unlike SA has no stochastic elements. Because of the latter and the fact that TS allows a considerable amount of freedom to be developed and adjust to the problem requirements it was chosen to be the candidate metaheuristic for the development of the new proposed algorithm.

In a similar way with the heuristic approach initially developed, the proposed tabu search algorithm commenced once an initial solution randomly generated was fed into its iterative procedure where the search of the space began until a pre-specified number of iterations was reached. Within the iterative procedure a number of transitions for each machine instance pair, belonging each time to certain part's machine operation sequence, was considered. For each machine instance its neighborhood solutions were found together with their corresponding values which were shorted in ascending order. Moreover, for every neighboring solution a temporarily tabu list was updated by storing all related attributes for current machine instance pair. Later a decision was made on which move to become admissible from the generated neighborhood. The sorted sequence with the objective values was examined by checking first whether the first entry has a value less than the global best stored. If that was the case then the corresponding move becomes admissible despite its tabu status because of the aspiration criterion employed. In case that none of the latter was happening the search of which move to become admissible was concentrated locally and the decision was made based on which move's tabu status is not tabu active. Every time a move was admissible a tabu list, permanent this time, was updated by storing all attributes of the implemented moves and only the forward moves of the remaining ones. Moreover, every move was kept tabu active for a certain number of iterations which were pre-specified by the decision maker. The magnitude of the latter was decided to take values between five and fifteen depending upon the size of the problem instance employed each time.

Computational results for the tabu search algorithm were also presented when the problem instances initially generated for the heuristic approach were employed. A key observation was concentrated on the effectiveness of the tabu search algorithm and its stability no matter what the initial solution was. In addition, the space was searched extensively via the iterative procedure avoiding frequent repetition of moves and getting stuck to local minima even when larger problem instances were taken into account. Overall, the tabu search algorithm performed better than the heuristic approach in terms of the quality of the solutions produced despite the inclusion of the part machine operation and the added complexity that the latter was causing.

9.2 Suggestions for Future Work

For the purpose of this study the considered aspects of fuzziness concentrated only on the maximum number of machines allowed in a cell because of the operation of the mathematical programming model which is the creation of cells with a specific number of machines in them. An extension of the concept would be to introduce fuzziness in other parameters of the mathematical model such as the part/machine utilisation amounts and the set-up costs. Moreover, additional objectives such as part/machine utilisation could also be added in the mathematical model and be examined further as a fuzzy equation converting the problem into a multiple fuzzy linear objective function.

The design of the initial randomly generated solution consisted of a very important stage that of part allocation. For allocating parts to machine cells a 'single' path was followed with specific rules related to remaining capacities and candidate part machine utilisations taking into account the part machine operation sequence at each step. Based upon this, an extension to the part allocation could be proposed by relaxing the key constraint, i.e. the part machine operation sequence, within the part allocation process by searching first the machine instances in the system. The latter could be achieved by performing an overall search of the machine instances in the system that could best accommodate current part before the actual allocation which will be carried out in agreement with the part machine sequence.

From the results obtained via the tabu search algorithm it was seen that the method provides an efficient way of tackling the cell formation problem when part machine operation sequence is included and when a variety of problem instances and especially large scale data sets are taken into account. Moreover, from the performance of the fuzzy models formed to measure the uncertainty encountered in the maximum number of machines allowed in a cell it was proved their advantage towards the deterministic models. An important next step would be the combination of the tabu search algorithm and the fuzzy models so that flexibility and consideration of large scale models within a robust and stable system could be achieved incorporating more realism within the CF problem.

As seen already, the heuristic approach can work as an intermediate mechanism for improving the solution produced via the tabu search when the output of the former is fed into the latter. This was applied only when the initial output from the tabu search happened to be worse than those obtained via the heuristic approach. The above could be generalised leading to a continuous combination of the tabu search with another heuristic or metaheuristic to form a hybrid search approach for the cell formation problem.

Bibliography

[ABC97] N. Aljaber, W. Baek, and C.L. Chen. A tabu search approach to the cell formation problem. *Computers and Industrial Engineering*, 32(1):169–185, 1997.

[ACGV91] R.G. Askin, J.B. Creswell, J.B. Goldberg, and A.J. Vakharia. A hamiltonian path approach to reordering the part-machine matrix for cellular manufacturing. *International Journal of Production Research*, 29(6):1081–1100, 1991.

[And73] M.R. Anderberg. *Cluster Analysis for applications.* New York: Academic Press, New York, U.S.A, 1973.

[ARS97] G.K. Adil, D. Rajamani, and D. Strong. Assignment allocation and simulated annealing algorithms for cell formation. *IIE Transactions*, 29(11):53–67, 1997.

[AS93] R. Askin and C. Standridge. *Modelling and Analysis of Manufacturing Systems.* John Wiley & Sons, New York, U.S.A, 1993.

[AS98] A. Agarwal and J. Sarkis. A review and analysis of comparative performance studies on functional and cellular manufacturing layouts. *Computers and Industrial Engineering*, 34(1):77–89, 1998.

[ASV97] R.G. Askin, H.M. Selim, and A.J. Vakharia. A methodology for designing flexible cellular manufacturing systems. *IIE Transactions*, 29:599–610, 1997.

[BAT92] D. Ben-Arieh and E. Triantaphyllou. Quantifying data for group technology with weighted fuzzy features. *International Journal of Production Research*, 30(6):1285–1299, 1992.

[BC91] W.J. Boe and C.H. Cheng. A close neighbour algorithm for designing cellular manufacturing systems. *International Journal of Production Research*, 29(10):2097–2116, 1991.

[Bez81] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithm.* Plenum Press, New York, U.S.A, 1981.

[BFHS88] J. Blazewicz, G. Finke, R. Haupt, and G. Schmidt. New trends in machine scheduling. *European Journal of Operational Research*, 37(3):303–317, 1988.

[Boc91] F.F. Boctor. A linear formulation of the machine-part cell formation problem. *International Journal of Production Research*, 29(2):343–356, 1991.

[BR03] C. Blum and A. Roli. Metaheuristics in combinatorial optimisation: Overview and conceptual comparison. *ACM Computing Surveys*, 35(2):268–308, 2003.

[Bur63] J.L. Burbidge. Production flow analysis. *The Production Engineer*, 42:742, 1963.

[Bur75] J.L. Burbidge. *The Introduction of Group Technology.* John Wiley & Sons, New York, U.S.A, 1975.

[Bur89] J.L. Burbidge. *Production flow analysis for planning group technology.* Oxford Science Publications, UK, 1989.

[BZ70] R.E. Bellman and L.A. Zadeh. Decision-making in a fuzzy environment. *Management Sciense*, 17(4):B141–B164, Dec. 1970.

[CCI02] W.M. Chan, C.Y. Chan, and W.H. Ip. A heuristic algorithm for machine assignment in cellular layout. *Computers and Industrial Engineering*, 44:49–73, 2002.

[CH81] T. Congawave and I. Ham. Cluster analysis applications for group technology manufacturing systems. *Proceedings, North American Manufacturing Research Conference (NAMRC), 9TH, Dearborn*, pages 503–508, 1981.

[CH91] C.H. Chu and J.C. Hayya. A fuzzy clustering approach to manufacturing cell formation. *International Journal of Production Research*, 29(7):1475–1487, 1991.

[Cho88] F. Choobineh. A framework for the design of cellular manufacturing systems. *International Journal of Production Research*, 26(7):1161–1172, 1988.

[CKM84] S. Chanas, W. Kolodziejczyk, and A. Machaj. A fuzzy approach to the transportation problem. *Fuzzy Sets and Systems*, 13:211–221, 1984.

[CM82] H.M. Chan and D.A. Milner. Direct clustering algorithm for group formation in cellular manufacturing. *Journal of Manufacturing Systems*, 1(1):65–75, 1982.

[CR86a] M.P. Chandrasekharan and R. Ragagopalan. An ideal-seed non-hierarchical clustering algorithm for cellular manufacturing. *International Journal of Production Research*, 24(2):451–464, 1986.

[CR86b] M.P. Chandrasekharan and R. Ragagopalan. MODROC: an extension of rank order clustering for group technology. *International Journal of Production Research*, 24(5):1221–1233, 1986.

[CR87] M.P. Chandrasekharan and R. Ragagopalan. ZODIAC - an algorithm for concurrent formation of part families and machine-cells. *International Journal of Production Research*, 25(6):835–850, 1987.

[CS97] S. Collett and R. Spicer. Improving productivity through cellular manufacturing. *Production and Inventory Management Journal*, 36(1):71–75, 1997.

[DAGP90] J.L. Deneubourg, S. Aron, S. Goss, and J.M. Pasteels. The self organising exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, 3:159–168, 1990.

[Dar87] J. Darzentas. *On Fuzzy Location Model, in Optimisation Models Using Fuzzy Sets and Possibility Theory, Kacprzyk, J. and Orlovski, S.A.(eds.).* D. Reidel, Dordrecht, MA., 1987. pp 328-341.

[DDCS99] M. Dorigo, G. Di Caro, and M. Sampels. Ant algorithms for discrete optimisation. *Artificial Life*, 5(2):137–172, 1999.

[DE90] E.L. Deporter and K.P. Ellis. Optimisation of project networks with goal programming and fuzzy linear programming. *Computers and Industrial Engineering*, 10:500–504, 1990.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimisation by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41, 1996.

[Dom90] J. Dombi. Membership function as an evaluation. *Fuzzy Sets and Systems*, 35:1–21, 1990.

[DS99] G. Da Silveira. A methodology of implementation of cellular manufacturing. *International Journal of Production Research*, 37(2):467–479, 1999.

[DW80] J. De Witte. The use of similarity coefficients in production flow analysis. *International Journal of Production Research*, 18(4):503–514, 1980.

[EET72] I.F.K. El-Essawy and J. Torrance. Component flow analysis - an effective approach to production systems design. *The Production Engineer*, 51:165, 1972.

[EKW89] G.W. Evans, W. Karwowski, and M.R. Wilhelm. *Applications of Fuzzy Set Methodologies in Industrial Engineering.* Elsevier, New York, 1989.

[FFW06] L.R. Foulds, A.P. French, and J.M. Wilson. The sustainable cell formation problem: manufacturing cell creation with machine modification costs. *Computers and Operations Research*, 33:1010–1032, 2006.

[FWB87] T. Fry, M.G. Wilson, and M. Breen. A successful implementation of group technology & cell manufacturing. *Production and Inventory Management Journal*, 28(3):4–6, 1987.

[GB97] A. Gill and R. Bector. A fuzzy linguistic approach to data quantification and construction of distance measures for the part family formation problem. *International Journal of Production Research*, 35(9):2565–2578, 1997.

[GGKS96] T. Gupta, M. Gupta, A. Kumar, and C. Sundaram. A genetic algorithm approach to cell composition and layout design problems. *International Journal of Production Research*, 34(2):447–482, 1996.

[GL97] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Norwell, MA, 1997.

[Glo77] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

[Glo86] F. Glover. Futute paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[Glo89a] F. Glover. Tabu search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glo89b] F. Glover. Tabu search - Part II. *ORSA Journal on Computing*, 2(1):4–32, 1989.

[Glo90] F. Glover. Tabu search: a tutorial. *Interfaces*, 20:74–94, 1990.

[GNP98] M. Gravel, A.L. Nsakanda, and W. Price. Efficient solutions to the cell-formation problem with multiple routings via a double-loop genetic algorithm. *European Journal of Operational Research*, 109:286–298, 1998.

[GRC95] N.N.Z. Gindy, T.M. Ratchev, and K. Case. Component grouping for GT applications - a fuzzy clustering approach with validity measure. *International Journal of Production Research*, 33(9):2493–2509, 1995.

[GS84] T. Greene and R. Sadowski. A review of cellular manufacturing assumptions, advantages, and design techniques. *Journal of Operations Management*, 4(2):85–97, 1984.

[GS90] T. Gupta and H. Seifoddini. Production data based similarity coefficient for machine-component grouping decisions in the design of a cellular manufacturing system. *International Journal of Production Research*, 28(7):1247–1269, 1990.

[Gup93] T. Gupta. Design of manufacturing cells for flexible environment considering alternative routing. *International Journal of Production Research*, 31(6):1259–1273, 1993.

[GYZ02] G. Gutin, A. Yeo, and A. Zverovitch. Traveling salesman should not be greedy: domination analysis of greedy type heuristics for the tsp. *Discrete Applied Mathematics*, 117:81–86, 2002.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor, University of Michigan, Michigan, 1975.

[HW89] N.L. Hyer and U. Wemmerlov. Group technology in the US manufacturing industry: A survey of current practices. *International Journal of Production Research*, 27:1287–1304, 1989.

[Hye84] N.L. Hyer. The potential of group technology for US manufacturing. *Journal of Operations Management*, 4(3):183–202, 1984.

[Iri68] M. Iri. On the sunthesi of loop and cutset matrices and related problems. RAAG *Memoirs*, 4(A-XII):376–410, 1968.

[Jac08] P. Jaccard. Nouvelles recherches sur la distribution florale. *Bull. Soc. Vaud. Sci. Nat.*, 44:223–270, 1908.

[JBK07] T.L. James, E.C. Brown, and K.B. Keeling. A hybrid grouping genetic algorithm for the cell formation problem. *Computers and Operations Research*, 34:2059–2079, 2007.

[JKC95] J.A. Joines, R.E. King, and C.T. Culbreth. A comprehensive review of production-oriented manufacturing cell formation techniques. *International Journal of Flexible Automation and Intelligent Manufacturing*, 3:254–264, 1995.

[JNN98] J.G. Jayakrishnan Nair and T.T. Narendran. CASE: a clustering algorithm for cell formation with sequence data. *International Journal of Production Research*, 36(1):157–179, 1998.

[JNN99] J.G. Jayakrishnan Nair and T.T. Narendran. ACCORD: a bicriterion algorithm for cell formation using ordinal and ratio level data. *International Journal of Production Research*, 37(3):539–556, 1999.

[KB90] J. Kasilingham and S. Bhole. Cell formation in flexible manufacturing systems under resource constraints. *Computers and Industrial Engineering*, 19:437–441, 1990.

[KBB04] C.O. Kim, J.G. Baek, and J.K. Baek. A two-phase heuristic algorithm for cell formation problems considering alternative part routes and machine sequences. *International Journal of Production Research*, 42(18):3911–3927, 2004.

[KC87] A. Kusiak and W.S. Chow. Efficient solving of the group technology problem. *Journal of Manufacturing Systems*, 6(2):117–124, 1987.

[KC90] K.R. Kumar and M.P. Chandrasekharan. Grouping efficacy: a quantitative criterion for goodness of block diagonal forms of binary matrices in group technology. *International Journal of Production Research*, 28(2):233–243, 1990.

[KE86] W. Karwowski and G.W. Evans. Fuzzy concepts in production management research: a review. *International Journal of Production Research*, 24:129–147, 1986.

[KGJV83] S. Kirkpatrick, C.D. Gelati Jr, and M.P. Vecchi. Optimisation by simulated annealing. *Sciense*, 220(4598):671–680, 1983.

[KH87] A. Kusiak and S.S. Heragu. Group technology. *Computers in Industry*, 9:83–91, 1987.

[Kin80] J.R. King. Machine-component grouping in production flow analysis: an approach using rank order clustering algorithm. *International Journal of Production Research*, 18(2):213–232, 1980.

[KKV86] K.R. Kumar, A. Kusiak, and A. Vannelli. Grouping of parts and components in flexible manufacturing systems. *European Journal of Operational Research*, 24:387–397, 1986.

[KL70] B.W. Kernighan and S. Lin. A heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[KLL93] M.H. Kim, J.H. Lee, and Y.J. Lee. Analysis of fuzzy operators for high quality information retrieval. *Information Processing Letters*, 46:251–256, July 1993.

[KN82] J.R. King and V. Nakornchai. Machine-component group formation in group technology: review and extension. *International Journal of Production Research*, 20(2):117–133, 1982.

[Koz92] J.H. Koza. *Genetic Programming II: On the programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, 1992.

[Kus87] A. Kusiak. The generalized group technology concept. *International Journal of Production Research*, 25(4):561–569, 1987.

[Kus90] A. Kusiak. *Intelligent Manufacturing Systems.* Prentice Hall, Englewood Cliffs, N.J., 1990.

[LDKN96] T.L. Lin, M.M. Dessouky, K.R. Kumar, and S.M. Ng. A heuristic-based procedure for the weighted production - cell formation problem. *IIE Transactions*, 28:579–589, 1996.

[Leb81] H. Leberling. On finding compromise solutions in multicriteria problems using the fuzzy min-operator. *Fuzzy Sets and Systems*, 6:105–118, 1981.

[LH92a] Y.-J. Lai and C.-L. Hwang. *Fuzzy Mathematical Programming - Methods and Applications.* Springer-Verlag, Berlin, 1992.

[LH92b] Y.-J. Lai and C.-L. Hwang. Interactive fuzzy linear programming. *Fuzzy Sets and Systems*, 45:169–183, 1992.

[LHZ95] G. Levasseur, M. Helms, and A. Zink. A conversion from a functional to a cellular manufacturing layout at STEWARD, INC. *Production and Inventory Management Journal*, 36(3):37–42, 1995.

[LL91] R.J. Li and E.S. Lee. An exponential membership function for fuzzy multiple objective linear programming. *Computers for Mathematical Applications*, 22(12):55–60, 1991.

[LSC83] S. Lou, Z. Sun, and H. Chen. *Fuzzy Mathematics.* Scientific Publishing House, China, 1983.

[Luh82] M.K. Luhandjula. Compensatory operators in fuzzy linear programming with multiple objectives. *Fuzzy Sets and Systems*, 8:245–252, 1982.

[LVM79] J.G. Lee, W. Vogt, and M. Mickle. Optimal decomposition of large scale networks. *Institute of Electrical and Electronic Engineers Transactions on Systems Man and Cybernetics*, 9:369–375, July 1979.

[LW06] D. Lei and Z. Wu. Tabu search for multiple-criteria manufacturing cell design. *International Journal of Advanced Manufacturing Technology*, 28:950–956, 2006.

[Mac67] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1:281–297, 1967.

[MAT02] A.M. Mukattash, M.B. Adil, and K.K. Tahboub. Heuristic approaches for part assignment in cell formation. *Computers and Industrial Engineering*, 42:329–341, 2002.

[McA72] J. McAuley. Machine grouping for efficient production. *Production Engineer*, 51(2):53–57, 1972.

[Mit66] S.P. Mitrofanov. *The Scientific Principles of Group Technology.* National Lending Library Translation, Boston Spa, Yorkshire, U.K., 1966.

[Mje86] K.M. Mjelde. Fuzzy resource allocation. *Fuzzy Sets and Systems*, 19:239–250, 1986.

[MRR$^+$53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of the state calculations by fast computing machines. *Journal for Chemical Physics*, 21:1087–1092, 1953.

[MS97] A. Masnata and L. Settineri. An application of fuzzy clustering to cellular manufacturing. *International Journal of Production Research*, 35(4):1077–1094, 1997.

[MSW72] W.T. McCormick, P.J. Schweitzer, and T.W. White. Problem decomposition and data reorganisation by a clustering technique. *Operations Research*, 20:993–1009, 1972.

[MWW00] K.L. Mak, Y.S. Wong, and X.X. Wang. An adaptive genetic algorithm for manufacturing cell formation. *International Journal of Advanced Manufacturing Technology*, 16:491–497, 2000.

[Ng93] S.M. Ng. Worst-case analysis of an algorithm for cellular manufacturing. *European Journal of Operational Research*, 69:384–398, 1993.

[OL96] I. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623, 1996.

[OLP79] E. Olivia-Lopez and G.F. Purcheck. Load balancing for group technology planning and control. *International Journal of Machine Tools and Manufacture*, 19:259–274, 1979.

[Pur75] G.F. Purcheck. A linear programming method for combinatorial group of an incomplete power set. *Journal of Cybernetics*, 5:51–76, 1975.

[Pur85] G.F. Purcheck. Machine component group formation: a heuristic method for flexible production cells and flexible manufacturing systems. *International Journal of Production Research*, 23(5):911–943, 1985.

[RB75] R. Ragagopalan and J.L. Batra. Design of cellular production systems: A graph-theoretic approach. *International Journal of Production Research*, 13(6):567–579, 1975.

[RR01] K.S. Ravichandran and K.C.S. Rao. A new approach to fuzzy part-family formation in cellular manufacturing systems. *International Journal of Advanced Manufacturing Technology*, 18:591–597, 2001.

[Saa80] T.L. Saaty. *The Analytic Hierarchy Process.* McGraw-Hill International, 1980.

[San90] S. Sankaran. Multiple objective decision making approach to cell formation - a goal programming model. *International Journal of Production Research*, 13:71–81, 1990.

[SAV98] H.M. Selim, R.G. Askin, and A.J. Vakharia. Cell formation in group technology: Review evaluation and directions for future research. *Computers and Industrial Engineering*, 34(1):3–20, 1998.

[SD95] H. Seifoddini and M. Djassemi. Merits of the production volume based similarity coefficient in machine-cell formation. *Journal of Manufacturing Systems*, 14(1):35–44, 1995.

[Sht89] A. Shtub. Modelling group technology cell formation as a generalized assignment problem. *International Journal of Production Research*, 27(5):775–782, 1989.

[Sin93] N. Singh. Design of cellular manufacturing systems: An invited review. *European Journal of Operational Research*, 69:284–291, 1993.

[SK93] S. Sankaran and R.G. Kasilingam. On cell size and machine requirements planning in group technology systems. *European Journal of Operational Research*, 69:373–383, 1993.

[SN91] G. Srinivasan and T.T. Narendran. GRAPHICS: a non-hierarchical clustering algorithm for group technology. *International Journal of Production Research*, 29(3):463–478, 1991.

[Sne57] P.H.A. Sneath. The application of computers to taxonomy. *Journal of General Microbiology*, 17:201–206, 1957.

[SNM90] G. Srinivasan, T.T. Narendran, and B. Mahadevan. An assignment model for the part-families problem in group technology. *International Journal of Production Research*, 28:145–152, 1990.

[SP78] G. Sommer and M.A. Pollatschek. *A Fuzzy Programming Approach to an Air Pollution Regulation Problem In Cybernetics and Systems Research, Trappl, R. and Klir, G.J.(eds.)*, volume 3. Hemisphere Pub. Corp., London, 1978. pp 303-323.

[Sri94] G. Srinivasan. A clustering algorithm for machine cell formation in group technology using minimum spanning trees. *International Journal of Production Research*, 32(9):2149–2158, 1994.

[SS03] K. Spiliopoulos and S. Sofianopoulou. Designing manufacturing cells: a staged approach and a tabu search algorithm. *International Journal of Production Research*, 41(11):2531–2546, 2003.

[Sta85] L.E. Stanfel. Machine clustering for economic production. *Engineering Costs and Production Economics*, 9:73–81, 1985.

[SVS04a] M. Solimanpur, P. Vrat, and R. Shankar. Ant colony optimisation algorithm to the inter-cell layout problem in cellular manufacturing. *European Journal of Operational Research*, 157:592–606, 2004.

[SVS04b] M. Solimanpur, P. Vrat, and R. Shankar. A multi-objective genetic algorithm approach to the design of cellular manufacturing systems. *International Journal of Production Research*, 42(7):1419–1441, 2004.

[TCB97] C.C. Tsai, C.H. Chu, and T.A. Barta. Modelling and analysis of a manufacturing cell formation problem with fuzzy mixed-integer programming. *IIE Transactions*, 29:533–547, 1997.

[VC97] A.J. Vakharia and Y.L. Chang. Cell formation in group technology: a combinatorial search approach. *International Journal of Production Research*, 35(7):2025–2043, 1997.

[VGFT06] E. Vila Gonçalves Filho and A.J. Tiberti. A group genetic algorithm for the machine cell formation problem. *International Journal of Production Economics*, 102:1–21, 2006.

[VK86] A. Vannelli and K.R. Kumar. A method for finding minimal bottleneck cells for grouping part-machine families. *Interantional Journal of Production Research*, 24(2):387–400, 1986.

[VK87] A. Vannelli and K.R. Kumar. Strategic subcontracting for efficient disaggregated manufacturing. *International Journal of Production Research*, 25(12):1715–1728, 1987.

[VMOR99] S. Voß, S. Martello, I.H. Osman, and C. Roucairol. *Meta-Heuristics: Advances and trends in local search paradigms for optimisation.* Kluwer, Boston, 1999.

[VN92] V. Venugopal and T.T. Narendran. A genetic algorithm approach to the machine-component grouping problem with multiple objectives. *Computers and Industrial Engineering*, 22(4):469–480, 1992.

[VW90] A.J. Vakharia and U. Wemmerlov. Designing a cellular manufacturing system: a materials flow approach based on operation sequences. *IIE Transactions*, 22(1):84–97, 1990.

[WCHWY07] X. Wu, C. Chao-Hsien, Y. Wang, and W. Yan. A genetic algorithm for cellular manufacturing design and layout. *European Journal of Operational Research*, 181:156–167, 2007.

[Wer87] B. Werners. An interactive fuzzy programming system. *Fuzzy Sets and Systems*, 23:131–147, 1987.

[Wer88] B.M Werners. *Aggregation models in mathematical programming, in Mathematical Models for Decision Support, Mitra. G. and Greenberg, H.J.(eds.).* Springer-Verlag, Berlin, Heidelberg, 1988. pp 295-305.

[WG90] J.C. Wei and N. Gaither. An optimal model for cell formation decisions. *Decision Sciences*, 21:416–433, 1990.

[WH86] U. Wemmerlov and N.L. Hyer. Procedures for the part/machine group identification problem in cellular manufacturing. *Journal of Operations Management*, 6:125–147, 1986.

[WH89] U. Wemmerlov and N.L. Hyer. Cell manufacturing in the US industry: A survey of users. *International Journal of Production Research*, 27:1511–1530, 1989.

[WJ97] U. Wemmerlov and D.J. Johnson. Cellular manufacturing at 46 user plants: Implementation experiences and performance improvements. *International Journal of Production Research*, 35(1):29–49, 1997.

[WL04] Y. Won and K.C. Lee. Modified p-median approach for efficient GT cell formation. *Computers and Industrial Engineering*, 46:495–510, 2004.

[WLW04] T.-H. Wu, C. Low, and W.-T. Wu. A tabu search approach to the cell formation problem. *International Journal of Advanced Manufacturing Technology*, 23:916–924, 2004.

[WZ78] G. Wiedey and H.J. Zimmermann. Media selection and fuzzy linear programming. *Journal of Operational Research Society*, 31:342–349, 1978.

[XM] Xpress-MP. Dash optimisation. *Blisworth, Northamptonshire, UK.*

[XW89] H. Xu and H.P. Wang. Part family formation for GT applications based on fuzzy mathematics. *International Journal of Production Research*, 27(9):1637–1651, 1989.

[YI91] T. Yang and J.P. Ignizio. Fuzzy programming with nonlinear membership functions: Piecewise linear approximation. *Fuzzy Sets and Systems*, 41:39–53, 1991.

[Zad62] L.A. Zadeh. From circuit theory to system theory. *Proceedings of Institute of Radio Engineering*, 50:856–865, 1962.

[Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[ZHR95] Z. Zhu, R.B. Heady, and S. Reiners. An efficient zero-one formulation of the cell formation problem. *Computers and Industrial Engineering*, 28(4):911–916, 1995.

[Zim78] H.J. Zimmermann. Fuzzy programming and linear programming with several objective functions. *Fuzzy Sets and Systems*, 1:45–55, 1978.

[Zim83] H.J. Zimmermann. Fuzzy mathematical programming. *Computers and Operations Research*, 10(4):291–298, 1983.

[Zim85] H.J. Zimmermann. Applications of fuzzy set theory to mathematical programming. *Information Scienses*, 36:29–52, 1985.

[Zim88a] H.J. Zimmermann. *Fuzzy Sets Theory - and Inference Mechanism, in NATO ASI Series, Mathematical Models for Decision Support, Mitra, G.(eds.)*, volume F48. Springer-Verlag, Berlin, Heidelberg, 1988. pp 727-741.

[Zim88b] H.J. Zimmermann. *Interactive decision support for semi-structured mathematical programming problems, in NATO ASI Series, Mathematical Models for Decision Support, Mitra, G.(eds.)*, volume F48. Springer-Verlag, Berlin, Heidelberg, 1988. pp 307-319.

[Zim91] H.J. Zimmermann. *Fuzzy Set Theory and Its Applications.* Kluwer Academic Publishers, Boston, MA., 1991.

[ZW92] C. Zhang and H.P. Wang. Concurrent formation of part families and machine cells based on the fuzzy set theory. *Journal of Manufacturing Systems*, 11(1):61–67, 1992.

[ZZ88] H.J. Zimmermann and P. Zysno. Decisions and evaluations by hierarchical aggregation of information. *Fuzzy Sets and Systems*, 10:243–266, 1988.

Appendix A

CF Mathematical Models - XPRESS-MP

A.1 Initial Model: Solution

```
Cost = 12

x(i,j,q) values:
x(1 2 4) 0.3
x(1 4 4) 0.1
x(1 8 4) 0.1
x(2 4 2) 1
x(2 5 2) 0.5
x(2 7 3) 0.9
x(2 9 3) 0.1
x(2 10 1) 0.6
x(3 3 3) 0.9
x(3 8 4) 0.2
x(3 9 3) 0.1
x(4 2 1) 0.6
x(4 2 4) 0.6
x(4 4 2) 0.5
x(4 4 4) 0.4
x(4 5 2) 0.5
x(4 6 1) 0.4
x(5 1 3) 0.4
x(5 2 1) 0.2
x(5 4 2) 0.8
x(5 5 2) 1.1
x(5 6 1) 1
x(5 7 3) 0.6
x(5 8 4) 1
x(5 9 3) 1
x(6 2 1) 0.7
x(6 7 3) 0.8
x(6 10 1) 0.3
x(7 4 2) 0.2
x(7 5 2) 0.2

v(q) values:
v(1) 1
v(2) 1
v(3) 1
v(4) 1

y(i,k,q) values:
y(1 1 4) 1
y(2 1 1) 1
y(2 2 2) 1
y(2 3 2) 1
y(2 4 3) 1
y(3 1 3) 1
y(3 2 4) 1
y(4 1 1) 1
y(4 2 2) 1
y(4 3 4) 1
y(5 1 1) 1
y(5 2 1) 1
y(5 3 2) 1
y(5 4 2) 1
y(5 5 3) 1
y(5 6 3) 1
y(5 7 4) 1
y(6 1 1) 1
y(6 2 3) 1
y(7 1 2) 1

w(j,q) values:
w(1 3) 1
w(2 1) 1
w(2 4) 1
w(3 3) 1
w(4 2) 1
w(4 4) 1
w(5 2) 1
w(6 1) 1
w(7 3) 1
w(8 4) 1
w(9 3) 1
w(10 1) 1
```

A.2 Extended Model: Solution

```
Cost = 219.92

x(i,j,q) values:
x(1 2 2) 0.3
x(1 4 2) 0.1
x(1 8 2) 0.1
x(2 4 3) 1
x(2 5 3) 0.5
x(2 7 4) 0.9
x(2 9 4) 0.1
x(2 10 1) 0.6
x(3 3 4) 0.9
x(3 8 2) 0.2
x(3 9 4) 0.1
x(4 2 1) 0.6
x(4 2 2) 0.6
x(4 4 2) 0.4
x(4 4 3) 0.5
x(4 5 3) 0.5
x(4 6 1) 0.4
x(5 1 4) 0.4
x(5 2 1) 0.2
x(5 4 3) 0.8
x(5 5 3) 1.1
x(5 6 1) 1
x(5 7 4) 0.6
x(5 8 2) 1
```

```
x(5 9 4) 1
x(6 2 1) 0.7
x(6 7 4) 0.8
x(6 10 1) 0.3
x(7 4 3) 0.2
x(7 5 3) 0.2

v(q) values:
v(1) 1
v(2) 1
v(3) 1
v(4) 1

s(i,j,q) values:
s(1 2 2) 1
s(1 4 2) 1
s(1 8 2) 1
s(2 4 3) 1
s(2 5 3) 1
s(2 7 4) 1
s(2 9 4) 1
s(2 10 1) 1
s(3 3 4) 1
s(3 8 2) 1
s(3 9 4) 1
s(4 2 1) 1
s(4 2 2) 1
s(4 4 2) 1
s(4 4 3) 1
s(4 5 3) 1
s(4 6 1) 1
s(5 1 4) 1
s(5 2 1) 1
s(5 4 3) 1
s(5 5 3) 2
s(5 6 1) 1
s(5 7 4) 1
s(5 8 2) 1
s(5 9 4) 1
s(6 2 1) 1
s(6 7 4) 1
s(6 10 1) 1
s(7 4 3) 1
s(7 5 3) 1

y(i,k,q) values:
y(1 1 2) 1
y(2 1 1) 1
y(2 2 3) 1
y(2 3 3) 1
y(2 4 4) 1
y(3 1 2) 1
y(3 2 4) 1
y(4 1 1) 1
y(4 2 2) 1
y(4 3 3) 1
y(5 1 1) 1
y(5 2 1) 1
y(5 3 2) 1
y(5 4 3) 1
y(5 5 3) 1
y(5 6 4) 1
y(5 7 4) 1
y(6 1 1) 1
y(6 2 4) 1
y(7 1 3) 1

w(j,q) values:
w(1 4) 1
w(2 1) 1
w(2 2) 1
w(3 4) 1
w(4 2) 1
w(4 3) 1
w(5 3) 1
w(6 1) 1
w(7 4) 1
w(8 2) 1
w(9 4) 1
w(10 1) 1
```

A.3 Complete Model - XPRESS-MP Code

```
(!*************************************************************
   Mosel CF Problem
   ================
   Objective: minimise distinct allocation of parts to cells,
              part/machine set-up costs and later revisits of
              parts to already visited cells
 *************************************************************!)
model "CELL/BLOCKS FORMATION"

uses "mmxprs"

declarations

! no. of parts in the plant
NP = 10
PARTS = 1..NP

! no. of machines in the plant
NM = 7
MACH = 1..NM

! number of machines operations
ZOPER = 1..5
```

```
! number of machines of the same type
KMAX = 1..7

! maximum number of cells to be created
 NC = 5
 CELLS = 1..NC

! minimum and maximum restrictions on the number of machines allowed
in a cell
UPMAX = 1..2

! biggest amount of machine utilisation used
UTILMAX: real

! smallest amount of machine utilisation used
UTILMIN: real

! utilization of part j to be processed in machine i
UTIL: array(MACH,PARTS) of real

! set-up time of machine i needed to process part j SETUP:
array(MACH,PARTS) of real

! number of different machine operations required by part j
ZTYPES: array(PARTS) of integer

! for part j the zth machine operation in sequence
L: array(PARTS,ZOPER) of integer

! array for indexing the no. of machines of the  same type
KTYPES: array(MACH) of integer

! max. and min. limits on the no. of machines allowed in a cell
UPDOWN: array(UPMAX) of integer

! no. of machines i or fraction thereof that process part j
!in cell q
x: array(MACH,PARTS,CELLS) of mpvar

! 1 if kth machine of type i is assigned to cell q
y:array(MACH,KMAX,CELLS) of mpvar

! No. of machines of type i that will be used by part j
! in cell q
s: array(MACH,PARTS,CELLS) of mpvar

! 1 if cell q is formed
v: array(CELLS) of mpvar

! 1 if part j is processed in cell q, 0 otherwise
w:array(PARTS,CELLS) of mpvar

! 1 if after the zth operation of part j in cell q,
! part leaves cell q but returns later; 0 otherwise
extra: array(CELLS,PARTS,CELLS) of mpvar
```

```
! 1 if part j is processed in cell q (zth machine operation),
! 0 otherwise
xx: array(MACH,PARTS,CELLS) of mpvar

end-declarations

initializations from "CF.dat"
  UTIL SETUP L KTYPES ZTYPES UPDOWN UTILMAX UTILMIN
end-initializations

forall(i in MACH, k in KMAX, q in CELLS) y(i,k,q) is_binary

forall(j in PARTS, q in CELLS) w(j,q) is_binary

forall(q in CELLS) v(q) is_binary

forall(q in CELLS, j in PARTS, i in MACH) extra(q,j,i) is_binary

forall(i in MACH, j in PARTS, q in CELLS) xx(i,j,q) is_binary

forall(i in MACH, j in PARTS, q in CELLS) s(i,j,q) is_integer


COST := sum(j in PARTS, q in CELLS) 10*w(j,q) +
        sum(i in MACH, j in PARTS)(SETUP(i,j) * sum(q in CELLS) s(i,j,q)) +
        sum(q in CELLS, j in PARTS, i in MACH) extra(q,j,i)


! Constraint 1: kth machine of type i must be assigned to exactly !
!-------------  one cell
forall(i in MACH, k in KMAX | k<=KTYPES(i))
 sum(q in CELLS) y(i,k,q) = 1

! Constraint 2: total no. of machines required to process part j in
!-------------  cell q
forall(i in MACH, j in PARTS | UTIL(i,j)>0.0)
 sum(q in CELLS) x(i,j,q)-UTIL(i,j)=0.0

! Constraint 3:
!-------------
forall(i in MACH) do
 forall(j in PARTS) do
  forall(q in CELLS) do
   x(i,j,q)-s(i,j,q)<=0.0
  end-do
 end-do
end-do

! Constraint 4:
!-------------  if x is zero then s has to become a
zero forall(i in MACH) do
 forall(j in PARTS) do
  forall(q in CELLS) do
   x(i,j,q)-UTILMIN*s(i,j,q)>=0.0
  end-do
 end-do
```

```
end-do

!Constraint 5: total number of machines of type i assigned to cell
!------------  q is less than or equal to the k no. of machines of !
!type i assigned to cell q.
forall(i in MACH) do
 forall(q in CELLS) do
  sum(j in PARTS | UTIL(i,j)>0.0) x(i,j,q)-
       sum(k in KMAX|k<=KTYPES(i)) y(i,k,q)<=0.0
 end-do
end-do

! Constraint 6: the number of machines included in a cell, when cell
!------------   is formed should be at most emax machines
 forall(q in CELLS)
 sum(i in MACH, k in KMAX | k<=KTYPES(i)) y(i,k,q)<=v(q)*UPDOWN(1)

! Constraint 7: the number of machines included in a cell
!------------   when cell is formed should be at least emin machines
forall(q in CELLS)
 sum(i in MACH, k in KMAX | k<= KTYPES(i)) y(i,k,q)>=v(q)*UPDOWN(2)

! Constraint 8: cells are formed in successive numerical order!
!------------
forall(q in CELLS | q<5) v(q+1)<=v(q)

! Constraint 9: duplicate machines when needed are allocated !
!------------   to lowered numbered cells in successive numerical
order forall(i inMACH) do
 forall(k in KMAX | KTYPES(i)<>1 and k<KTYPES(i)) do
  sum(q in CELLS) q*y(i,k,q)-sum(q in CELLS) q*y(i,k+1,q)<= 0.0
 end-do
end-do

! Constraint 10:
! -------------
forall(i in MACH) do
 forall(j in PARTS) do
  forall(q in CELLS) do
   x(i,j,q)-UTIL(i,j)*w(j,q)<=0.0
  end-do
 end-do
end-do

! Constraint 11:
! -------------
! define xx(i,j,q)
! when the value of x is greater than 0, xx variable is equal to 1
forall(i in MACH) do
 forall(j in PARTS) do
  forall(q in CELLS) do
   x(i,j,q) - UTILMAX*xx(i,j,q) <= 0.0
  end-do
 end-do
end-do
```

```
! Constraint 12:
! -------------
! define xx(i,j,q)
! when the value of x is greater than 0, xx variable is equal to 1
forall(i in MACH) do
 forall(j in PARTS) do
  forall(q in CELLS) do
   x(i,j,q) - UTILMIN*xx(i,j,q) >= 0.0
  end-do
 end-do
end-do

! Constraint 13:
! -------------
forall(j in PARTS, q in CELLS) do
 forall(z in ZOPER|z<=ZTYPES(j) and ZTYPES(j)>=3 and L(j,z)>=0.0) do
  forall(r in ZOPER|r<=ZTYPES(j) and r>=z+2 and L(j,r)>=0.0) do
   xx(L(j,z),j,q) + xx(L(j,r),j,q)-
   sum(zz in z+1..r-1|zz<=ZTYPES(j) and L(j,zz)>0.0) xx(L(j,zz),j,q) -
   extra(q,j,L(j,z)) -1 <= 0
  end-do
 end-do
end-do

setparam("XPRS_VERBOSE", true)

minimize(COST)
fopen("OUT.dat", F_OUTPUT)

writeln("Cost = ", getobjval)
writeln('')
writeln('x(i,j,q)
values:')
forall(i in MACH) do
 forall(j in PARTS) do
  forall(q in CELLS) do
   if (getsol(x(i,j,q)) > 0.0) then
    writeln('x(', i,' ',j,' ',q,') ',getsol(x(i,j,q)))
   end-if
  end-do
 end-do
end-do
writeln('')
writeln('v(q) values: ')
forall(q in CELLS) do
 if (getsol(v(q)) > 0.0) then
  writeln('v(',q,') ',getsol(v(q)))
 end-if
end-do
writeln('')
writeln('s(i,j,q) values: ')
forall(i in MACH) do
 forall(j in PARTS) do
  forall(q in CELLS) do
```

```
    if (getsol(s(i,j,q)) >0.0) then
     writeln('s(', i,' ',j,' ',q,') ',getsol(s(i,j,q)))
    end-if
   end-do
  end-do
 end-do
 writeln('')
 writeln('xx(i,j,q) values: ')
 forall(i in MACH) do
  forall(j in PARTS) do
   forall(q in CELLS) do
    if (getsol(xx(i,j,q)) >0.0) then
     writeln('xx(',i,' ',j,' ',q,') ',getsol(xx(i,j,q)))
    end-if
   end-do
  end-do
 end-do
 writeln('')
 writeln('extra(q,j,i) values: ')
 forall(q in CELLS) do
  forall(j in PARTS) do
   forall(i in MACH) do
    if (getsol(extra(q,j,i)) > 0.0) then
     writeln('extra(', q,' ',j,' ',i,') ',getsol(extra(q,j,i)))
    end-if
   end-do
  end-do
 end-do
 writeln('')
 writeln('y(i,k,q) values: ')
 forall(i in MACH) do
  forall(k in KMAX) do
   forall(q in CELLS) do
    if (getsol(y(i,k,q))> 0.0) then
      writeln('y(',i,' ',k,' ',q,') ', getsol(y(i,k,q)))
    end-if
   end-do
  end-do
 end-do
 writeln('')
 writeln('w(j,q) values: ')
 forall(j in PARTS) do
  forall(q in CELLS) do
   if (getsol(w(j,q))> 0.0) then
    writeln('w(',j,' ',q,') ', getsol(w(j,q)))
   end-if
  end-do
 end-do

 fclose(F_OUTPUT)
 end-model
```

A.3.1 Complete Model: Solution

```
Cost = 219.92

x(i,j,q) values:
x(1 2 1) 0.3
x(1 4 1) 0.1
x(1 8 1) 0.1
x(2 4 1) 1
x(2 5 4) 0.5
x(2 7 2) 0.9
x(2 9 2) 0.1
x(2 10 3) 0.6
x(3 3 2) 0.9
x(3 8 1) 0.2
x(3 9 2) 0.1
x(4 2 1) 0.2
x(4 2 3) 1
x(4 4 1) 0.4
x(4 4 4) 0.5
x(4 5 4) 0.5
x(4 6 1) 0.4
x(5 1 3) 0.4
x(5 2 3) 0.2
x(5 4 4) 0.8
x(5 5 4) 1.1
x(5 6 1) 1
x(5 7 2) 0.6
x(5 8 1) 1
x(5 9 2) 1
x(6 2 3) 0.7
x(6 7 2) 0.8
x(6 10 3) 0.3
x(7 4 4) 0.2
x(7 5 4) 0.2

v(q) values:
v(1) 1
v(2) 1
v(3) 1
v(4) 1

s(i,j,q) values:
s(1 2 1) 1
s(1 4 1) 1
s(1 8 1) 1
s(2 4 1) 1
s(2 5 4) 1
s(2 7 2) 1
s(2 9 2) 1
s(2 10 3) 1
s(3 3 2) 1
s(3 8 1) 1
s(3 9 2) 1
s(4 2 1) 1
s(4 2 3) 1
s(4 4 1) 1
```

```
s(4 4 4) 1
s(4 5 4) 1
s(4 6 1) 1
s(5 1 3) 1
s(5 2 3) 1
s(5 4 4) 1
s(5 5 4) 2
s(5 6 1) 1
s(5 7 2) 1
s(5 8 1) 1
s(5 9 2) 1
s(6 2 3) 1
s(6 7 2) 1
s(6 10 3) 1
s(7 4 4) 1
s(7 5 4) 1

xx(i,j,q) values:
xx(1 2 1) 1
xx(1 4 1) 1
xx(1 8 1) 1
xx(2 4 1) 1
xx(2 5 4) 1
xx(2 7 2) 1
xx(2 9 2) 1
xx(2 10 3) 1
xx(3 3 2) 1
xx(3 8 1) 1
xx(3 9 2) 1
xx(4 2 1) 1
xx(4 2 3) 1
xx(4 4 1) 1
xx(4 4 4) 1
xx(4 5 4) 1
xx(4 6 1) 1
xx(5 1 3) 1
xx(5 2 3) 1
xx(5 4 4) 1
xx(5 5 4) 1
xx(5 6 1) 1
xx(5 7 2) 1
xx(5 8 1) 1
xx(5 9 2) 1
xx(6 2 3) 1
xx(6 7 2) 1
xx(6 10 3) 1
xx(7 4 4) 1
xx(7 5 4) 1

extra(q,j,i) values:

y(i,k,q) values:
y(1 1 1) 1
y(2 1 1) 1
y(2 2 2) 1
y(2 3 3) 1
y(2 4 4) 1
y(3 1 1) 1
y(3 2 2) 1
y(4 1 1) 1
y(4 2 3) 1
y(4 3 4) 1
y(5 1 1) 1
y(5 2 1) 1
y(5 3 2) 1
y(5 4 2) 1
y(5 5 3) 1
y(5 6 4) 1
y(5 7 4) 1
y(6 1 2) 1
y(6 2 3) 1
y(7 1 4) 1

w(j,q) values:
w(1 3) 1
w(2 1) 1
w(2 3) 1
w(3 2) 1
w(4 1) 1
w(4 4) 1
w(5 4) 1
w(6 1) 1
w(7 2) 1
w(8 1) 1
w(9 2) 1
w(10 3) 1
```

A.3.2 Illustration of Later Revisits of Parts

```
Cost = 220.79

x(i,j,q) values:
x(1 2 1) 0.3
x(1 4 1) 0.1
x(1 8 1) 0.1
x(2 4 1) 1
x(2 5 3) 0.5
x(2 7 3) 0.9
x(2 9 2) 0.1
x(2 10 2) 0.6
x(3 3 2) 0.9
x(3 8 1) 0.2
x(3 9 2) 0.1
x(4 2 1) 0.6
x(4 2 2) 0.6
x(4 4 1) 0.9
x(4 5 1) 0.5
x(4 6 2) 0.4
x(5 1 3) 0.4
```

```
x(5 2 1) 0.2
x(5 4 1) 0.8
x(5 5 3) 1.1
x(5 6 2) 1
x(5 7 3) 0.6
x(5 8 1) 1
x(5 9 2) 1
x(6 2 2) 0.7
x(6 7 3) 0.8
x(6 10 2) 0.3
x(7 2 1) 0.5
x(7 4 1) 0.2
x(7 5 1) 0.2

v(q) values:
v(1) 1
v(2) 1
v(3) 1

s(i,j,q) values:
s(1 2 1) 1
s(1 4 1) 1
s(1 8 1) 1
s(2 4 1) 1
s(2 5 3) 1
s(2 7 3) 1
s(2 9 2) 1
s(2 10 2) 1
s(3 3 2) 1
s(3 8 1) 1
s(3 9 2) 1
s(4 2 1) 1
s(4 2 2) 1
s(4 4 1) 1
s(4 5 1) 1
s(4 6 2) 1
s(5 1 3) 1
s(5 2 1) 1
s(5 4 1) 1
s(5 5 3) 2
s(5 6 2) 1
s(5 7 3) 1
s(5 8 1) 1
s(5 9 2) 1
s(6 2 2) 1
s(6 7 3) 1
s(6 10 2) 1
s(7 2 1) 1
s(7 4 1) 1
s(7 5 1) 1

xx(i,j,q) values:
xx(1 2 1) 1
xx(1 4 1) 1
xx(1 8 1) 1
xx(2 4 1) 1
xx(2 5 3) 1
xx(2 7 3) 1
xx(2 9 2) 1
xx(2 10 2) 1
xx(3 3 2) 1
xx(3 8 1) 1
xx(3 9 2) 1
xx(4 2 1) 1
xx(4 2 2) 1
xx(4 4 1) 1
xx(4 5 1) 1
xx(4 6 2) 1
xx(5 1 3) 1
xx(5 2 1) 1
xx(5 4 1) 1
xx(5 5 3) 1
xx(5 6 2) 1
xx(5 7 3) 1
xx(5 8 1) 1
xx(5 9 2) 1
xx(6 2 2) 1
xx(6 7 3) 1
xx(6 10 2) 1
xx(7 2 1) 1
xx(7 4 1) 1
xx(7 5 1) 1

extra(q,j,i) values:
extra(2 2 4) 1

y(i,k,q) values:
y(1 1 1) 1
y(2 1 1) 1
y(2 2 2) 1
y(2 3 3) 1
y(2 4 3) 1
y(3 1 1) 1
y(3 2 2) 1
y(4 1 1) 1
y(4 2 1) 1
y(4 3 2) 1
y(5 1 1) 1
y(5 2 1) 1
y(5 3 2) 1
y(5 4 2) 1
y(5 5 3) 1
y(5 6 3) 1
y(5 7 3) 1
y(6 1 2) 1
y(6 2 3) 1
y(7 1 1) 1

w(j,q) values:
w(1 3) 1
w(2 1) 1
```

```
w(2 2) 1
w(3 2) 1
w(4 1) 1
w(5 1) 1
w(5 3) 1
w(6 2) 1
w(7 3) 1
w(8 1) 1
w(9 2) 1
w(10 2) 1
```

Appendix B

Generic Forms of Selected Fuzzy Aggregation Operators

The following notation is used in the transformation of fuzzy models. The notation has been adapted from Zimmermann [Zim91].

$\mu_G(x)$: the membership function of the objective function. For the linear non-increasing membership function used in Chapter 4 the value for objective function is: $\mu_G(x) = 1 - (c^T x - D^0)/P_0$

$\mu_{C_i}(x)$: the membership function of the constraints. Two function types are considered in Chapter 4:

(a) Linear non-increasing. The membership function is
$\mu_{C_i}(x) = 1-(Ax-b)/P_{R1}$ where P_{R1} is the tolerance value for this constraint.

(b) Triangular. The following two formulae are used:
$\mu_{C_i}(x) = 1 - (Ax - b)/P_{R1}$ for the upper bound, and
$\mu_{C_i}(x) = 1 - (b - Ax)/P_{R2}$ for the lower bound,
where P_{R2} is the tolerance value for the lower bound and P_{R1} is the tolerance value for the upper bound.

B.1 'Min' Operator

Zadeh [Zad65] proposed 'min' operator to define the intersection of the following aggregated rule:

$$\mu_D(x) = \mu_G(x) \wedge \mu_{C_i}(x) = \mu_G(x) \cap \mu_{C_i}(x) \tag{B.1}$$

The aggregated membership function becomes

$$\mu_D(x) = min[\mu_G(x), \mu_{C_l}(x)] \ \ \forall \ \ l \tag{B.2}$$

To find the maximum value of $\mu_D(x)$, the model can be defined as:

$$max \ \ \lambda \tag{B.3}$$

subject to

$$\lambda \leq \mu_G(x) \tag{B.4}$$

$$\lambda \leq \mu_{C_l}(x) \ \ \forall \ \ l \tag{B.5}$$

$$0 \leq \lambda \leq 1 \tag{B.6}$$

B.2 'Min-bounded-sum' Operator

Luhandjula [Luh82] proposed the following aggregation rule:

$$\mu_D = \gamma \times \mu_{G \cap C_l} + (1 - \gamma) \times \mu_{G \cup C_l} \tag{B.7}$$

If the intersection and the union are represented by the 'min' and 'bounded-sum' operators the aggregated membership function becomes:

$$\mu_D = \gamma \times min_{S=0}^{T} \ \mu_S + (1 - \gamma) \times min(1, \sum_{S=0}^{T} \mu_S) \tag{B.8}$$

The equivalent form after using the operator is:

$$max \ \ \gamma\omega + (1 - \gamma)u \tag{B.9}$$

subject to

$$\omega \leq \mu_G(x) \tag{B.10}$$

$$\omega \leq \mu_{C_l}(x) \ \ \forall \ \ l \tag{B.11}$$

$$u \leq 1 \tag{B.12}$$

$$u \leq \mu_G(x) + \sum_l \mu_{C_l}(x) \tag{B.13}$$

$$0 \leq \omega \leq 1 \tag{B.14}$$

$$\gamma \leq 1 \tag{B.15}$$

B.3 'Fuzzy and' Operator

Werners [Wer88] suggested modification of aggregation rule (B.7) proposed by Luhandjula. The membership function of the resulting fuzzy set has the following form:

$$\mu_D = \gamma min_{S=0}^{T} \mu_S + (1-\gamma)/(T+1) \times \sum_{S=0}^{T} \mu_S \qquad \text{(B.16)}$$

The equivalent model after this operation is performed becomes:

$$max \;\; \omega + (1-\gamma)/(t+1) \times \sum_{s=0}^{t} a_s \qquad \text{(B.17)}$$

subject to

$$\omega + a_0 \leq \mu_G(x) \qquad \text{(B.18)}$$

$$\omega + a_l \leq \mu_{C_l}(x) \;\; \forall \;\; l \qquad \text{(B.19)}$$

$$\omega + a_s \leq 1 \;\; \forall \;\; s \qquad \text{(B.20)}$$

$$\omega, a_s \geq 0 \qquad \text{(B.21)}$$

$$\gamma \leq 1 \qquad \text{(B.22)}$$

Appendix C

Random Data Generation

```
function [UTIL,SETUP,L,KTYPES,TM,ZTYPES] =
                             RandData(NM,NP,ZOPER,UTILRANGE,SETUPRANGE)
% function [UTIL,SETUP,L,KTYPES,TMI,ZTYPES] =
                  RandData(NM,NP,ZOPER,UTILRANGE,SETUPRANGE)
% RandData: Generate pseudo-random utilisation matrix (for cell formation),
% set-up cost matrix, array of part machine sequence operations
%
% output arguments:
% *****************
% UTIL:    generated NM x NP utilisation matrix
% SETUP:   set-up cost for part machine assignment
% KTYPES: number of machines used by each part (i.e. sum of each row of UTIL)
% ZTYPES: number of machine operations required for each part
% L:       part machine sequence
% TMI:     total number of machine instances
%
% input arguments:
% ****************
% NM:         number of machine types
% NP:         number of parts/components
% ZOPER:      number of part operations (relative to columns)
% UTILRANGE: utilisation range [r(1) r(2)], r(1) min, r(2) max
% SEUPRANGE: set-up cost range [r(1) r(2)], r(1) min, r(2) max
%
% Note: based on Uniform Pseudo-Random Number Generation.

% initial random generation, no ZOPER considered yet
init_gen = round((UTILRANGE(1) +
(UTILRANGE(2)-UTILRANGE(1)).*rand(NM,NP))*10)/10;

% The entries in init_gen (column-wise) should not exceed ZOPER, thus:
% (a) set up the vector of part operations from 1 up to y
err1 = [1:1:ZOPER]; % [1:p:y]: from 1 up to y with step p
% (b) set up the probabilities relative to each element of vector err1
% This is used for randerr, i.e. random binary matrix [0 1] generation

% first create a random row vector with integer numbers in a range,
% this is flexible: use any, in this case range [0 99] used;
```

```
err2a = randint(1,length(err1),[0 99]);
% sum(err2a) is different than 1 usually. Make sum(err2) equal to 1, as
% it will used in randerr i.e. the elements of the  probability vector err2
% must add up to 1.
err2 = err2a/sum(err2a);

out = randerr(NP,NM,[err1;err2])'; % generate NM by NP random binary matrix
% note that in order to avoid having all zeros in any of the columns we
% must first generate the mxn matrix out and then take the transpose. This
% is because randerr works with rows and not columns.

% generate the finalised random matrix UTIL with the utilisation included.
% perform an element by element multiplication
UTIL = init_gen.*out;

% find KTYPES
KTYPES = ceil(sum(UTIL'));

TM=sum(KTYPES);

% find ZTYPES
ZTYPES = sum(UTIL>0);

SETUP=zeros(NM,NP);
for i=1:NM
 for j=1:NP
     if UTIL(i,j)>0
      SETUP(i,j)=round((SETUPRANGE(1) +
                     (SETUPRANGE(2)-SETUPRANGE(1))*rand(1))*100)/100;
     end
 end
end

L=zeros(NP,max(ZTYPES));
for yy=1:NP % work along the rows of L
    % find indices of nonzero elements in UTIL column
    [u]=find(UTIL(:,yy)>0.0);
    % rr1 and rr2 is an attempt of slightly perturbing the sequence of L
    % random generation of shift register for circshift 1
    rr1 = (-1)^(ceil(length(u).*rand(1,1)))*ceil(length(u).*rand(1,1));
    % random generation of shift register for circshift 2
    rr2 = (-1)^(floor(length(u).*rand(1,1)))*floor(length(u).*rand(1,1));
    L(yy,1:length(u))=circshift(u',[rr1 rr2]);
end
```

Appendix D

Part Allocation - `MatLab` Code

```
function [PARTMATRIX,PCMATRIX,part_moves] =
PartAllocation(UTIL_sortedi,CELLMATRIX2,NP,NC,EMAX,NM,KMAX,L,UTIL)

%function [PARTMATRIX,PCMATRIX,part_moves] = PartAllocation(UTIL_sortedi,CELLMATRIX2,NP,NC,EMAX,NM,KMAX,L,UTIL)
% Allocate parts to machine cells
% Inputs:  UTIL_sortedi, CELLMATRIX2, NP, NC, EMAX, NM, KMAX, L, UTIL
% Outputs: PARTMATRIX (i,k,j) part j uses kth machine of type i
%          PCMATRIX (i,k,j,q) part j uses kth machine of type i in cell q
%          part_moves (j,q,i,k)

% initialise 3-D matrix (i,k,j) machine no., machine instance, part.
PARTMATRIX = zeros(NM,KMAX,NP);

% Find which machines appear at each cell using CELLMATRIX2
MACHINENUMBERq = zeros(NC,EMAX); % initialise matrix for storing machine types per cell
temp_matrix2 = []; % initialise temp matrix for sorting out
for q=1:NC
    temp_matrix2 = CELLMATRIX2(:,:,q);
    % find row-col indices of the current 1-to-1 relation of machines
    % and machine instances for each cell q
    [MACHi,INSTANCEi] = find(temp_matrix2);
    MACHINENUMBERq(q,1:length(MACHi)) = MACHi';
end MACHINENUMBERq_o=MACHINENUMBERq;

pm_sequence = []; % initialise temp vector for part machine sequence manipulation
UTIL_temp=UTIL; % used to account for possible splits
PCMATRIX = zeros(NM,KMAX,NP,NC); % initialise 4D matrix (machine no., instance no., part no., cell no.)
part_moves=[]; % initialise vector for keeping part no, cell no, machine no., instance no.
if all(UTIL_temp==0) % check
    warning('Before allocation: The UTIL matrix is a ZERO matrix!')
end

for j = [UTIL_sortedi(1:1:length(UTIL_sortedi))] % LOOP J
    %disp('LOOP J')
    % remove zero entities from the current part machine sequence vector, see L
    pm_sequence = rvze(L(j,:)); % for current part
    % ------------------------------------------------------------------
    % APPROACH 1: Sequence of cells based on majority of machine in cells
    %             relative to the part machine sequence
    % ------------------------------------------------------------------    %
    % % total part machine sequence
    %   machines_in_cell = []; % initialise temp vector for keeping machines in cells
    %   cells_and_machineno = []; % initialise temp vector for keeping cell and its number of machines relative
    %                              % to pm_sequence (machine sequence for the current part).
    % % check for each cell how many machines are the same with those held in pm_sequence and sum them up
    %   for q=1:NC % for each cell
    %       machines_in_cell = rvme(rvze(MACHINENUMBERq(q,:))); % remove zeros & multiples of a machine in each row
    %       machines_sum = 0; % initialise sum of machines for each cell in the next loop
    %       for w = 1:length(machines_in_cell)
    %           machines_sum = machines_sum + sum(machines_in_cell(w) == pm_sequence);
%           % compare each entry of the machines_in_cells with the part
%           % machine operation sequence(machine_operation_sequences)
%       end
%       cells_and_machineno = [cells_and_machineno ; q machines_sum];
%   end
%   [machines_per_cell, machines_per_celli] = sort(cells_and_machineno(:,2),'descend');
%   % machines_per_cell returns the sum of the machines relative to the machine
%   % operation sequence, with respect to cell numbers.
%   % machine_per_celli returns the corresponding indices.
%   cells_machines_sorted = [cells_and_machineno(machines_per_celli,1) machines_per_cell];
%   temp_cell_seq = cells_machines_sorted(:,1)'; % make it a row vector
%
% ---------------------------------------------------------------------
% APPROACH 2: cell sequence based on maximum continuous sequence of machines relative to the
%             part machine sequence
%             Approach 2 was chosen instead of 1
% ---------------------------------------------------------------------
% sort cells preserving the sequence of machines relative to part machine sequence
jj=1; % pointer relative to part machine sequence, changes based upon the latest zero elements in ac
qq=[1:NC];
qq_used=[];
temp_cell_seq = []; % initialise temp vector for classifying cells
while jj<=length(pm_sequence)
    i_set1=[]; % initialize index for keeping the non-zero indices column wise in ac_all
    % find ac_all relative to part's machine sequence
    [ac_all] = ContMacSeq(MACHINENUMBERq,pm_sequence,NC);
    i_set1 = find(ac_all(:,jj)==(pm_sequence(jj)*ones(length(ac_all(:,jj)),1)));
    %keyboard
    % if i_set1 is empty break while loop
    if isempty(i_set1)
        break
    end
    % if jj has reached the last element of the part machine sequence
    if jj==length(pm_sequence)
        %keyboard
        % store in temp_cell_seq the cell where the first index of i_set1, last column
        % of ac_all_temp, is located e.g. cell 2
        temp_cell_seq = [temp_cell_seq;qq(i_set1(1))]; % fill cells
        % and add this cell in the vector of cells used
        qq_used=[qq_used;qq(i_set1(1))];
        break % give control to the loop that follows
    else
        %keyboard
        % find zi_all1 from ac_all (zi_all1 matrix partially defined depending upon the jj value)
        zi_all1=RetZeroInd(ac_all(i_set1,jj:length(ac_all(1,:))),pm_sequence);
        % yz1_temp indices relative to the partial matrix
        [yz1_temp,index1_temp] = sort(zi_all1(:,1),'descend');
        % yz1_temp indices relative to the original matrix;
        % when jj=1, yz1_temp value is the same either locally and globally
        yz1_temp=yz1_temp+(length(ac_all(1,:))-length(ac_all(1,jj:length(ac_all(1,:)))));
        % keyboard
        % fill cells
        temp_cell_seq = [temp_cell_seq;qq(i_set1(index1_temp(1)))];
        qq_used=[qq_used;qq(i_set1(index1_temp(1)))];
        % jj is assigned the value of the yz1_temp(1) with respect to the original matrix
        % global update
        jj=yz1_temp(1);
    end
    %keyboard
end
% add all remaining cells not involved in the arranged sequence
for i=1:length(qq)
    if  isempty(find(temp_cell_seq==qq(i)))
        temp_cell_seq=[temp_cell_seq;qq(i)];
    end
end
temp_cell_seq=temp_cell_seq';
%keyboard
for d=1:length(pm_sequence) % LOOP D
    %disp('LOOP D')
    q2=1; % initialise counter for navigating through cells
    while q2 <= length(temp_cell_seq) % less or equal to the current sorted cells
        %disp('LOOP Q2')
        %keyboard
        pm_sequence(d) == rvme(rvze(MACHINENUMBERq(temp_cell_seq(q2),:)));
        if any(pm_sequence(d) == rvme(rvze(MACHINENUMBERq(temp_cell_seq(q2),:))))==1 % IF Q2(1)
            %disp('IF Q2(1)')
            % any returns a logical true (1) when the expression - vector evaluates to at least one non-zero entry
            % for the current part, for the current cell, for the current machine find its machine instances
            mach_instances = []; % initialize vector for keeping machine instances of a specific
            % machine type within one cell
            mach_instances = CELLMATRIX2(pm_sequence(d),:,temp_cell_seq(q2));
            [instance_no, instance_noi] = find(mach_instances==1);
            % for the current machine instance of type pm_sequence(d) perform a number of checks
            k2=1; % initialise counter for navigating through machine instances
            while k2<=length(instance_noi) % LOOP k2
                %disp('LOOP K2')
                %keyboard
                flag1=0;
                flag2=0;
                if sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))<1 & UTIL_temp(pm_sequence(d),j)>=1
                    %keyboard
                    %disp('IF k2(1)')
                    UTIL_1 = 1-sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:));
                    PARTMATRIX(pm_sequence(d),instance_noi(k2),j)=UTIL_1;
                    UTIL_temp(pm_sequence(d),j)=UTIL_temp(pm_sequence(d),j)-UTIL_1;
                    part_moves=[part_moves; j,temp_cell_seq(q2),pm_sequence(d),instance_noi(k2)];
                    % assign 1 to PCMATRIX when part j, uses machine instance k2 of type pm_sequence(d)
                    % in cell temp_cell_seq(q2)
                    PCMATRIX(pm_sequence(d),instance_noi(k2),j,temp_cell_seq(q2))=1;
                    flag1=1;
    if UTIL_temp(pm_sequence(d),j)==0.0
        break % exit this while loop and move onto the outer loop
    end
elseif flag1==0 & sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))<1 & UTIL_temp(pm_sequence(d),j)<1
    %disp('IF k2(2)')
    %keyboard
    if  sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))<=UTIL_temp(pm_sequence(d),j)
        %disp('IF k2(2.1)')
        if (sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))+UTIL_temp(pm_sequence(d),j))==1
            %disp('IF k2(2.1.1)')
            PARTMATRIX(pm_sequence(d),instance_noi(k2),j)=UTIL_temp(pm_sequence(d),j);
            UTIL_temp(pm_sequence(d),j)=0;
            part_moves=[part_moves; j,temp_cell_seq(q2),pm_sequence(d),instance_noi(k2)];
            % assign 1 to PCMATRIX when part j, uses machine instance k2 of type pm_sequence(d)
            % in cell temp_cell_seq(q2)
            PCMATRIX(pm_sequence(d),instance_noi(k2),j,temp_cell_seq(q2))=1;
        elseif (sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))+UTIL_temp(pm_sequence(d),j))>1
            %disp('IF k2(2.1.2)')
            %keyboard
            UTIL_1=1-sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:));
            PARTMATRIX(pm_sequence(d),instance_noi(k2),j)=UTIL_1;
            UTIL_temp(pm_sequence(d),j)=UTIL_temp(pm_sequence(d),j)-UTIL_1;
            part_moves=[part_moves; j,temp_cell_seq(q2),pm_sequence(d),instance_noi(k2)];
            % assign 1 to PCMATRIX when part j, uses machine instance k2 of type pm_sequence(d)
            % in cell temp_cell_seq(q2)
            PCMATRIX(pm_sequence(d),instance_noi(k2),j,temp_cell_seq(q2))=1;
            if UTIL_temp(pm_sequence(d),j)==0.0
                break % exit this while loop and move onto the outer loop
            end
        else
            %keyboard
            %disp('IF k2(2.1.3)')
            PARTMATRIX(pm_sequence(d),instance_noi(k2),j)=UTIL_temp(pm_sequence(d),j);
            UTIL_temp(pm_sequence(d),j)=0;
            part_moves=[part_moves; j,temp_cell_seq(q2),pm_sequence(d),instance_noi(k2)];
            % assign 1 to PCMATRIX when part j, uses machine instance k2 of type pm_sequence(d)
            % in cell temp_cell_seq(q2)
            PCMATRIX(pm_sequence(d),instance_noi(k2),j,temp_cell_seq(q2))=1;
            break
        end
        flag2=1;
    end
end
%keyboard
if all([flag1 flag2]==0) & sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))<1 &
   sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))>=UTIL_temp(pm_sequence(d),j);
     %disp('IF k2(3)')
     %keyboard
    if (sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))+UTIL_temp(pm_sequence(d),j))==1
        %disp('IF k2(3.1)')
```

```
                            %keyboard
                            PARTMATRIX(pm_sequence(d),instance_noi(k2),j)=UTIL_temp(pm_sequence(d),j);
                            UTIL_temp(pm_sequence(d),j)=0;
                            part_moves=[part_moves; j,temp_cell_seq(q2),pm_sequence(d),instance_noi(k2)];
                            % assign 1 to PCMATRIX when part j, uses machine instance k2 of type pm_sequence(d)
                            % in cell temp_cell_seq(q2)
                            PCMATRIX(pm_sequence(d),instance_noi(k2),j,temp_cell_seq(q2))=1;
                        elseif (sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:))+UTIL_temp(pm_sequence(d),j))>1
                            %disp('IF k2(3.2)')
                            %keyboard
                            UTIL_1=1-sum(PARTMATRIX(pm_sequence(d),instance_noi(k2),:));
                            PARTMATRIX(pm_sequence(d),instance_noi(k2),j)=UTIL_1;
                            UTIL_temp(pm_sequence(d),j)=UTIL_temp(pm_sequence(d),j)-UTIL_1;
                            part_moves=[part_moves; j,temp_cell_seq(q2),pm_sequence(d),instance_noi(k2)];
                            % assign 1 to PCMATRIX when part j, uses machine instance k2 of type pm_sequence(d)
                            % in cell temp_cell_seq(q2)
                            PCMATRIX(pm_sequence(d),instance_noi(k2),j,temp_cell_seq(q2))=1;
                            if UTIL_temp(pm_sequence(d),j)==0.0
                                break % exit this while loop and move onto the outer loop
                            end
                        else
                            %keyboard
                            %disp('IF k2(3.3)')
                            PARTMATRIX(pm_sequence(d),instance_noi(k2),j)=UTIL_temp(pm_sequence(d),j);
                            UTIL_temp(pm_sequence(d),j)=0;
                            part_moves=[part_moves; j,temp_cell_seq(q2),pm_sequence(d),instance_noi(k2)];
                            % assign 1 to PCMATRIX when part j, uses machine instance k2 of type pm_sequence(d)
                            % in cell temp_cell_seq(q2)
                            PCMATRIX(pm_sequence(d),instance_noi(k2),j,temp_cell_seq(q2))=1;
                            break
                        end
                    end
                    k2=k2+1;
                end
            end
            if UTIL_temp(pm_sequence(d),j)==0.0
                %keyboard
                break % exit this while loop and move onto the outer loop
            end
            q2=q2+1; % accumulate cell counter
        end
    end
end
```

Appendix E

Tabu Search Algorithm - MatLab Code

```
function[BEST_COST,BEST_SOL,BEST_CELLMATRIX2,BEST_part_moves,BEST_W_jq,BEST_S_ij,BEST_PCMATRIX,BEST_PARTMATRIX,i_objvalprofile,...
              ObjValProfile,BestLocalVector]=...
         TabuSearchv4tt(L,NP,NC,KMAX,NM,EMIN,EMAX,PARTMATRIX,UTIL,UTIL_sortedi,SETUP,INIT_OBJVAL,INIT_SOL,CELLMATRIX2,PCMATRIX,...
                        W_jq,S_ijq,S_ij,extra_moves,part_moves,tti)

% function [BEST_COST,BEST_SOL,BEST_CELLMATRIX2,BEST_part_moves,BEST_W_jq,BEST_S_ij,BEST_PCMATRIX,BEST_PARTMATRIX,i_objvalprofile,...
%              ObjValProfile,BestLocalVector] =...
%         TabuSearchv4tt(L,NP,NC,KMAX,NM,EMIN,EMAX,PARTMATRIX,UTIL,UTIL_sortedi,SETUP,INIT_OBJVAL,INIT_SOL,CELLMATRIX2,PCMATRIX,...
%                        W_jq,S_ijq,S_ij,extra_moves,part_moves,tti)
%
% Tabu Search Iterative Procedure
%
% Inputs:  L part machine operation
% -------  NP number of parts
%          NC number of cells to be created
%          KMAX maximum part machine operation sequence
%          NM number of machines
%          EMIN minimum number of machines
%          EMAX maximum number of machines in a cell
%          PARTMATRIX (i,k,j) allocation of part j using kth instance of machine of type i in cell q
%          UTIL_sortedi descending order of parts of total processing requirements
%          SETUP set-up cost for part j using machine of type i
%          INIT_OBJVAL initial objective value
%          INIT_SOL initial solution
%          CELLMATRIX2 initial allocation of machines to cells
%          PCMATRIX (i,k,j,q) initial part machine cell allocation
%          W_jq distinct allocation of parts to cells
%          S_ijq integer number of machines of type i used by part j in cell q
%          S_ij number of machines of type i for part j
%          extra_moves number of later revisits of parts to already visited cell
%          part_moves (j q i k) relative to part machine sequence
%          tti tabu tenure
% Outputs: BEST_COST global best cost
% -------- BEST_SOL global best solution
%          BEST_CELLMATRIX2 global best CELLMATRIX2
%          BEST_part_moves the overall best (j,q,k,i) system configuration
%          BEST_W_jq best distinct allocation of parts to cells
%          BEST_S_ij best integer number of machines used by parts
%          BEST_PCMATRIX (i,k,j,q)
%          BEST_PARTMATRIX best allocation of parts to machine cells relative to part machine sequence
%          i_objvalprofile number of iterations involved
%          ObjValProfile value of the objective function at each obtained
%          BestLocalVector number of iterations involved together with objective function values of implemented moves only

% initialise best cost function equal to initial solution
BEST_COST=INIT_OBJVAL;
% the same for best solution (for the record)
BEST_SOL=INIT_SOL;
% initialise CELLMATRIX2_temp matrix to navigate through moves
CELLMATRIX2_temp=CELLMATRIX2;
```

```
% assume BEST_CELLMATRIX2 is initially the input CELLMATRIX2_temp
BEST_CELLMATRIX2=CELLMATRIX2_temp;
% initialise part_moves_temp (j q i k)
part_moves_temp = part_moves;
BEST_part_moves=part_moves_temp;
% initialise W_jq_temp
W_jq_temp = W_jq;
BEST_W_jq=W_jq_temp;
% initialise vector to keep the no. of intercellular moves
INTERCELLMOVS_temp=[];
SOL_temp={};
OBJVAL_temp = [];
ObjValProfile=INIT_OBJVAL;
% initialise matrix for keeping cell q, machine i and instance k. Cell q is the cell
% with the majority of machines of part's j sequence
qik_receiver=[];
tl=[]; % initialise tabu list
tt=tti; % initialise tabu tenure value
% insert a new iteration in order to allow parts to be reconsidered within the tabu
i_global=0; i_objval=0; i_objvalprofile = [0]; warning_size1 = 0;
BestLocalVector=[];
while i_global<2  % considers the parts sequence
    %keyboard
    % sort parts according in descending order of their intercellular movements caused
    % wi changes as W_jq_temp keeps changing throughout the iterations
    [wy,wi] = sort(sum(W_jq_temp')-1,'descend');
    wi; % wi gives the index of the sorted elements
    counter_part=0;
    i_part=0;
    while i_part < 1*length(wy)    % chooses the part;
        if counter_part>=length(wy)
           counter_part=0;
        end
        j2 = counter_part+1;
        i_source=0;
        counter_source=0; %for donor
        counter_objval_=0;
        while i_source<2*length(wy) % based upon the current part choose source cell
            if counter_objval_>=7 % length(qik(:,1))
                CELLMATRIX2_temp = BEST_CELLMATRIX2;
                [OBJVAL_temp,part_moves_temp,W_jq_temp,SOL_temp] = CalcObjValPA(UTIL_sortedi,CELLMATRIX2_temp,NP,NC,EMAX,NM,KMAX,L,UTIL,SETUP);
                break % if it does not improve, continue with next part
            end
            i_source;
            II=find(wi(j2)==part_moves_temp(:,1));
            qik = part_moves_temp(II,2:4);
            % counter for choosing donor index
            if counter_source>=length(qik(:,1))
                counter_source=0;
            end
            c1 = counter_source+1; % choose the index of the row of the current qik (defines the current donor)
            source_cell=qik(c1,1); % source_cell is the actual cell based on c1 index
```

```
all_cells = [1:NC];
[cc]=find(qik(c1,1)==all_cells); % find where the current qik(c1,1) exists in vector all_cells
all_cells(cc)=[]; % remove donor
sxz1=size(CELLMATRIX2);
CELLMATRIX2_4D = zeros(sxz1(1),sxz1(2),sxz1(3),length(all_cells)); % initialise matrix storage for the different attempts
OBJVALM = zeros(1,length(all_cells)); % init objval vector storage
tl_tempx = []; % initialise a vector for keeping temporary keeping any forward or reverse moves of candidate transitions
CELLMATRIX2_temp2 = CELLMATRIX2_temp;
for i_dest=1:length(all_cells) % find destination cells i.e. neighboring solutions for current machine instance pair
    % disp('Just entered i_dest')
    CELLMATRIX2_temp = CELLMATRIX2_temp2;
    dest_cell=all_cells(i_dest); % current receiver cell
    c2receiver=dest_cell;
    %keyboard
    receiver_sum=sum(sum(CELLMATRIX2_temp(:,:,dest_cell)));
    donor_sum =sum(sum(CELLMATRIX2_temp(:,:,qik(c1,1))));
    temp_sum=sum(sum(sum(CELLMATRIX2_temp)));
    init1_sum=sum(sum(sum(CELLMATRIX2)));
    %keyboard
    if receiver_sum >EMAX | receiver_sum < EMIN | donor_sum < EMIN | donor_sum > EMAX | temp_sum~=init1_sum
        warning_size1 = warning_size1+1
        error('problem encountered!')
    end
    % SINGLE MOVE ******************************************
    if sum(sum(CELLMATRIX2_temp(:,:,dest_cell)))<EMAX & sum(sum(CELLMATRIX2_temp(:,:,qik(c1,1))))<=EMAX &...
           sum(sum(CELLMATRIX2_temp(:,:,qik(c1,1))))>EMIN
        %disp('just entered single move')
        %keyboard
        % if target cell capacity < EMAX and donor cell capacity > EMIN do single move,
        % donate machine instance to target cell
        % target dest_cell cell receives a pair(destination cell)
        CELLMATRIX2_temp(qik(c1,2),qik(c1,3),dest_cell)=1;
        % remove machine instance pair from donor cell
        % qik(source cell)
        CELLMATRIX2_temp(qik(c1,2),qik(c1,3),qik(c1,1))=0;
        % locally store the possible attempt of donor to
        % receiver using the receiver index in the 4th-dimension
        CELLMATRIX2_4D(:,:,:,i_dest)= CELLMATRIX2_temp;
        %temporarily find obj val
        [OBJVAL_tempx,part_moves_tempx,W_jq_tempx,SOL_tempx] = ...
              CalcObjValPA(UTIL_sortedi,CELLMATRIX2_4D(:,:,:,i_dest),NP,NC,EMAX,NM,KMAX,L,UTIL,SETUP);
        OBJVALM(i_dest)= OBJVAL_tempx; % store current objval
        ObjValProfile = [ObjValProfile; OBJVALM(i_dest)];
        i_objval=i_objval+1;
        i_objvalprofile = [i_objvalprofile;i_objval];
        tl_tempx = [tl_tempx;[i_dest qik(c1,:) tt;i_dest dest_cell qik(c1,2) qik(c1,3) tt;]];
        %disp('before leaving single move')
        %keyboard
        % INTERCHANGE ******************************************
    else %if sum(sum(CELLMATRIX2_temp(:,:,dest_cell)))==EMAX & sum(sum(sum(CELLMATRIX2_temp)))==sum(sum(sum(CELLMATRIX2)))
        %disp('just entered interchange')
```

```
%keyboard
% receiver cell receives pair (i,k) needed and donates a pair not required by current
% part back to donor. Receiver cell receives a pair-i,k (belonging to its machine sequence)
% from donor cell
% find all machine-instance pairs in receiver cell including those utilised by current part
[mach_temp, inst_temp]=find(CELLMATRIX2_temp(:,:,dest_cell));
machine_pairs=[mach_temp inst_temp];
machine_pairs_temp=[mach_temp inst_temp];
if isempty(machine_pairs_temp)
    continue
end
% take each segment of current part from part_moves_temp
ii=find(wi(j2)==part_moves_temp(:,1));
part_moves_temp2=part_moves_temp(ii,:);
iiq=find(dest_cell==part_moves_temp2(:,2));
% take the 2:4 columns of current part moves for the current indices
qik_receiver = part_moves_temp2(iiq,2:4);
% exclude pairs (i,k) used by receiver cell (current part)-store them in machine_pairs_temp
% receiver will sent back to donor a machine pair not used by current part
for zz3=1:length(qik_receiver(:,1))
    zz4=1;
    while zz4<=length(machine_pairs_temp(:,1))
        if all(qik_receiver(zz3,2:3)==machine_pairs_temp(zz4,:))
            machine_pairs_temp(zz4,:)=[];
        end
        zz4=zz4+1;
    end
end
if isempty(machine_pairs_temp)
    continue
end
machine_pairs_donate=machine_pairs_temp;
% assign pair i,k from donor cell to receiver
%keyboard
CELLMATRIX2_temp(qik(c1,2),qik(c1,3),dest_cell)=1;
% remove machine instance from donor cell
CELLMATRIX2_temp(qik(c1,2),qik(c1,3),qik(c1,1))=0;
% target cell sends a pair -i,k (from machine_pairs_donate) to donor cell ->
CELLMATRIX2_temp(machine_pairs_donate(1,1),machine_pairs_donate(1,2),qik(c1,1))=1;
% remove this pair from receiver cell
CELLMATRIX2_temp(machine_pairs_donate(1,1),machine_pairs_donate(1,2),dest_cell)=0;
% locally store the possible attempt of donor to
% receiver using the receiver index in the 4th-dimension
%keyboard
CELLMATRIX2_4D(:,:,:,i_dest)= CELLMATRIX2_temp;
%temporarily find obj val
[OBJVAL_tempx,part_moves_tempx,W_jq_tempx,SOL_tempx] = ...
      CalcObjValPA(UTIL_sortedi,CELLMATRIX2_4D(:,:,:,i_dest),NP,NC,EMAX,NM,KMAX,L,UTIL,SETUP);
OBJVALM(i_dest)= OBJVAL_tempx; % store current objval
tl_tempx = [tl_tempx;[i_dest qik(c1,:) tt;i_dest dest_cell qik(c1,2) qik(c1,3) tt;...
```

```
                             i_dest dest_cell machine_pairs_donate(1,1),machine_pairs_donate(1,2) tt;...
                             i_dest qik(c1,1) machine_pairs_donate(1,1),machine_pairs_donate(1,2) tt]];
        ObjValProfile = [ObjValProfile; OBJVALM(i_dest)];
        i_objval=i_objval+1;
        i_objvalprofile = [i_objvalprofile;i_objval];
    end
end
% check for empty TL
if ~isempty(tl)
    [tlf]=find(tl(:,4)==0);
    tl(tlf,:)=[]; % if any remove
end
%---- compare solutions before moving on and implement accordingly
% classification of attempts based on OBJVAL
[OBJVALM_sorted,OBJVALM_sortedi] = sort(OBJVALM,'ascend'); %from local best to worst
OBJVALM_sorted
CELLMATRIX2_4D_sorted=CELLMATRIX2_4D(:,:,:,OBJVALM_sortedi);
ix2=[];% sort the tl_tempx list according to the sorted objvals (mapping index multiplicities)
ix3=[];
ix3_=[];
ix4=[];
ix5=[];
% find the segments in tl_tempx corresponding to each generated OBJVALM_sortedi,
% i.e. to each generated solution
for ix=1:length(OBJVALM_sortedi)
    ix2 = [ix2;find(OBJVALM_sortedi(ix)==tl_tempx(:,1))];
end
% and sort them
tl_tempx_sorted = tl_tempx(ix2,:);
% check if local best is better than the global best
if OBJVALM_sorted(1)<BEST_COST
    BestLocalVector = [BestLocalVector;[(i_objval-length(all_cells)):1:i_objval]' ones(length(all_cells)+1,1)*OBJVALM_sorted(1)];
    % in fact from theory we need to check if it is also in the
    % TL but due to the AC we can spare this checking statement
    % and implement directly and update TL regardless
    % update CELLMATRIX2_temp
    CELLMATRIX2_temp = CELLMATRIX2_4D_sorted(:,:,:,1);
    % perform a part reallocation
    [OBJVAL_temp,part_moves_temp,W_jq_temp,SOL_temp] = CalcObjValPA(UTIL_sortedi,CELLMATRIX2_temp,NP,NC,EMAX,NM,KMAX,L,UTIL,SETUP);
    % update global cost value
    BEST_COST=OBJVAL_temp;
    % Please note that OBJVAL_temp and OBJVALM_sorted(1) are exactly the same as the first is produced
    % by employing the same CELLMATRIX2. Part reallocation is only performed to receive the updated
    % part_moves_temp and W_jq_temp needed within the iterative procedure
    % update global solution
    BEST_SOL=SOL_temp;
    % update machine cell allocation
    BEST_CELLMATRIX2=CELLMATRIX2_temp;
    % update part machine cell allocation relative to the part machine sequence
    BEST_part_moves = part_moves_temp;
    % update distinct allocation of parts to cells
    BEST_W_jq=W_jq_temp;
    clear OBJVAL_temp SOL_temp % clear these two as they are of no further usage
    % update tabu list
    % find the segment in tl_tempx_sorted corresponding to the implemented move
    ix3=find(OBJVALM_sortedi(1)==tl_tempx_sorted(:,1));
    % find the segments for the remaining neighboring solutions in tl_tempx_sorted
    ix3_=find(OBJVALM_sortedi(1)~=tl_tempx_sorted(:,1));
    % if tabu list is empty
    if isempty(tl)
        % place all moves for implemented (reverse and forward)and for the remaining
        % only the forward (in the tabu list the reverse move is placed first and then the forward)
        tl=[tl_tempx_sorted(ix3,2:5);tl_tempx_sorted(ix3_(2):2:ix3_(length(ix3_)),2:5);tl];
        % tl_tempx_sorted(ix3_(2):2:ix3_(length(ix3_) meaning
        % start from the second row and consider by step of two the next entry till the length has been reached
    else
        % referring to local segment of implemented move (reverse and forward)
        for wx1=1:length(ix3)
            xx2=1;
            while xx2<=length(tl(:,1))
                % if all entries both reverse and forward are in the tabu list delete them for now...
                if all(tl_tempx_sorted(wx1,2:4)==tl(xx2,1:3))
                    tl(xx2,:)=[];
                end
                xx2=xx2+1;
            end % if not do not add in tabu list
        end
        % referring to the remaining segments only for forward moves (even indices)
        for wx2=ix3_(2):2:ix3_(length(ix3_))
            xx3=1;
            move_exist1=[];
            while xx3<=length(tl(:,1))
                % if the forward moves exist increment
                % move_exists1 by one
                if all(tl_tempx_sorted(wx2,2:4)==tl(xx3,1:3))
                    move_exist1=move_exist1+1;
                end
                xx3=xx3+1;
            end % if not do not add in tabu list
            % if they don't exist add the forward moves in tl
            if isempty(move_exist1)
                tl=[tl_tempx_sorted(wx2,2:4) tt+1;tl];
            end
        end
        %update all
        tl=[tl_tempx_sorted(ix3,2:5);tl(:,1:3) tl(:,4)-1];
    end
else % else if worst than BEST_COST find which one to implement from all neighboring solutions
     % choose to implement that one whose forward move is not in tabu list
    counter_objval_= counter_objval_+1;
    collect_notimpl=[];
    if ~isempty(tl)
        for wx3=1:length(OBJVALM_sortedi)
            ix4=find(OBJVALM_sortedi(wx3)==tl_tempx_sorted(:,1)); %
            xx4=1;
        while xx4<=length(tl(:,1))
            if all(tl_tempx_sorted(ix4(2),2:4)==tl(xx4,1:3))
                % put in collect_notimpl these object val. whose forward moves are in the tabu
                % list, therefore they can't be implemented
                collect_notimpl = [OBJVALM_sortedi(wx3);collect_notimpl];
                break
            end
            xx4=xx4+1;
        end
    end
    % take all
    % choose to implement the local solution not in TL
    [impl_candidate,impl_candidatei] = setdiff(OBJVALM_sortedi,collect_notimpl);
    %[ci,c] = setdiff(A, B) returns the values in A that are not in B in c and the corresponding indices in ci
% if the tabu list is empty that means that the neighboring solution with the smallest objective value can be
% implemented (the first elemenst in sorted OBJVALM)
else
    impl_candidatei=1;
end
% if there is at least one neighboring solution that is not
% tabu
if ~isempty(impl_candidatei)
    % take the minimum of all and update the allocation of machines to cells accordingly
    CELLMATRIX2_temp = CELLMATRIX2_4D(:,:,:,OBJVALM_sortedi(min(impl_candidatei)));
    % perform part reallocation and evaluation of the objective value
    [OBJVAL_temp,part_moves_temp,W_jq_temp,SOL_temp] = CalcObjValPA(UTIL_sortedi,CELLMATRIX2_temp,NP,NC,EMAX,NM,KMAX,L,UTIL,SETUP);
    BestLocalVector = [BestLocalVector;[(i_objval-length(all_cells)):1:i_objval]' ones(length(all_cells)+1,1)*OBJVAL_temp];
    clear OBJVAL_temp SOL_temp
    % update TL...
    % find the segment referring to the implemented move
    ix5=find(OBJVALM_sortedi(min(impl_candidatei))==tl_tempx_sorted(:,1));
    % find the segments of the remaining moves
    ix5_=find(OBJVALM_sortedi(min(impl_candidatei))~=tl_tempx_sorted(:,1));
    % if tabu is empty place all moves: both reverse and forward of the implemented + the forward moves of the non implemented
    if isempty(tl) % if TL empty place all moves by default
        tl=[tl_tempx_sorted(ix5,2:5);tl_tempx_sorted(ix5_(2):2:ix5_(length(ix5_)),2:5);tl];
    else
        for wx4=1:length(ix5) % referring to local segment of implemented move (reverse and forward)
            xx5=1;
            while xx5<=length(tl(:,1))
                if all(tl_tempx_sorted(wx4,2:4)==tl(xx5,1:3))
                    tl(xx5,:)=[];
                end
                xx5=xx5+1;
            end % if not do not add in tabu list
        end
        for wx5=ix5_(2):2:ix5_(length(ix5_)) % referring to the remaining segments only for forward moves (even indices)
            %wx5
            xx6=1;
            move_exist2=[];
            % check if the forward moves exist in tl; if
            % they do leave them as they are, do not update the tt; otherwise place them with tt=t_max
            while xx6<=length(tl(:,1))
```

```
                                %size(tl_tempx)
                                %tl_tempx_sorted(wx5,2:4)
                                %keyboard
                                if all(tl_tempx_sorted(wx5,2:4)==tl(xx6,1:3))
                                    move_exist2=move_exist2+1;
                                end
                                xx6=xx6+1;
                            end % if not do not add in tabu list
                            if isempty(move_exist2)
                                tl=[tl_tempx_sorted(wx5,2:4) tt+1;tl];
                            end
                        end
                        %update all (implemented placed in tl_tempx_sorted and rest are reduced by one)
                        tl=[tl_tempx_sorted(ix5,2:5);tl(:,1:3) tl(:,4)-1];
                    end
                else % if all local solutions are in TL continue from global best, or least worst among the neighbors
                     % or worst neighbor (chose the more appropriate)
                    %keyboard
                    %CELLMATRIX2_temp = CELLMATRIX2_4D(:,:,:,length(OBJVALM_sortedi)); % current worst
                    %CELLMATRIX2_temp = CELLMATRIX2_4D_sorted(:,:,:,1); % current least worst
                    CELLMATRIX2_temp = BEST_CELLMATRIX2; % global best
                    % perform reallocation and evaluation of the objective function
                    [OBJVAL_temp,part_moves_temp,W_jq_temp,SOL_temp] = CalcObjValPA(UTIL_sortedi,CELLMATRIX2_temp,NP,NC,EMAX,NM,KMAX,L,UTIL,SETUP);
                    BestLocalVector = [BestLocalVector;[(i_objval-length(all_cells)):1:i_objval]' ones(length(all_cells)+1,1)*OBJVAL_temp];
                    clear OBJVAL_temp SOL_temp
                end
            end
            %----
            counter_source=c1;
            i_source=i_source+1; % accumulate iteration counter; move onto next move
        end
        counter_part=j2;
        i_part=i_part+1;
    end
    i_global=i_global+1; % accumulate iteration counter for overall procedure
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [OVx,PMT,WJQ,SOLx] =
CalcObjValPA(UTIL_SI,CMTRX,NP,NC,EMAX,NM,KMAX,L,UTIL,SETUPx)
% calculate obj val for given problem with new part allocation
% do a part allocation
[PMTRX,STP_matrix,PMT] =
PartAllocation(UTIL_SI,CMTRX,NP,NC,EMAX,NM,KMAX,L,UTIL);
% evaluate all elements encountered in the objective function
[xtral] = FindExtraMoves(UTIL_SI,PMT);
% S should be manipulated from PCMATRIX_temp
[setup_costtx,S_ijq_x,S_ij_x] = SetupCostTotal(STP_matrix,SETUPx);
[WJQ,W_x]=CalcW(PMT,UTIL_SI,NP,NC); SOLx={WJQ, S_ijq_x, xtral};
% evaluate current objective function
OVx = 10*sum(sum(WJQ)) + sum(sum(SETUPx.*S_ij_x)) + sum(xtral);
```