THE UNIVERSITY OF ADELAIDE

Design of Dynamic Cellular Manufacturing Systems

by

Mirko M. Bajic

The University of Adelaide
Faculty of Engineering
Department of Mechanical Engineering

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy on September 13, 2000.

Qualified on January 02, 2001

To my parents

ABSTRACT

Existing approaches to the cell formation problem concentrate on simultaneous part family formation and machine sharing. A fundamental problem faced when part families and independent cells are desired is that two or more cells may share a machine type. Then, an integer assignment to each cell, subject to total machine availability constraints, leads to underutilisation or overloading of the important machines in the cells. Practical shopfloor conditions such as machine failures, fluctuations in part mix or demands may also make cells unstable. These methods do not use flow data. Hence, they cannot solve together the machine grouping, machine sharing, intracell layout, intercell layout and handling subproblems. Beyond the logical grouping of the resources, another important aspect of manufacturing system design is the physical placement of these resources on the shop floor, which defines the actual flow patterns of parts.

This dissertation addresses, for the first time, an analytical approach to the integrated problems of designing the dynamic cellular manufacturing (*DCM*) system layout concurrently with its material flow (handling) requirements, in such a manner that minimises the material handling within the system. The proposed strategy encourages the design of a dynamic layout to identify simultaneously the machine groups, economical machine distribution, and intracell and intercell layouts.

Thus the proposed mathematical models are based on network analysis and graph structures for flowline decomposition of machine groupings, and flowline layout design of dynamic cellular manufacturing cells. To minimise travel distances for forward, and backward material flow arcs, the derived model minimises total travel distances and machine duplications. Consequently, stages I and II of this research generates machine groups, identifies a flowline layout for each group, indicates which flowlines must be placed adjacent to each other to minimise intercell distances, and an approximate configuration of the aisles. Thus by capturing the directionality embedded in the operation sequences of a variety of parts produced, the associated facility floor area can be optimised. It is concluded that the classification of flow arcs used is effective for assessing whether material handling and layout, or machine sharing, is necessary to minimise intercell or intracell material travel distances respectively.

As a development from stages I and II, stages III and IV outlines the associated economics of machine distribution and layouts of a dynamic cellular manufacturing facility. Furthermore, by employing simulated annealing *(SA)* algorithms the design (machine placement) of the shop layout for these dynamic cells can be optimised.

Illustrative case study material demonstrates that the sequential use of these proposed stages effectively integrates machine grouping and layout design requirements into dynamic cell configurations. It is established that these evolved dynamic cells have a superior utilisation of resources, when compared with equivalent classical cellular systems employing traditional design methods such as clustering analysis, operation sequence clustering and graph based layout techniques. In addition, parametric analyses of the proposed *SA* algorithm procedure shows an improvement in the cells responsiveness and effectiveness, with respect to the reported mean values of the material flow cost function. Furthermore, examples are utilised for both equal and unequal resource dimensions, with the average value of the material flow cost comparable with other methodologies. It should be appreciated however, that the *SA* algorithm was programmed in *MATLAB*, and has matrix and graphical interface outputs, which is again a further step forward in this research area.

Statement of Originality

To the best of the author's knowledge and belief all of the material presented in this thesis, except where due references is made in the text, has not been presented previously for the award of any degree or diploma in any University.

If accepted for the award of the degree of Doctor of Philosophy, the author consents that this thesis copy, when deposited in the University Library, be made available for loan and photocopying.

Signature:

Date: 06/02/2001

Acknowledgments

The long and lonely journey towards the completion of a doctoral dissertation is one of the periods in life when you learn to appreciate truth and friendship. I was fortunate to have Dr. K. Baines and Dr. C. Cresswell as my supervisors during my four years at the University of Adelaide. I would like to gratefully acknowledgment their guidance, encouragement and continuous support of this research project and preparation of this thesis. I appreciate the freedom that they gave me to pursue my own ideas and to explore new areas, and the invaluable advice that they offered to maintain a clear goal as my work progressed. At the end, they became more than supervisors and collaborators; good friends.

I would also like to thank colleagues and staff members in the Department of Mechanical Engineering of The University of Adelaide for their valuable input and support during my studies. In particular, special thanks to A. Prof. M. Zockel for his inputs and helpful comments and suggestions.

This work was supported by an Australian Postgraduate Award from the Australian Government. The financial support from The University of Adelaide is also greatly appreciated.

I am extremely grateful to my lovely daughters (Jelena and Milica) who have been patient throughout this research while they had to stay at home for many weeks and wait for me many late nights.

List of Contents

List of Figures

List of Tables

Acronyms

AI – Artificial Intelligence

AGV – Automated Guided Vehicle

ALDEP – Automated Layout Design Planning

BEA – Bound Energy Algorithm

CAD – Computer Aided Design

CAM – Computer Aided Manufacturing

CC – Classification and Coding

CF – Cell Formation

CM – Cellular Manufacturing

CMS – Cellular Manufacturing System

CNC – Computer Numeric Centre

COFAD – Computerised Facility Design

CORELAP – Computerised Relationship Layout Planning

CRAFT – Computerised Relative Allocation Facilities Technique

DCF – Discounted Cash Flow

DCM – Dynamic Cellular Manufacturing

DCMS – Dynamic Cellular Manufacturing System

DLP – Dynamic Layout Planning

DOLA – Dynamic Optimal Linear Arrangement

ES – Expert System

EA – Evaluation Algorithm

FFA – Factory Flow Analysis

FMS – Flexible Manufacturing System

GA – Genetic Algorithm

GT – Group Technology

KBES – Knowledge Based Expert System

LA – Line Analysis

MAPI – Machinery and Allied Products Institute

MC – Machine Cell

MCST – Minimum Cost Spanning Tree

MPG – Maximal Planar Graph

MPWG – Maximal Planar Weighted Graph

MST – Maximum Spanning Tree

MWDRST – Maximal Weighted Directed Rooted Spanning Tree

MWDST - Maximal Weighted Directed Spanning Tree

NP – Non Polynomial

OLA – Optimal Linear Arrangement

OPT – Optimised Production Technology

PFA – Production Flow Analysis

PF – Part Families

QAP – Quadratic Assignment Problem

ROC – Rank Order Clustering

ROI – Return On Investment

SA – Simulated Annealing

SCP – Similarity Coefficient Product

SCTF – Similarity Coefficient Total Flow

SDPI – Steepest Descent Pairwise Interchange

SLP – Systematic Layout Planning

SP - String Processing

TC – Total Cost

TCR – Total Closeness rating

TOC – Theory Of Constraints

TSP – Travelling Salesman Problem

WDRST – Weighted Directed Rooted Spanning Tree

WIP – Working In Progress

WDRT – Weighted Directed Rooted Tree

CHAPTER ①

Introduction

1.1. Introduction and Significance

In the early years (early 20th century) of the industrial era manufacturing facilities were generally laid out as a process orientation, meaning that all similar machines were grouped together. This configuration is known to cause long lead times, excessive work in progress and significant material handling. As demand grows, production grows and typically, travelling distances increase due to the larger number of machines and racks to store work in progress. Manufacturing engineers realised that product oriented layout, or flow line organisation, would in many cases be more efficient by reducing lead-time, work in progress and handling.

However, these advances are not new. In 1715 Adam Smith in his book “The Wealth of Nations” described how a number of workers, specialising in different processes, worked together as a team to make pins. He was probably the first to describe group technology [Smith 1715, Burbidge 1995].

In the later years of the 20th century, the popularity of group technology *(GT)* cells increased as a response to a quickly changing market. Group technology cells (cellular manufacturing) is one of the better approaches for effective arrangement of the shop resources. Theoretically, this system is capable of superior flexibility, product quality and productivity, whilst reducing lead-time and work in progress. Thus, this manufacturing system permits quick adaptation to any change in the market place. *GT* cell concepts have intrinsic values but also have serious weaknesses, which are characterised by low machine utilisation and lack of flexibility. A definition of a manufacturing system with its inputs and outputs is presented in *Figure 1.1.*

1.2. Group Technology Definition

Mitrofanov [1959] conceived the *GT* concept as a manufacturing philosophy. Gallagher [1973] also reported that *GT* was developed to a sophisticated level in the U.S.S.R. before it was applied elsewhere. In the early years of development it aimed at grouping the components to be processed at a given the early years of development it aimed at grouping the components to be processed at a given processing facility, utilising the information from route cards. A major part of the work in *GT* was directed towards its practical applications to different specific production problems. However, analytical work in *GT* has lagged behind its industrial applications. In 1979 Burbidge advocated that *Group Technology,* by identifying similar parts and grouping them together in families, could take advantage of their similarities in design and manufacture.

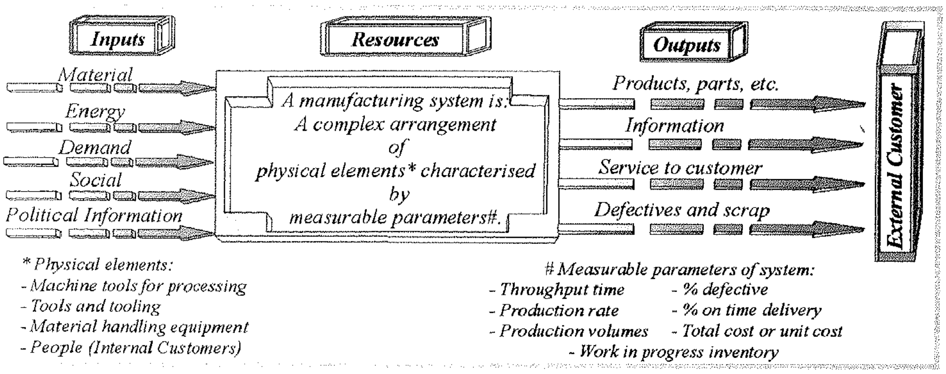


Figure 1. 1 - Definition of a manufacturing system with its inputs and outputs

Cellular manufacturing *(CM)* has been recognised as one of the recent technological innovations for improving productivity and competitiveness. By dedicating a machine cell to the production of a part family, many of the efficiencies of mass production can be realised in a less repetitive batch environment. Reduction in set up times, work-in-progress inventory levels, and production lead times are some of the benefits associated with *CM* systems.

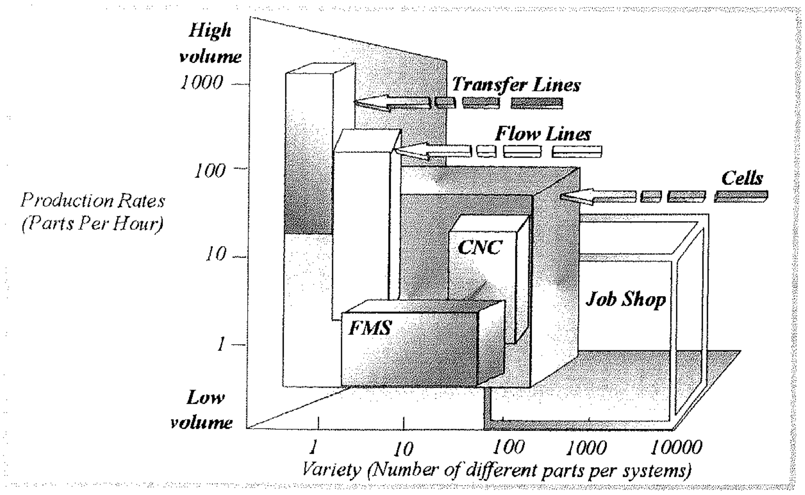


Figure 1. 2 - A comparison of different kinds of manufacturing systems

1.3. The Concept of Cellular Manufacturing

The traditional approach to the organisation of production is to use line layouts for high volume production, and functional layouts for low volume cases. With traditional methods of manufacturing a reduction in batch sizes would result in higher manufacturing costs due to increased cost of setups. *GT* has invalidated this relationship and given economies to small batch manufacturing, which were earlier believed possible only for high volume production. Thus *GT* is a proven technique for improving productivity in batch production systems utilising *CM* principles. A current comparison of the different

The machines, which are needed to produce each family of parts, are grouped together in what is called a *manufacturing cell*. For example, a manufacturing cell might include a lathe, a drill, a milling machine and a grinder. The first phase in the design and implementation of a *CM* system is the identification of part families and machine cells. This phase is commonly referred to as the Part Family/Machine Cell *(PF/MC)* formation problem. *PF/MC* formation is the process of analysing part and machine populations, grouping parts with similar design features, tooling requirements, or manufacturing routines, into families, and grouping the required machines into cells to produce the *PF's*. This problem has captured the attention of researchers for over a decade and numerous methodologies have been proposed. Missing from the literature, however, are case studies reporting the usefulness of these techniques in designing an actual *CM* system layout [Wemerlow 1987]. The continued development of new procedures, coupled with minimal reported successes, indicates that the *PF/MC* formation/layout problem has yet to be resolved to the satisfaction of researchers and practitioners alike.

Thus it is only during the last decade that the rapid redesign of *CM* systems has begun to receive a great deal of attention. Production flow analysis, mathematical programming, graph theory, statistical clustering, are some of the approaches that have been suggested for part and machine cell groupings. A number of other methods have been developed, but the *CM* rapid redesign problem is yet to be solved satisfactorily for many industrial situations. This is the case especially with respect to intercell and intracell formation and for the increasing requirement of low volume, high variety products.

1.4. The Practical Significance of Cellular Manufacturing

The descriptive and case study literature concerning *CM* argues that batch lot users may achieve a number of significant advantages. Clearly grouping parts into families should lead to lower average machine setup (change over) times, because successive batches processed by a machine will be similar due to grouping of the parts and dedication of the machines. Lowering the average set-up time should lead to the use of smaller batch sizes, increasing the flexibility of the shop. As the average set-up time is reduced, the machine utilisation per batch should be lower, where utilisation is defined as the percent of total available time that a machine is busy. This should lead to an effective increase in capacity, permitting more products to be processed in a fixed amount time. Hyer [1989] states that grouping machines into manufacturing cells should reduce the average material handling time, since much of the travelling will be within the cells. In addition, such cells should reduce the length of queues to machines, lowering the work in progress inventory [Suresh 1985]. The combination of these factors implies faster average throughput time.

CM is especially effective in improving the productivity of multi-product, small-to-medium batch size manufacturing systems, which are computer-integrated. Here a particular application of *GT* is the design of a *CM* system, where parts, having similar or identical machine requirements and operation sequences, are grouped into families, and the complete set of machines and their capacity routines are grouped into machine groups or cells. A typical 3D-cell model is presented in *Figure 1.3.*

machine groups or cells. A typical 3D-cell model is presented in *Figure 1.3*.

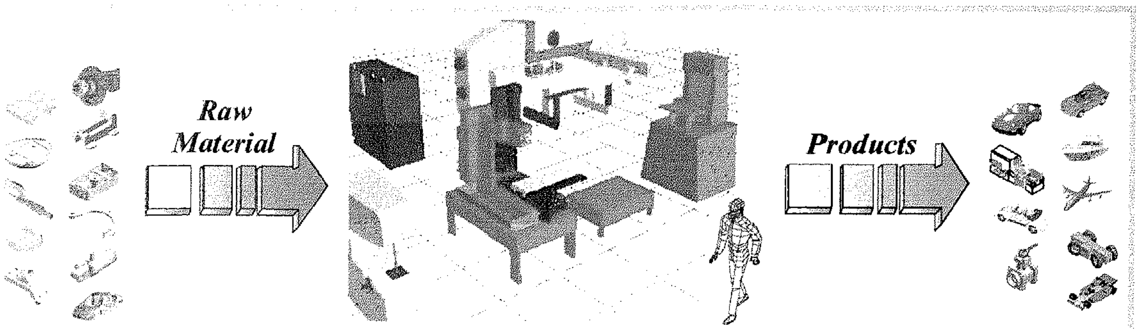


Figure 1. 3 - A 3D model of the classical cell within a manufacturing system

The effectiveness of a *CM* system is influenced by fluctuations in product demand, product mix, and resource availability. The majority of existing *PF/MC* models assumes each of these factors to be constant. In reality, the demand for a product varies according to its stage in the product life cycle, when new products are introduced, and when the production of older products is discontinued. In addition, resources are continually being replaced due to age and/or obsolescence.

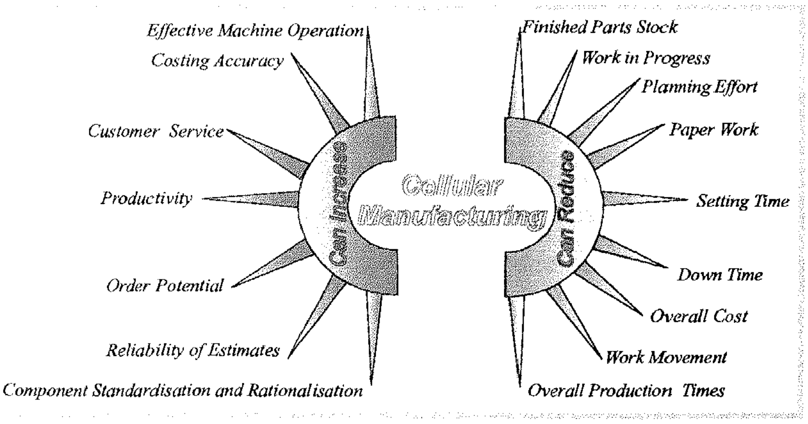


Figure 1. 4- Advantages and disadvantages of GT cells

On the other hand, disadvantages (as shown in *Figure 1.4*) of CM systems are that they are highly sensitive to machines failure and their performance rapidly deteriorates under changes in product mix and inadequate load balancing. Other disadvantages are:

- cells can lead to uneven and sometimes low machine utilisations, cell systems sometimes require the duplication of equipment (additional investments), cell systems are less flexible than job shops (as a result, their performance can actually worsen unless properly designed), cell systems are costly to construct due to cost of machine re-allocation, cell systems are vulnerable to equipment failure and require a relatively higher emphasis on maintenance than job shops, and cell system operations require strict discipline over time in order to avoid non-cell parts being routed to the cells and distorting flows and performance.

The traditional approach to cell formation is to use machine-part matrix clustering to create an independent cell for each part family. Machine sharing and intercell flows are discouraged. However, this creates the problem of determining the number of shared machines that must be assigned to each cell, without affecting machine utilisation. Their input data usually does not consider flow directions and

Hence, these approaches fail to relate the intracell and intercell layout problems to the machine grouping and sharing decisions. With product mixes and demands being subject to change, this rigid traditional definition of cells does not totally support current design principles of flexibility and effective factory integration. Thus, any new cell formation method must address the critical issues of:

- part formation, machine grouping, machine sharing between cells, intracell layout and intercell layout, to meet organisational and operational goals of a company.

1.5. The Fundamental Problems in *CM*

The fundamental problem of *CM* system design is the identification of part families and the composition of machine cells. Given a set of parts, processing requirements and available resources, the objective of the *PF/MC* formation problem is to obtain a satisfactory partition of parts into families, and machines into cells, with respect to one or more design objectives. The system design must be capable of producing the required volume of parts without exceeding the capacity of the resources. Thus the purpose of this section is to outline issues related to formulating and solving the *PF/CM* formation problem, and to identify the desirable characteristics of a new design methodology.

The design of a *CM* system has a number of objectives. The following list of objectives, taken from Ballakur [1987] and Wemmerlow [1988], are as follows:

- minimise throughput times; minimise setup times; minimise inventories; maximise resources utilisation; maximise output; minimise machine relocation costs; minimise intracell and intercell moves of material; minimise operating costs; minimise investment; minimise the number of cells; minimise duplication of machines in different cells; maximise the percentage of operations of a part processed within a single cell with minimal setup time; minimise job lateness; obtain a flow line structure within the cell.

Consequently, the *CM* system design process is a multi-objective task. Some of these objectives are conflicting and will require trade-offs during the design process. The majority of cell formation techniques focus on a single objective, such as minimising intercell movement, or minimising capital investment on new machines. Clearly these objectives are conflicting; one can minimise intercell movement by investing in new machines, alternatively reduced capital investment may come at the expense of increased material handling. Thus, any new methodology should address the multi-objective nature of the *PF/MC* formation problem by including multiple objectives in the design evaluation.

A variety of manufacturing system parameters must also be considered during the process of forming part families and machine cells. Typical parameters used in the *PF/MC* formation problem are as follows:

- part processing requirements; available resources; operation times; set up times; production volumes; resource capacities.

Depending on the design objective, certain cost information is also needed. For example, material handling costs, and costs of acquiring additional resources, would be required to evaluate the trade-off handling costs, and costs of acquiring additional resources, would be required to evaluate the trade-off between machine duplication and intercell movement. Thus, the new methodology developed to solve the *PF/MC* formation problem should include all relevant parameters and costs.

Existing *PF/MC* formation techniques fail to address the dynamic nature of the production environment. The underlying assumption of these techniques is that the part and machine populations are fixed and that demand is constant. As previously mentioned, production volumes will vary from one time period to another, and the machine population will also change as older machines are replaced and new technology is acquired. Any new methodology should address the dynamic nature of the production environment by incorporating a multi-period forecast of product mix and resource availability. This requires the specification of the system parameters by time periods.

Furthermore, to obtain a feasible design, certain constraints must be considered during the design process. For example, there must be sufficient resource capacity to produce the required production volumes. Other typical constraints include restrictions placed on capital investment, machine utilisation, the number of machines assigned to a cell, and the number of cells formed. The new methodology should include all such relevant system constraints.

Once the design objectives, system parameters and constraints have been identified, a mathematical formulation of the *PF/MC* formation problem can be developed. In the case of a single objective, such as minimising intercell transfers, the problem is frequently modelled as a mixed integer program. Some researchers however, have difficulty solving the problem directly using mathematical programming techniques, due to the large number of variables and constraints involved [Shafer and Rogers, 1991].

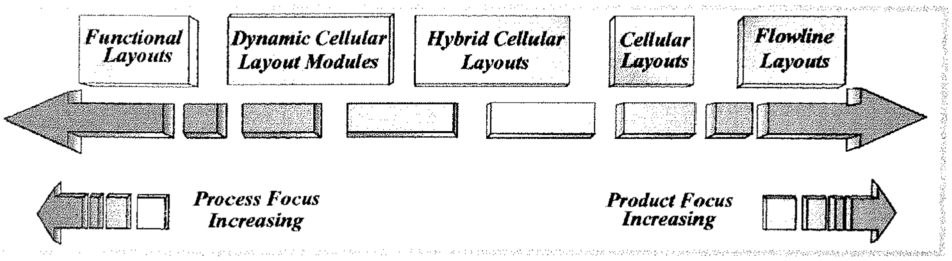


Figure 1. 5 - Proposed research-motivating layout parameters

In general, it is not easy to identify an optimal (non dominated) solution to a multi-objective problem with conflicting objectives. In addition, it is not possible to include all relevant factors in a model of the *PF/MC* formation problem. Thus, intangible criteria that are not easily measured are difficult to include in the design objective function. Clearly, it is essential that the decision-maker be presented with a set of solvable designs. The alternative designs can then be evaluated with respect to overall system performance. Any new methodology should be capable of generating good alternative solutions. Thus, motivating layout parameters for the proposed research are shown in *Figure 1.5*.

1.6. Research Objectives

- This dissertation addresses an innovative analytical approach to the integrated problems of designing the dynamic cellular manufacturing (*DCM*) system layout concurrently with its radically different approach to the cell formation problem. Specifically, this study provides a method to derive dynamic cell designs that are economic and efficient to operate. In doing so it considers the topology of the material flow network (skeleton), cell formation, intercell layout, intracell layout and the determination of the layout of the resource groups on the shop floor, which are all highly interrelated issues. In an ideal cellular manufacturing system only the intracell layout, or the physical placement of machines within a cell, is important, since each cell is independent and there is no transfer of parts between cells. However, in the arrangement of dynamic cells both intracell and intercell layout (ie. physical placement of machines and cells within the shop floor) are important.

The design problem is modelled by a comprehensive mathematical program which captures critical practical concerns such as operational and handling costs, material flow congestion and material flow travelling distances and capacities. The model is decomposed into three Non Polynomial time solvable *(NP-hard)* subproblems. Part family formation at the start is avoided. Cell formation is dynamic ie. the layout allows a particular machine to process parts from one or more families. For the first step, the travel chart as the input data is identified. Next, the dis-aggregation of the manufacturing shop into dynamic cells minimises the travelling of the parts between machines. By developing an analytical optimisation approach, the dynamic cell formation establishes the assignment of parts to machines and logical grouping of machines into cells. In the first subproblem, a maximal weighted directed rooted spanning tree is obtained. This yields machine groups, a material flowline layout network for each cell, and allows machine sharing amongst cells without physically separating them. A tree layout for the material flowlines is also suggested. The second subproblem concerns the reorientation of the material flow network (pivoting optimum linear arrangement), which is a special case of the *QAP*, to further minimise the intercell material flow network distances. Furthermore, parts which cause intercell flows and machine sharing amongst nonadjacent cells can be identified for more detailed analyses. In essence, this method exploits layout and handling strategies to minimise machine duplication for both intracell and intercell flows. For the material flow routing problem, optimal polynomial solutions are derived by an integer programming model-algorithm, whilst solutions to the optimum arrangement of flows are delivered from the *QAP*.

Finally, an integrated *DCM* layout method combines an effective algorithm for the material flow design, with a simulated annealing scheme, to solve the global *DCM* shop design problem. Thus the shop layout of dynamic cells is developed in a manner that minimises material handling between the resources, which at the same time incorporates machine capacity, costs, setup times, and machine utilisation, as well as cell production rates and batches. These factors are incorporated in an economical machine duplication methodology. For this purpose the simulated annealing *(SA)* algorithm is employed which accounts fully for all the physical constraints. Once the sizes and shapes of the resources are known, the shop layout is determined by a similar algorithm. Thus, the resulting dynamic cell shop consists of the cells and associated machines. Throughout the proposed methodology a consistent objective is the minimisation of the material handling effort.

This innovative approach will address simultaneously most major decisions involved in manufacturing shop design, and provide near-optimal solutions. The approach will also generate an economic production system designed for reducing operational costs and will be validated by using several examples from the literature solved by established methods. Complementary theoretical justifications for the mathematical programming models will also be provided.

Several new issues will be addressed in this thesis. In the cell formation procedure a new methodology will be introduced by employing a maximum weighted directed rooted spanning tree from graph theory (commonly a undirected tree is employed). This addresses the practical need of reducing materials handling costs corresponding to each cell. In addition, the associated economics of machine duplication (economical duplication) are addressed. Furthermore, by considering the machine dimensions the physical placement of resources within the cell are established. Finally, a novel layout methodology is developed to design a dynamic shop that optimises a mix of resources. This approach will address the majority of shop floor-practical cases, where a pure cellular arrangement is not feasible.

In the worked examples, parametric analysis of the *SA* algorithm will show an improvement in the responsiveness and effectiveness of the proposed *SA* layout procedure. In addition, mean values of the material flow cost will be reported, which also is a further improvement, since current researchers rarely include in their reports parametric analysis results. The *SA* algorithm is programmed in *MATLAB*, has matrix and graphical interface outputs, which is again a further step forward in this research area. Comparison examples will be utilised for both equal and unequal resource dimensions, with the average value of the material flow costs comparable with other methodologies.

1.7. Overview of the Thesis

The thesis structure is as follows:

- The introduction and benefits of cellular manufacturing are discussed in Chapter 1.
- Chapter 2 presents a thorough review of the state-of-the-art design and layout of cellular manufacturing systems and discusses the various judgemental criteria utilised.
- Chapter 3 describes the concept of integrated *DCMS* layout design methodology with a critical analysis of cellular manufacturing approaches. In addition, the reasons are given for the continued requirement to apply Group Technology *(GT)* techniques, together with the consideration of machine sharing. It is shown that these techniques, which allow this simultaneous solution, are very efficient in delivering workable group technology solutions.
- Chapter 4 shows that the proposed method of forming dynamic manufacturing cells is a combination of flow and mathematical programming concepts applied to traditional *GT* cell layout design. However, the basic assumption here is that machine sharing, on a part family basis, creates load balancing problems. One possible solution is for the shared machines to be retained in functional layouts provided the machine sharing is within the cell or between adjacent

cells. It is thus proposed that the traditional complex problem of layout design is simplified by partitioning the layout design problem into two parts. The first part generates the basic "tree and branches flow" pattern, whilst the second part simplifies the flow by re-ordering the branches.

- In Chapter 5 the *DCMS* analytical approach is tested and compared against three other methodologies from the literature. In addition, further physical location of the machines is also considered.
- Chapter 6 addresses the issue of the economical duplication of machines, and the optimisation of physical location of the resources on the shop floor. Furthermore, a simulated annealing scheme is presented that integrates material flow design procedure with the layout problem, to complete the integrated solution strategy.
- Finally Chapter 7 discusses the conclusions of this work, highlights its contributions and the recommended avenues for further research.

CHAPTER 2

Literature Review of Cell Formation and Layout

2.1 Introduction

In this chapter various approaches are reviewed that have been adopted in an attempt to solve the problem of cellular layout, and forming machines into groups, as well as components into associated families for cellular manufacturing. *CM* has been recognised as one of the most innovative approaches to improve productivity and flexibility for today's many-products and low-volume production environment, because it can effectively transform batch-type production into line-type production.

In recent implementation of *CM* manufacturing personnel were interested in concurrent formation of part families and groupings of machine tools. In addition to the inherent advantages of physically grouping machines into cells, material handling distances are also quantitatively reduced. Although a large number of varied studies covering a wide spectrum of activities have been reported, all these approaches share a common feature, ie. each utilises a binary machine matrix, where each matrix element takes a value of *1* or *0*. A matrix element *(1)* means that a part will visit the corresponding machine. The inherent problem with these two-value matrix approaches is that a part will visit only one machine for a particular shape. There is no convention of specifying alternative machines for the feature of that part.

Recently, the importance of layout design has received significant attention. It is recognised that with the rapid changes in production techniques and equipment technology, very few companies are able to retain their current facilities and layouts without damaging their competitive position. Additionally, many previously purchased facilities are modified each year. The investment in these facilities attests to the importance of the subject matter of layouts. A good layout design will not only reduce capital costs, but also eliminate unnecessary activities, thus effectively increasing productivity. A number of papers introducing solution algorithms and computer-aided planning tools have been published. In particular, Tompkins [1984] and Francis [1992] have given surveys of these efforts.

Each of the four traditional plants layouts (shown in *Figure 2.1)* has specific features and situations of applicability, as discussed below:

- In a functional layout, machines with identical or similar processing capabilities are located together as single machines that can process parts from several part families. This type of layout has advantages such as high machine utilisation and high flexibility in allocating operations to several alternative machines. However, since the material follows complicated routes between machines, this results in long throughput times, high *WIP* levels and high material handling costs.

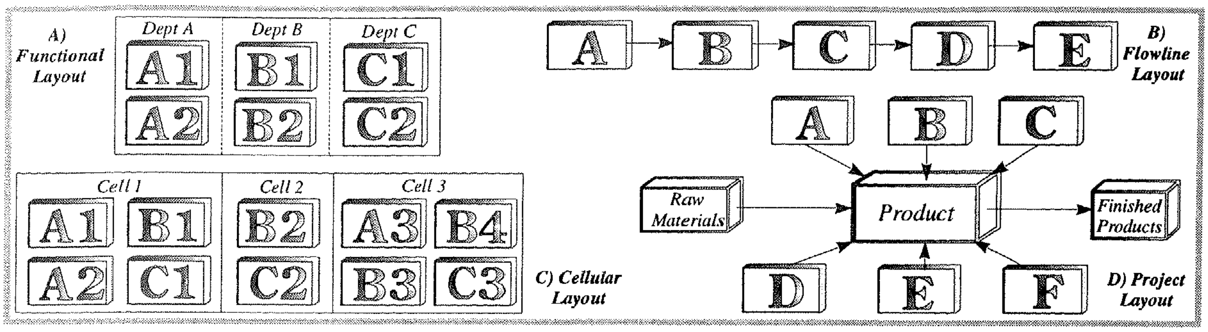


Figure 2. 1 - Types of plant layout

- In a flowline layout, different types of machines are arranged in a line, and parts typically flow from one machine directly to the next one. Because it encourages forward-in-sequence or bypass material flows, a flowline layout is preferable to the other layouts. Unfortunately, flowlines are usually infeasible for a multi-product plant due to the multiple material flow paths corresponding to the variety of operational sequences.

- In a cellular layout, parts are clustered into families based on common or similar operational sequences, and the machines required for each part family are grouped and placed together in a manufacturing cell. A cellular layout reduces throughput times, lowers material handling costs and decreases *WIP* levels. However, the disadvantages are duplication of machines and limited flexibility in case of machine failures, or changes in the part mix. A variation of this layout is the product layout that is dedicated to the manufacture of a single product or variations of the same product.

- In a project layout, the part or product to be manufactured remains stationary and all manufacturing equipment required is moved to the location of the product. Usually, such a layout is used only in shipbuilding, aircraft assembly, and other large scale construction projects.

2.2. Design of Cellular Manufacturing Systems

One of the first problems to be solved in the system design stage is called cell formation *(CF),* and here the task is to group parts with similar design features or processing requirements into part families, and forming the corresponding machines into cells. The *CF* problem has attracted much attention and considerable effort has been put into developing efficient procedures. Some references (Burbidge [1990 and 1995], Heragu [1993]) today see "process organisation" (organisation's based on process specialisation) as the "normal state" for production, and see "product organisation " (*CM*, continuous line flow) as only partially possible in a limited number of special cases.

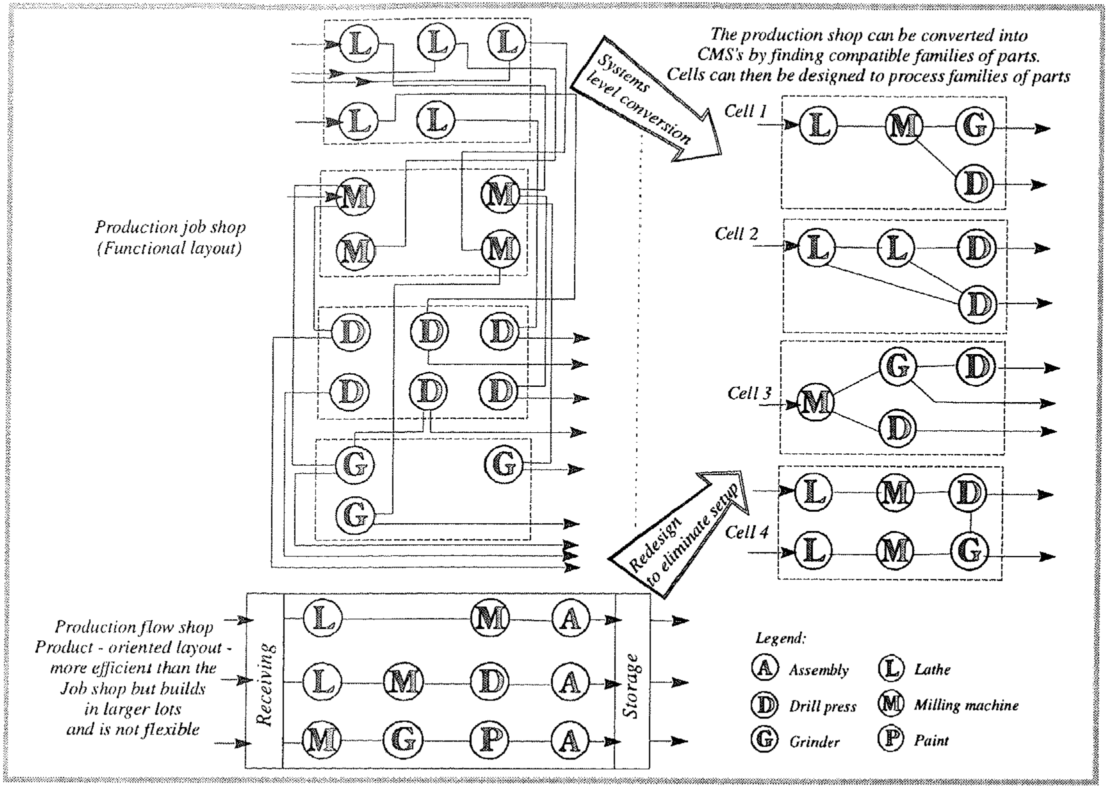

The production shop can be converted into CMS's by finding compatible families of parts. Cells can then be designed to process families of parts
Systems level conversion
Production job shop (Functional layout)
Cell 1
Cell 2
Cell 3
Cell 4
Redesign to eliminate setup
Production flow shop Product - oriented layout - more efficient than the Job shop but builds in larger lots and is not flexible
Receiving
Storage
Legend:
A Assembly
L Lathe
D Drill press
M Milling machine
G Grinder
P Paint

Figure 2. 2 - Two classical manufacturing systems in common use today require a system level conversion to be reconfigured into manufacturing cells [Black 1995]

The problem of grouping parts and machines for *CM* has been addressed extensively in recent years (Burbidge [1995] and Heragu [1993]). A numbers of methods have been developed, but the problem is yet to be solved satisfactorily for real industrial situations in conjunction with the necessary practical considerations in *CF* (such as machine utilisation, economical duplication, setup time etc.). In order to design cell systems, several methods, based on the analysis of production information, have been developed, leading to different structures of cell system. Two classical manufacturing systems in common use today require a system level conversion to be reconfigured into manufacturing cells (*Figure 2.2*).

2.2.1. Classification and Coding (*CC*)

Classification and coding offer advantages, such as design rationalisation and variety reduction, and enable the analysis to be carried out systematically. However, they involve an exhaustive scrutiny of drawings and design data. This method is more comprehensive than Production Flow Analysis *(PFA)* discussed in the next section.

There are design codes, manufacturing codes and codes that cover both design and manufacture. The main attributes of *CC* are as follows:

- *Classification* sorts items into classes or families based on their physical or functional similarities. It uses a *code* to accomplish this goal.
- *Coding* is the assignment of symbols (letters, numbers, or both) to specific component elements based on differences in shape, function, material, size, and manufacturing processes (*Figure 2.3.*

show the *CODE* system).

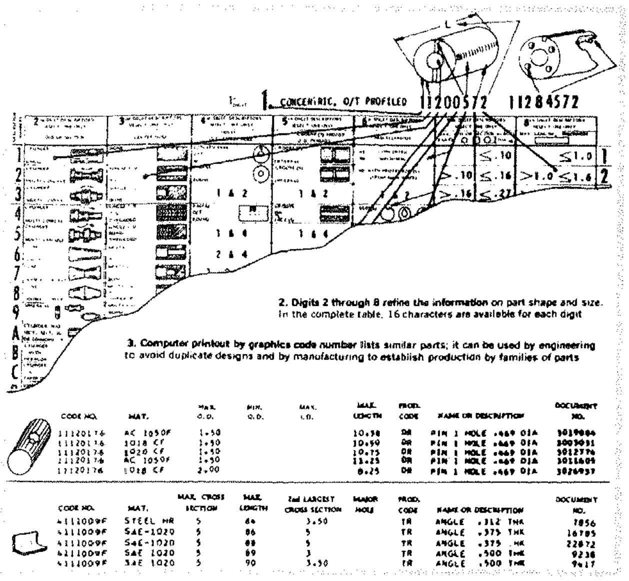

1. CONCENTRIC, O/T PROFILED
11200572
11284572
2. Digits 2 through 8 refine the information on part shape and size. In the complete table, 16 characters are available for each digit
3. Computer printout by graphics code number lists similar parts; it can be used by engineering to avoid duplicate designs and by manufacturing to establish production by families of parts
CODE NO.
MAT.
MAX. O.D.
MIN. O.D.
MAX. I.D.
MAX. LENGTH
PROD. CODE
NAME OR DESCRIPTION
DOCUMENT NO.

Figure 2. 3 - Classification and coding CODE system (Dornbush E., 1969, American Machinist)

2.2.2. Production Flow Analysis (*PFA*)

Burbidge [1995] wrote "If one uses *PFA* for *CM*, it is generally possible in any batch production or jobbing factory, to find a division of machines into groups, and of parts into associated families. With very few exceptions, these groups finished all the parts in the families of parts which they make, with no back-flow and no cross-flow between groups". It is generally possible therefore, to change from process organisation to *CM*.

PFA is a technique, which uses the information available on route cards to simplify material flow and to find the part families and machine groupings. The approach sorts through all the components, and groups them by a matrix analysis, using product routing information. This method is simple, cheap and fast, and *PFA* is a valuable tool in systems reorganisation.

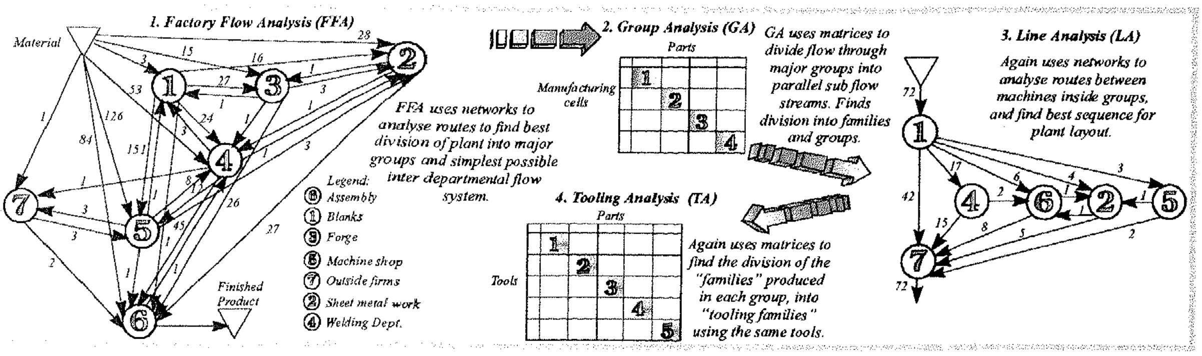

1. Factory Flow Analysis (FFA)
Material
Finished Product
FFA uses networks to analyse routes to find best division of plant into major groups and simplest possible inter departmental flow system.
Legend:
⑥ Assembly
① Blanks
③ Forge
⑤ Machine shop
⑦ Outside firms
② Sheet metal work
④ Welding Dept.
2. Group Analysis (GA)
Parts
Manufacturing cells
GA uses matrices to divide flow through major groups into parallel sub flow streams. Finds division into families and groups.
3. Line Analysis (LA)
Again uses networks to analyse routes between machines inside groups, and find best sequence for plant layout.
4. Tooling Analysis (TA)
Parts
Tools
Again uses matrices to find the division of the "families" produced in each group, into "tooling families" using the same tools.

Figure 2. 4 - Stages in production flow analysis

Production flow analysis *(PFA)* proposed by [Burbidge 1971 and 1995] (*Figure 2.4.*), requires subjective and non-quantifiable inputs into the decision process. When applied to a single cell, the classic framework for implementation of Production Flow Analysis (*PFA*) consists of four stages, each stage achieving material flow reduction for a progressively reducing portion of the factory ie. Factory Flow Analysis (*FFA*), Group Analysis (*GA*), Line Analysis (*LA*) and Tooling Analysis (*TA*), to develop a layout for a *CMS*.

In *FFA* (inter-departmental layout) *(Figure 2.4)*, dominant material flows between shops (or buildings) are identified. In addition, if parts are observed to backtrack between any of the shops, these flows are eliminated by a minor redeployment of machines.

		PART / PRODUCT																											
		L48388	*L48267B*	*M44276E*	*M47693F*	*L48388M*	*M44276D*	*E41795*	*E48596*	*E34267*	*E12204*	*E12288*	*K47697*	*E47782*	*E48586*	*K34596*	*E33494*	*M48265D*	*K44276C*	*M45691D*	*M45691B*	*M48386H*	*K34098A*	*E7392*	*E46364*	*E33295*	*K45199*	*K43590*	*E18694*
MACHINE	*DMT(3)*	X								X			X	X	X								X		X		X	X	X
	DM(3)		X	X			X				X		X			X	X	X					X			X	X	X	
	PG		X			X										X	X	X	X										
	DXY(3)	X	X													X							X				X	X	
	P&GR				X																								
	PGR											X													X				
	PGH																												
	PGG																			X	X					X			
	P&G						X	X	X	X	X											X	X			X	X	X	X
	RP																							X					
	PGB			X									X	X						X	X					X			
	W&P																						X				X	X	
	WG3										X																		

Table 2. 1 - PFA component – machine chart GA initial record. [Burbidge 1971]

In *GA* (intercell layout) (*Table 2.1 and 2.2*), the flows in each of the shops identified by *FFA* are analysed. *GA* analyses operational sequences of the parts being produced in a particular shop to identify possible manufacturing cells. Loads are calculated for each *PF* (Part Family) to obtain the machine requirements for each cell. Each cell usually contains the entire machines necessary to completely manufacture its *PF*. Due to sharing and non-availability of machines, some intercell material flows and flows to/from vendors may arise. *GA* uses cluster analysis of a binary machine-part matrix to form machine groups and part families simultaneously.

		PART / PRODUCT																												
		L48267B	*K34596*	*M48265D*	*E33494*	*K44276C*	*L48388M*	*E7392*	*K34098A*	*K45199*	*K43590*	*M48195C*	*M44276D*	*E34267*	*E12204*	*E18694*	*E41795*	*E48596*	*M48386H*	*L48388*	*M45691D*	*M45691B*	*M44276E*	*K47697*	*E47782*	*E46364*	*E12288*	*E33295*	*E48586*	*M47693F*
MACHINE	*PG*	X	X	X	X	X	X																							
	DM 3/1	X	X	X	X				**GROUP-1**																					
	DXY 3/1	X	X																											
	RP							X															**ONE “EXCEPTION”**							
	P&G		**FAMILY - 1**						X	X	X	X	X	X	X	X	X	X	X		**GROUP-2**							X		
	DMT 3/2								X	X	X			X		X				X										
	DM 3/2								X	X	X		X		X															
	DXY 3/2								X	X	X									X										
	W&P								X	X	X	X														**FAMILY -3**				
	WG3														X															
	PGG												**FAMILY - 2**						**GROUP-3**		X	X						X		
	PGB																				X	X	X	X	X			X		
	PGR																									X	X			
	DMT 3/3																							X	X	X			X	
	DM 3/3																						X	X				X		
	P&GR																													X

Table 2. 2 - Component - machine chart. Finding families and groups after GA complete [Burbidge 1971]

In *LA* (*Figure 2.4*), a linear or U-layout is designed for the machines assigned to each cell. *LA* assumes that the machine compositions of each cell are known. It uses flow data, captured in the operation sequences of the parts, and a travel chart giving the frequency of use of each routing, to develop a layout for each cell for efficient transport, as well as minimum material handling and travel by operators. In this research work the algorithms for line analysis have been applied to group analysis, thereby combining the two stages (*GA* and *LA*).

	Digit 1	*Digit 2*		*Digit 3*		*Digit 4*	*Digit 5*	*Digit 6*	*Digit 7*	*Digit 8*
	Method of holding	Dimension				Matching with			Material	Surface accuracy
		3 Jaw chuck		D_w	L	Special attachments	Boring tool carrier	Quadruple single point tool holder		
		Bore dia. ϕ	Overall							
0	3 Jaw chuck outer			< 40	L/D_w <0.1	w/o	w/o	w/o	GG-formed	rough turned ∇
1	3 Jaw chuck inner	ϕ42	160	41… 100	L/D_w <0.5	Axial copying	Boring, counter-sinking, reaming, tapping.	Uniform cutting, w/o accuracy.	ST-formed	fine turned ∇∇
2	4 Jaw chuck	ϕ60	250	101… 200	L/D_w up to limit of chuck	Face copying	Only outer turning.	Uniform cut, or staggered cut, with accuracy, simple boring up to ϕ48.	NE-formed	outer fit
3	Spring collet	ϕ80	315	301… 400	Shafts <500	2 Axis copying	1 with 2	Outer shaping, chamfering, inserting with form tool, not copying.	GG-cut off	inner fit (+ outer)
4	Mandrel or arbor	ϕ80	400	401… 500	Shafts 500… 1000	Conical Surface tapering±12°	Shaping, etc. with form tool; with 3; not copying.	3 with 4	ST-cut off	positional accuracy
5	Jig or fixture	ϕ125	500	501… 1000	Shafts 1m…2m	Steep cone	Inner shaping inserting chamfering ; with 3; copying.	Shaping, inserting chamfering with form tool; copying.	NE-cut off	polishing
6	Between centers			> 1000	Shafts 2m…5m	Short thread milling	Inner & outer at the same time	5 with 2 & 1 or 3	GG-bar	knurling, etc.
7	Chuck-center				Shafts > 5m	Threading with lead screw		6 with back tool holder	ST-bar	
8	Steadies					Thread with copying			NE-bar	
9	Eccentric (face plate)					Unround copying		Automatic cycle with 4th & 5th digits	non-metal	

Table 2. 3 - Tooling Analysis [Gallagher 1973]

In *TA (Table 2.3),* the principles of *GA* and *LA* are integrated with data on the shape, size, material, tooling, etc. and attributes of the parts. *TA* helps to schedule the cell by identifying families of parts with similar operational sequences, tooling and setups. It seeks to sequence parts on each machine and to schedule all the machines in the cell to reduce setup times and batch sizes. This increases available machine capacity on bottleneck machines in the cell. The intracell machine layout problem is addressed by *LA,* with *TA* seeking to minimise setup and tooling change times on essential machines within a cell by sequence dependent scheduling.

The input data, and hence the solution techniques, used in these four stages are different sequentially with almost no feedback between the stages. However, there are two difficulties within the *PFA* approach:

1. both factory flow analysis and group analysis are based on heuristic rules, and
2. the part-machine matrix showing the operations needed for component processing becomes too large to handle in the case of large numbers of components and machines.

Based on *PFA*, the objective of the cell formation is permutation of the columns and rows of the incidence matrix so that a block-diagonal structure is obtained. The resulting diagonal blocks representing the manufacturing cells. Clustering analysis is one of the most frequently applied mathematical tools in *CM*. The two basic formulations of the clustering models are matrix formulation, and mathematical integer programming formulation. In the matrix formulation, judgement regarding the number of clusters and the numbers of elements in each cluster is performed manually, whilst in the integer programming formulation both the number of clusters and elements are determined by the clustering algorithm technique.

2.2.3. Machine-Component Group Analysis

One of the frequently used representations of the *CM* problem proposed by Burbidge is a machine-part incidence matrix $[a_{ij}]$, which consists of '1' (empty) entries, where an entry 1 (empty) indicates that machine i is used (or not used) to process part j. Typically, when an initial machine-part incidence matrix $[a_{ij}]$ is constructed, clusters of machines and parts are not visible. A clustering algorithm allows the transformation of the initial incidence matrix into a structured (possible block diagonal) form. To illustrate the clustering approach to *CM*, consider the machine-part incidence matrix *(Equation 2.1)*.

$$\left[a_{ij}\right] = \textit{Machine number} \begin{array}{c|ccccc|} & \multicolumn{5}{c}{\textit{Part number}} \\ & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 & 0 \end{array}$$

Equation 2. 1

Rearranging rows and columns in matrix *(Equation 2.1)* results in matrix *(Equation 2.2)*. Two machine cells (clusters) *MC-1*= {2,4} and *MC-2* = {1,3}, and two corresponding part families *PF-1*= {1,3} and *PF-2*={2,4,5} are visible in this matrix.

$$\begin{array}{cc|cc|ccc|} & & \multicolumn{2}{c}{\overbrace{\quad}^{PF-1}} & \multicolumn{3}{c}{\overbrace{\qquad}^{PF-2}} \\ & & 1 & 3 & 2 & 4 & 5 \\ \hline MC-1 & \left\{\begin{matrix}2\\4\end{matrix}\right. & \begin{matrix}1\\1\end{matrix} & \begin{matrix}1\\1\end{matrix} & \begin{matrix}0\\0\end{matrix} & \begin{matrix}0\\0\end{matrix} & \begin{matrix}0\\0\end{matrix} \\ \hline MC-2 & \left\{\begin{matrix}1\\3\end{matrix}\right. & \begin{matrix}0\\0\end{matrix} & \begin{matrix}0\\0\end{matrix} & \begin{matrix}1\\1\end{matrix} & \begin{matrix}1\\1\end{matrix} & \begin{matrix}1\\0\end{matrix} \\ \hline \end{array}$$

Equation 2. 2

Clustering of a binary incidence matrix may result in mutually separable clusters and partially separable clusters. Mutually separable clusters are shown in matrix *(Equation 2.2)*, whilst partially clusters are presented in matrix *(Equation 2.3)*. *PF-2* from matrix *(Equation 2.2)* cannot be separated into two disjointed clusters because of part 5, which is to be machined in two cells *MC-1* and *MC-2*.

$$\begin{array}{cc|cc|cc|c|} & & \multicolumn{2}{c}{\overbrace{\quad}^{PF-1}} & \multicolumn{2}{c}{\overbrace{\quad}^{PF-2}} & \\ & & 1 & 3 & 2 & 4 & 5 \\ \hline MC-1 & \left\{\begin{matrix}2\\4\end{matrix}\right. & \begin{matrix}1\\1\end{matrix} & \begin{matrix}1\\1\end{matrix} & \begin{matrix}0\\0\end{matrix} & \begin{matrix}0\\0\end{matrix} & \begin{matrix}0\\0\end{matrix} \\ \hline MC-2 & \left\{\begin{matrix}1\\3\end{matrix}\right. & \begin{matrix}0\\0\end{matrix} & \begin{matrix}0\\0\end{matrix} & \begin{matrix}1\\1\end{matrix} & \begin{matrix}1\\1\end{matrix} & \begin{matrix}1\\0\end{matrix} \\ \hline \end{array}$$

Equation 2. 3

Removing part 5 from matrix *(Equation 2.2)* results in the decomposition of the matrix into two clusters, two machine cells, *MC-1*= {2,4} and *MC-2* = {1,3} and two part families, *PF-1* = {1,3} and *PF-2* =

{2,4} *(Equation 2.3)*. The two clusters are called partially separable clusters and the overlapping part (part 5) is called a bottleneck part (ie. the part that is processed on machines belonging to more than one machine cell). A way to eliminate a bottleneck part is to use an alternative process plan. For example, if an alternative process plan for part 5 in matrix *(Equation 2.3)* involves machines 2 and 4 this would result in two mutually separable machine cells. Alternative process plans are frequently available for many parts. It is obvious that grouping of parts with alternative routes increase the likelihood of generating ideal machine cells.

Analogous to the bottleneck part, a bottleneck machine is defined as a machine that processes parts belonging to more than one cell, ie. it does not allow for decomposition of machine-part incidence matrices into disjointed submatrices [Kusiak 1992]. The presence of bottleneck machines implies intercell flows can be eliminated by duplicating the bottleneck machines. Machine duplication is often expensive, and justified only if the savings achieved by reduction in intercell flows outweighs the initial expenditure for purchasing extra machines. It is often necessary to identify those machines whose duplication is a feasible economic option. This is performed using production data such as machine utilisation, machine availability, machine costs and travel costs. For this purpose Burbidge [1990] has classified machines into the following five major categories:

- S - special machines, only one of each type, whose work cannot usually be done by other machine types, eg. grinding machines; *I* - same as *S* except there are more machines of each type; *C* - common machines, many of each type, have many operations that can be performed on other machines, eg. lathes and mills; *G* - general machines, very few of each type, used for a very large variety of parts, eg. *CNC* machines, machining centres, and *E* -equipment machines, used to assist manual operations, eg. benches and vices.

Usually, the machine duplication problem concerns machines of types *I* and *C*. Machines of other categories lead to merging of cells or creation of common facilities cells (*S* and *G* types) or can be freely duplicated in every cell that needs them (*E* type).

Matrix formulation has two disadvantages:

- it is difficult to represent and visualise clusters for matrices having a large number of rows and columns, and
- in many cases it is difficult to obtain diagonal, or close to diagonal structure of the clustered matrix.

Whilst intuitive manual methods may be adequate for small problems, they become progressively less manageable, very time consuming and prone to failure when applied to larger problems. Burbidge [1977] has reported on the successful use of manual methods to enlarge machine-component matrices, but Carrie [1974] states that, "...Burbidge's assertion that manual sorting of the matrix is suitable for large problems up to 2,000 components is grossly optimistic".

2.2.4. Similarity Coefficient Methods (*SCM*)

2.2.4.1. The Single Linkage Cluster Analysis Method

The method of single linkage cluster analysis, developed by Sneath [1968] in the field of numerical taxonomy, was applied to the problem of group analysis in *PFA* by McAuley [1972]. He was the first author who applied similarity coefficients to solve cell formation problems. He used the Jaccard similarity coefficient to transfer the incidence matrix to a triangular machine-by-machine *(M × M)* similarity matrix. The method involves a hierarchical process of machine grouping in accordance with computed similarity coefficients. McAuley defines the similarity coefficient for any machine pair as, "the number of components which visit both machines, divided by the sum of components which visit one or other of the machines". This approach contains two essential steps; the first step defines the similarity coefficient, which shows the interdependence for each machine - machine combination, whilst the second step - clustering of machines, is based on similarity coefficients.

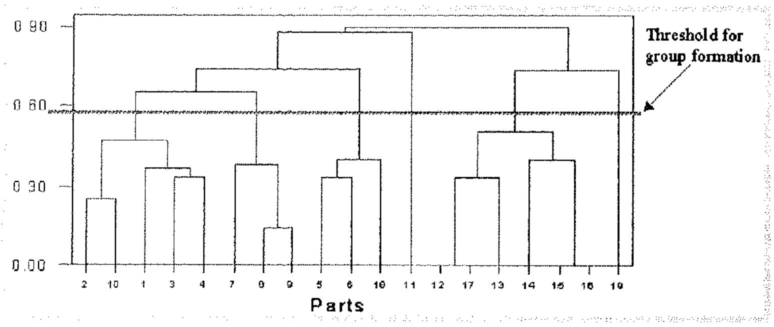


Figure 2. 5 - Hierarchical clustering of a machine - part matrix

In the cell formation problem, the similarity measure is calculated for each pair of machines, or each pair of components. Grouping of machines or components is based on a specified similarity (admission level). Any pair of components or machines, which has a similarity measure higher than the admission level and clusters, are merged when their similarity measure is higher than the admission level. By plotting the cluster versus admission level, a visual picture of the clustering process is revealed *(Figure 2.5).* To illustrate this clustering approach McAuley's use of the Jaccard similarity measure and single linkage cluster analysis will now be demonstrated.

The concept of similarity coefficients is based on the similarity between two machine cells. Thus, consider a set $P = \{x_1, x_2, \ldots, x_n\}$ of n machines which make up cell P, and a set $Q = \{y_1, y_2, \ldots, y_m\}$ of m machines which make up cell Q. Then the similarity coefficient S_{pq} between cells P and Q is given by:

$$S_{pq} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} x_i y_j}{\min(n,m)} \qquad \text{Equation 2. 4}$$

where: $x_i y_j = 1$ *if* $x_i = y_j$ *Equation 2. 5*

$x_i y_j = 0$ *if* $x_i \neq y_j$ *Equation 2. 6*

$$\min(n, m) = n \quad \textit{if} \quad n < m$$

$$\min(n, m) = m \quad \textit{if} \quad m < n$$

In order to illustrate the concept, consider the following cells:

1. cell C_1 consists of three machines $\{M_1, M_2, M_3\}$ required to manufacture part P_1,
2. cell C_2 consists of two machines $\{M_3, M_4\}$ required to manufacture part P_2,
3. cell C_3 consists of two machines $\{M_1, M_3\}$ required to manufacture part P_3, and
4. cell C_4 consists of two machines $\{M_4, M_5\}$ required to manufacture part P_4.

The cellular similarity coefficient between cells C_1 and C_2 is then given by:

$$S_{C_1C_2} = \frac{(M_1M_3+M_1M_4)+(M_2M_3+M_2M_4)+(M_3M_3+M_3M_4)}{min\,(3,2)} = \qquad \textit{Equation 2. 7}$$

$$= \frac{(0+0)+(0+0)+(1+0)}{2} = \frac{1}{2}$$

The numerator of the coefficient $S_{C_1C_2}$ indicates the number of common machines between cells C_1 and C_2, whilst the difference between the denominator (ie. 2) and the numerator, indicates the number of machines that have to be added to the larger cell C_1 in order to form a new cell to manufacture both parts P_1 and P_2. In this example, one machine has to be added to cell C_1 to form a new cell for parts P_1 and P_2. The similarity coefficient reflects the degree of similarity between two machine cells, and can be used as a basis for *CM* design. If two existing cells have a similarity coefficient close to one, then a smaller number of machines are needed in forming a new one from the two existing cells.

Seifoddini and Wolfe [1986] improved McAuley's method in three ways, which led to significant increases in computational complexity. They:

1. added a technique that duplicated machines to eliminate bottleneck parts,
2. employed an average linkage instead of the single linkage cluster analysis, and
3. employed techniques that reduced the amount of data storage required for processing.

Kusiak [1987] proposed a fourth measure. His measure of similarity between two machines j and l is calculated as follows:

$$\mu_{jl} = \sum_i \delta(S_{ij}, S_{il}) \qquad \textit{Equation 2. 8}$$

where:

μ_{jl} = similarity between part j and part l,

i = number of machines,

$\delta(S_{ij}, S_{il}) = 1$ if parts l and j both need machine i, or both do not need that machine, and

$\delta(S_{ij}, S_{il}) = 0$ otherwise.

The 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 far removed from each other in terms of similarity.

In the Kusiak [1992] paper, two new similarity measures were defined and incorporated in two heuristic algorithms (with and without bottleneck parts). The first similarity measure is applicable for the *CM* problem with basic and alternative process plans (where a block diagonal structure is embedded into the machine-part matrices). The second similarity measure generalises the first matrix and is applicable to non-decomposable matrices. The algorithms are illustrated with numerical examples and results are provided. Kusiak discussed two algorithms for solving the *CM* problem. One is suitable for the *CM* problem with basic and alternative process plans. The second algorithm is the clustering identification algorithm Kusiak [1987] applied to the similarity matrix, which showed that similarity coefficients have a significant effect on the quality of the clusters and in particular has the advantage of flexibility.

De Witte [1980] described the use of similarity coefficients in *PFA*, where process routines are the basic data from which interrelations between operations in a functionally structured production system may be found. He defined three new similarity coefficients used to measure the absolute relation between machine types, and the relative single interdependence of the machine-component types. A quantitative analysis of these interrelations permits operations to be grouped into cells in such a way that independents between cells fulfil certain conditions. One possibility for describing interrelations between operations is by using similarity coefficients, which determines the application of the specific cell structure.

Waghodekar and Sahu [1984] proposed a new cell formation technique called *MACE* (Machine-Component Cell Formation) which uses the Jaccard similarity. *MACE* consists of three outputs based on three different similarity coefficients as follows:

- The first similarity coefficient considered was the Jaccard coefficient.
- The second similarity coefficient was the product of two ratios *(SCR)*. Specifically, the first ratio was the number of components that visited both machine types divided by the total number of components that required the first machine type. The second ratio was the number of components that visited both machine types divided by the number of components that required the second machine type.

The third similarity coefficient was based on the total flow *(SCTF)* of common components processed by a pair of machine types, and the same two ratios *(SCR)* were needed for this similarity. This heuristic attempts to minimise the number of bottleneck parts without considering any possible constraints on the characteristics of the resultant cells (ie. the number of machines in each cell, and the total number of cells). It produces consistent results irrespective of the sequence of machines and parts in the initial incidence matrix.

2.2.4.2. Sorting-Based Algorithms

Another clustering technique has been developed by King [1980], called the rank order clustering *(ROC)* method, which is designed to generate diagonalised groupings of the matrix entries. Being more specific than the McCormick [1972] technique, it is therefore better suited to the problem of group analysis in *PFA*. The *ROC* algorithm operates as follows:

- For each row of the machine-component matrix in turn, read the pattern of cell entries as a binary word. Rank the rows in order of decreasing binary value. Rows with the same value should arbitrarily be ranked in the same order in which they are in their current matrix.

The *ROC* algorithm rearranges rows and columns in an iterative manner that will, ultimately, in a finite number of steps, produce a matrix in which both columns and rows are arranged in order of decreasing value when read as binary words. With the *PFA* solution providing an initial matrix input to the *ROC* algorithm, King [1980] generated a different solution in which there were five machine-component groups and four exceptional elements. This is because *ROC* is a ranking and not an optimising procedure, and is determined by the form of the initial matrix. However, this *ROC* algorithm has two major limitations:

1. 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, and
2. efficient implementation is not possible for very large problems, because the sorting procedure has a complexity of cubic order.

In 1984 Waghodekar showed that the *ROC* method may not produce consistent results when the rows or columns in the initial matrix were interchanged. The *ROC2* algorithm, developed by King [1982], improves on the original *ROC* by applying a quicker sorting procedure (in order to avoid extensive storage and handling of very large matrices).

***2.2.4.3. The Bond Energy Method* (BEA)**

A general cluster analysis technique, called the bond energy algorithm, has been developed by McCormick [1972]. He maximises the sum of all products of nearest-neighbour elements in the permuted matrix. Disconnected machine cells are formed after two iterations; the first finds the optimal column permutation, the second finds the optimal row permutation. As far as it is known, there have been no previous reports of this method having been used in connection with the problem of *PFA*. A bond is said to exist between each pair of adjoining row and column elements in the machine-component matrix and this creates a "bond energy", which is defined by the product of the values of the adjoining pair of elements.

McCormick's *BEA* is a sub-optimal heuristic procedure which provides satisfactory solutions for a variety of clustering problems. The *BEA* algorithm is:

1. *Place one of the columns arbitrarily. Set i =1.*
2. *Try placing individually each of the N-i columns in each of the i + 1 possible positions, and compute*

each column's contribution to the total bond energy. Place r the column that gives the largest incremental contribution to the total bond energy in its best location. Increment i by 1 and repeat until i= N (where N is the total number of columns).

3. *When all the columns have been placed, repeat the procedure on the rows (replacing N in the above with M , the total number of rows).*

The *BEA* is generally applicable to clustering problems, including those in which there may be more than the two types of matrix entry normally considered for grouping purposes.

2.2.5. Network Flow

The objective of the network flow methodology developed in the Lee [1967] article, is to measure the functional similarity between machines and then to group the machines into cells in such a way that all the parts in each family can be processed in a machine cell. This solution to the *CM* problem is obtained in terms of one complete-loop and several sub-loops, identified by using a relaxation method for solving the network flow problem. The objective of the article is to formulate a capacitated network model to generate optimal machine cells and part families in *CM* problems, without specifying the number of clusters in advance. The flow distance function for machine dissimilarity is used as the length of the arcs of the network. Each arc length can in turn be interpreted as the per-unit flow costs when one unit of flow is sent from the source to the terminal nodes. Machine cells are identified, based on closed loops, after applying the relaxation method for solving the minimal cost flow problem. Part families are then formed using the information given by the machine-part matrix.

The advantage of this network model in cluster analysis is its extremely efficient computational performance. The approach was found appropriate for solving large-scale *CM* problems including up to 500 machines and several thousands of parts using a minicomputer. A disadvantage though of this approach is that it only deals with the machine-matrix without considering the operational sequences.

2.2.6. Cost - Based Methods

Another approach is to consider that the cost performance of a *CM* system is directly dependent upon the manufactured product mix. In this approach, the issue of robustness of a cellular design to product demand changes, has been addressed by only a few research studies.

Rajamani [1992] considered the cell formation in a manufacturing environment where there are significant sequence dependent setup times and costs. The trade-off that exists between the savings on sequence dependent setup costs and additional investment on new machines was identified and explicitly modelled. The model developed was a mixed integer program.

The cell formation approach proposed by Seiffodini [1990] considers the random nature of the product mix by assigning probabilities to discrete product mixes and to the associated machine-component incidence matrix. For each product mix under consideration, a *CF* is determined, and the intercell material handling costs that correspond to this configuration, under all possible product mix\mixes, are calculated. Subsequently, the expected inter-cell material handling cost for each configuration is evaluated and used to select a near optimal solution. This approach does not take into account shop characteristics that affect the probabilities assigned to each product mix, such as resource capacity constraints. Furthermore, the fact that the optimum shop partition is calculated for every product mix limits the number of alternative states to be considered. Minimisation of the expected material handling cost has also been the target of related research conducted by Rosenblatt, et al [1992], who considered the stochastic nature of the product demand in their study of plant layout.

Harhalakis [1994] focuses on the cell formation under random product demand and presents an approach to obtain robust shop decomposition (cellular designs with satisfactory performance over a certain range of demand variation). The statistical characteristics of the independent demand, as well as the capacity of the system resources, are explicitly considered. The design objective is to minimise the expected intercell material handling traffic, a measure originally introduced by Seiffodini [1990]. In 1994 Harhalakis presented a two-stage design approach. In the first stage the statistics of the feasible production volumes are determined given the statistics of the independent demand, and are used to compute the design criterion for that shop configuration. This design stage provides the link between the forecasted market demand, and the feasible production volumes, upon which the cell formation process should be based. In the second stage a near optimal cell formation is determined using a grouping methods also presented by Harhalakis [1990].

2.2.7. Mathematical (Integer) Programming Formulation

Robinson [1986] addressed the formulation of families of parts, which required similar manufacturing processes with closely related machines, employing a Flexible Manufacturing System *(FMS)* using the topological concepts of polyhedral dynamics (ie. q-analysis). Here a qualitative grouping of parts was accomplished through the application of polyhedral dynamic concepts, in particular the q -analysis, and showed that the use of the q -analysis may provide:

- suggested part groupings based upon similar processing requirements,
- a measure of the integration of the parts into the overall manufacturing process,
- the strength of the interconnections between parts groups,
- the resilience of the *FMS* to change in the availability of parts and the potential of machines to become bottlenecks, and
- a measure of *FMS* complexity resulting in a qualitative understanding of manufacturing process stability.

In the Kusiak [1987] paper the relationship between the matrix model, the p - median model, and the classical group technology concept, was discussed. A generalised group technology concept, based on the generation for one part of a number of different process plan, was presented. A corresponding integer-programming model was formulated, and illustrated with examples. The author suggested that the programming models, which were developed, provided more convenient representation of the clustering problem than the matrix model. However, this formulation seeks to maximise the total sum of similarity measurements between all possible pairs of objects for a specified number of groups, under the condition that each object is assigned to one cluster. Additionally, it is assumed that the number of groups' (p) is known in advance. Evidently, this is not a very natural assumption, since in reality the number of clusters is unknown. Moreover, this method is not appropriate for large-scale applications, because it requires excessive computer time and memory for a formulation of constraints in terms of decision variables for product groupings.

Vakharia et al. [1993] has studied the impact of demand changes on the performance of *CM* systems. They recognised that resource capacities limit the ability of a cellular system to adequately respond to such changes, and proposed a system redesign methodology to address these robustness issues. Their method is based on a zero-one mathematical programming formulation and attempts to allocate new parts, or reallocate those for which large demand changes have occurred. It is noted that the manufacturing cells remain unchanged in composition and, therefore, system robustness is not addressed at the shop design stage. This technique may be viewed as complementary to a robust shop design methodology.

2.2.8. Graph Theoretical Approach

To improve the method of clustering, Rajagopalan and Batra [1975] have developed a "graph-theoretic approach" (graph theory terminology is presented in Appendix A). In this paper, a method for designing a cell is presented, which is based on the use of similarity coefficients, taking into account that some machine types can be allocated to several cells. Three similarity coefficients are introduced: one based on the absolute interdependence between machine types, the second based on relative mutual interdependence, and the third on single interdependence. The clustering was based on the graph-theoretic approach.

This method, so far, has been used in designing a cellular manufacturing system in such a way that for every component there is at least one cell in which the component can be fully processed. The method shows two important characteristics: that it is possible to analyse a greater number of routines and machine types, and it can be used even if the relations between machine types are fuzzy. The vertices of this graph are the machines, and the arcs Jaccard similarity coefficients. Here a clique (means of classifications) is a maximal collection of vertices, every pair of which is connected by an edge of the graph. The main disadvantage of this approach is that because of the high density of the graph, a very large number of cliques are usually involved and many of the cliques are not vertex jointed.

Vannelli, [1986] considered in his paper the problem of finding the minimal number of bottleneck machines and/or parts while disconnecting the operation into several groups. The problem is first shown to be equivalent to finding the minimal cut-nodes (the cut-node algorithm is presented in Appendix B) of a graph while disconnecting the graph into m subgraphs, each having at most k nodes. A dynamic programming approach reported by Lee [1979] is then implemented and extended to find the bottleneck cells. This algorithm is a heuristic for finding minimal cut-nodes of a graph, given that the designer specifies the number of disconnected groups desired, an upper bound on the number of nodes in each group, and a starting node in each group. The interactive capability of this approach allows a variety of good groupings to be considered. The intention is to give the designer complete flexibility in the design and analysis of systems.

2.3. Cellular Layout

The cellular layout approach has been recognised as an effective means of enhancing the efficiency of a discrete parts manufacturing shop. The design problem of a cellular layout is formulated in two distinct stages:

- Allocation of the machine cells to areas within the shop-floor (intercell or facility layout).
- Layout of the machines within each cell (intracell or machine layout).

This section reviews the appropriate work in both areas of the facility and machine layout problems. Various formulations, algorithms, and heuristics for these problems are presented. The facility layout and machine layout problems seek the best arrangement and configuration of facilities and machines, respectively.

Although both facility and machine layout are location problems, the facility layout problem lacks a flow structure. Furthermore, a major disadvantage of most existing formulations of the cell layout problems is that they require the location of sites (to which the facilities are to be assigned) be a known priory by the designer. However, since both are location problems and belong to a class of Non Polynomial *(NP)* time solved complete problems, it is important to review the past and present solution techniques of the facility layout problem in order to understand the fundamentals of layout problems.

2.3.1. Facility Layout Problem

The facility layout problem in *CMS* hasn't captured researchers' attention as much as cell formation in the past two decades. The assignment of facilities to discrete locations is a combinatorial problem that has been of interest to some earlier investigators [Armour 1963, Gavett 1966]. Some optimal-producing procedures have been presented [Bazaara 1983, Lawler 1963]. However, the only work for small problems (number of facilities less than 25) has yielded a number of interesting heuristic procedures proposed by Armour [1963], Buffa [1964], Hiller [1966] and Lee [1967].

Facility layout problems have been of interest to researchers of different disciplines since the early 1960's. Solution techniques to address such problems come from many differing fields. Furthermore, different names have been applied to these problems in the literature; Muther [1961] prefers "Layout Planning", Buffa [1964] uses "Facilities Allocation" while Hiller [1963] and others [Apple 1963, Lee 1967] prefer "Plant Layout". The word "facility" sometimes is use interchangeably for "office", "plant", or "department". The layout problem, however, plays an important role in shop floor design. A poorly designed layout will result in poor productivity, increased work in progress, disordered material handling, and so on. Only a few researches have dealt with this comprehensive subject.

2.3.1.1. Traditional Layout Design Processes

Traditionally, the facility layout problem is referred to as plant layout. The research field of layout has predominantly related to qualitative approaches and depended heavily on the subjective use of checklists, motion economy principles, and rules of thumb. In one of the earlier texts of layout planning, Immer [1950] discussed the basic steps involved in the analysis and improvement of an existing layout. The main idea is that the system productivity can be increased by gradually rearranging the existing machine lines to comply with the desired material flow.

In 1973, Muther developed a layout procedure named systematic layout planning *(SLP)*. *SLP (Figure 2.6)* has been recognised as a milestone in traditional facilities layout methodology. In *SLP*, block layout alternatives are systematically produced on the systems material flows (from-to chart) and activity relationships (activity relationship chart). Space requirements and practical limitations are considered in the design process to generate satisfactory solutions. The *SLP* method has difficulties in defining the adjacency relationship (definition presented in Appendix A) and in converting the space relationships and diagrams into feasible block layout alternatives. Nevertheless, *SLP* provides an overall guideline for developing subsequent methods of traditional layout design.

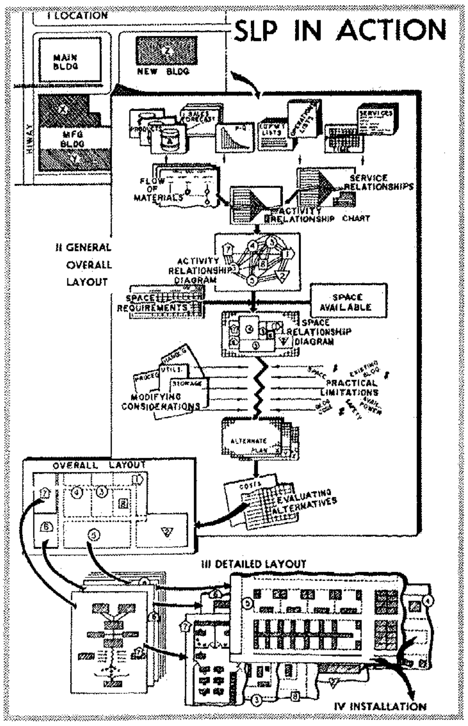


Figure 2. 6 – Systematic layout planning [Muther 1975]

In 1977, Apple summarised the facilities layout design methods and proposed a sequential design procedure. Apple's procedure recommends a set of detailed design steps, including relevant issues to be considered at each step. Following these procedures, one can systematically generate layout alternatives from the data collection stage, to the end of the design implementation phase. His method emphasises the importance of designing an effective flow pattern for the productive system and uses the planned flow pattern as an overall guide for the entire layout design process.

These traditional layout design methods have underlined the basic design process of facilities layout planning. These consists of six steps:

- defining the problem; analysing the problem; generating alternative designs; evaluating the alternatives; selecting the preferred design and implementing the design.

Virtually all contemporary layout planning methods are based on these steps suggested by Muther and Apple. Although quantitative approaches have recently played a more dominate role in developing facilities layout design methods, the systematic structure of the design process, the flow analysis (from-to-chart), and the adjacency analysis (relationship chart) are still the foundations of the layout planning tools.

2.3.1.2. Quadratic Assignment Problem (QAP) Model

The problem of assigning new facilities to sites or location when there is an interchange between new facilities, is refered to as a quadratic assignment problem *(QAP). QAP*, which is a problem in combinatorial optimisation, was first introduced by Kopmanns and Beckmann [1957]. The problem involves assigning n facilities to m locations so that two total cost matrices can be formatted. One matrix gives the magnitude of the flow of materials between any two facilities, and the other specifies the distance between any two locations. Detailed further discussion about the *QAP* model, together with solution methodologies such as: exacts, integer-linear programming, heuristics [steepest-descent pairwise-interchange method (*SDPI*)] are presented within Appendix C.

2.3.1.3. Computerised Layout Planning Techniques

Computer programs for determining layouts generally fall into two classes:

- construction algorithms and improvement algorithms.

a) Construction Algorithms

A construction algorithm constructs the layout from flow data and the information in the activity relation chart. Construction algorithms do not require the user to specify an initial layout and usually generates a layout from scratch. Because these algorithms construct layouts based on the closeness ratings given by the *REL* (relationship) chart, they are sometimes referred to as qualitative layout algorithms. Some of the computerised, construction algorithms are *PLANET* [Apple 1972], *CORELAP* [Lee 1967] and *ALDEP* [Seehof 1967].

ALDEP

Seehof [1967] of the *IBM* Corporation developed *ALDEP* (Automated Layout Design Planning). *ALDEP* makes use of the closeness ratings that appear in the activity *REL* chart. *ALDEP* first selects a department at random and places it in the upper left-hand corner of the layout. A department with a high closeness rating (*A* or *E*) is then placed in the layout adjacent to the first department. Successive departments are placed in the existing area in a top down fashion, following a "sweep" pattern. The process is continued until all the departments have been placed. The performance score for each layout is then computed based on a numerical scale attached to the closeness ratings. Thus, the recommended layout has the highest closeness rating score. However, a drawback to this method of scoring is that the *ALDEP*-designed layouts can have very irregularly shaped departments. Such configurations tend to be impractical from an operational viewpoint, as well as expensive to construct if walls are to be used to separate departments. Another limitation of *ALDEP* is the assessment of the numerical values assigned to the closeness ratings for each department. This method is very subjective, since no two layout analyses will produce the same *REL* chart.

CORELAP

Similar to *ALDEP, CORELAP* (Computerised Relationship Layout Planning) [Lee 1967] is a construction routine that also places departments within the layout, using closeness rating codes. The major difference between *CORELAP* and *ALDEP* is that *CORELAP* does not randomly select the first department to be placed. Rather, a total closeness rating *(TCR)* based on numerical values, is computed for each department and compared. Although *CORELAP* requires no initial layout and does not randomly select an initial layout as compared to *ALDEP*, it is nevertheless a qualitative heuristic method, since assigning closeness ratings to departments is again quite subjective. Neither *CORELAP* nor *ALDEP* considers the actual material flows between facilities in their selection for placement.

PLANET

The layout program *PLANET* provides three selection rules to determine the order in which facilities enter the proposed layout. All these rules are based on the cost of unit flow between each pair of facilities and a set of "placement priorities" specified by the user. However, the flow sequence is not taken into account in the ordering process.

The basic differences between these aforementioned construction methods lie in the selection rules with which the sequence of facilities' entering the layout design is determined. Because of their sequential and non-backtracking nature, construction methods usually require less computing effort and their solution quality is not as competitive as the other methods (eg. *SLP*).

b) Improvement Algorithms

Improvement algorithms seek a best layout arrangement from an initial layout considering the effect of interchanging the location of two or more facilities. They are generally preferred to construction routines algorithms because these often result in departments with odd shapes. Furthermore, they are not as subjective as the construction algorithms. Whereas the construction algorithms are based on an active relationship chart as the input, improvement algorithms facilitates the design of layouts based on flow considerations. Due to this major distinction, improvement algorithms are sometime referred to as quantitative layout programs.

CRAFT (Computerised Relative Allocation of Facilities Technique)

CRAFT is one of the oldest computerised layout programs as well as one of the most publicised. It was developed by Armour and Buffa [1963] and subsequently tested, refined, and applied by Buffa [1964]. The primary objective used in *CRAFT* is to minimise the total transportation cost of the layout, where transportation cost is defined as the product of the cost to move loads per unit distance from department *i* to department *j*, and the distance between departments *i* and *j*. To be more specific, the total cost of the product flow between all pairs of departments is:

$$TC = \sum_{i=1}^{n} \sum_{j=1}^{n} u_{ij} \, v_{ij} \, d_{ij} \qquad \textit{Equation 2. 9}$$

where: n - Number of departments,

v_{ij} - Flow-number of loads moved from department *i* to department *j* in a given time,

u_{ij} - Cost-material handling cost to move a unit distance from department *i* to *j*, and

d_{ij} - Distance separating departments *i* and *j*.

The input information v_{ij}, u_{ij} and d_{ij} may be represented as *n*-by-*n*- matrices (the from-to-matrices). *CRAFT* assume departments to be either rectangularly shaped or composed of rectangular pieces. Furthermore, departments are assumed to be located at their area centroids. The centroid is another term for the coordinates of a centre of gravity or balance point. The accuracy of a department, assumed located at its centroid, depends upon the shape of the department. The assumption is most accurate when the shape of the department is square or rectangular, but is less accurate for oddly shaped departments. The distance between departments is assumed as the rectilinear distance between centroid locations. For example, with two centroids, (X_A, Y_A) and (X_B, Y_B) Hakimi [1964] proposed the rectilinear distance to be calculated as:

$$DIST_{(A \text{ to } B)} = |X_A - X_B| + |Y_A - Y_B| \qquad \textit{Equation 2. 10}$$

Although *CRAFT* is considered one of the most publicised public domain, quantitative layout programs, it draws some criticism in the literature. The criticisms include the following:

- *CRAFT's* pairwise interchange heuristic does not guarantee that the least-cost layout will be found, since all possible interchanges (three, four or more at a time) are not considered.
- The *CRAFT* solution is path dependent. That is, the final layout is highly dependent on the initial layout supplied. In order to get an unbiased final layout, the layout analysis should specify a total of *n!* possible initial layout arrangements of *n* facilities, choosing the best final solution generated. This is definitely impractical.
- The method of centroids becomes less accurate or valid as the shape of the department becomes more irregular. Such an assumption might affect the outcome of the solution when departments are large. For the machine layout problem, the shapes of machines cannot be altered.

In spite of its drawbacks, *CRAFT* still is an excellent design or benchmark solution for comparison with other solutions. It also should be kept in mind that the major contribution of computerised approaches is the search phase of the layout process, not the selection phases. If the layout analyst is aware of this, the *CRAFT* layout program can serve a very useful purpose. This model is capable of handling a problem of up to 40 departments or machines, but does not guarantee an optimum solution.

Both *SPACECRAFT* [Johnson 1982], which is designed to produce layouts for multistorey structures, and *COFAD* (Computerised Facility Design) [Tompkins 1984], which incorporates the choice of a material handling system as part of the decision process, are an extension and modification, respectively, of the original *CRAFT* program. However, they share the same advantages and disadvantages employed in *CRAFT*.

Both the constructive and improvement algorithms are heuristic and they do not guarantee an optimal solution. They can be used for solving problems that are too large to be solved exactly (ie. greater than 50 machines). One of the major differences between the improvement and constructive algorithms is that the former has the disadvantage of requiring the initial layout to be specified. However, improvement algorithms generally result in more useable layouts, since construction algorithms often give layouts with oddly shaped departments.

2.3.1.4. Graph Theoretics Formulation

A number of researchers have used graph theory to solve facility layout design formulations (terminology and definition are presented in the Appendix A). Probably the first suggestion that the mathematics of graphs could be useful in layout planning was by Levin [1964], who proposed one of the first adjacency-based layout approaches based on graph theory. Adjacency graphs have three major shortcomings for use as the basis for a design skeleton when material flows are major factors. They are as follows:

1. Adjacency graphs are limited to consideration for only those flows between adjacent cells. That is, some of the desired adjacencies must be abandoned in order to meet the planarity condition and to obtain a solution.
2. The flow networks of the actual material flow paths are ignored by the adjacency graphs.
3. The locations of facilities resulting from adjacency graphs are quite subjective, that is, if a department A is adjacent to department B, a precise location of department B from department A cannot be captured.

a) Planar Graph Approach

Graphs of special topological characters called planar and dual graphs are of particular interest to facility layout researchers [Carrie 1975, Giffin 1986]. By definition, a graph is said to be planar if it can be mapped onto a plane such that no two edges intersect.

Furthermore, if a graph contains the maximum number of edges without loosing planarity, then it is a maximal planar graph (*MPG*). That is, a maximal planar graph is a planar graph to which no other edges may be added without the graph becoming non-planar. Each face (where a face is the area bounded by a cycle of edges) of a *MPG* is bounded by exactly three edges. Thus, Busackeer [1965] has shown that a planar graph is maximal if, and only if, it is triangular, ie. all its faces, including the infinite face (a region outside the graph) are bounded by three edges. He has also shown that every planar graph is a subgraph of a triangular graph having the same number of vertices, and established two significant relationships as follows:

$$e_{max} = 3n - 6 \qquad \textit{Equation 2. 11}$$

$$f_{max} = 2n - 4 \qquad \textit{Equation 2. 12}$$

where: e_{max} - the maximum number of edges in a *MPG*,

f_{max} - the maximum number of faces, in a *MPG*, and

n - the number of vertices in a graph.

These two upper-bound equations indicate that the number of nodes has to be greater than or equal to three, and the number of faces is always even. A graph is planar only if it has a "dual" solution. This indicates that if a graph G is planar, than there exists an associated graph G', such that it is dual. Conversely, a dual graph is also a planar graph. The dual graph G' is constructed by placing a point in each face of the planar graph G, including the infinite face, and connecting the two points whose faces have a common edge. The points become the vertices of the dual and the connecting lines becomes its edges. The number of edges in a planar graph and its dual are equal, whereas the number of faces and vertices are interchangeable.

Because of its characteristics the planar graph can be mapped onto a 2-dimensional facility layout problem. In a single floor layout, the block plan of the facilities' arrangement can be represented as a "layout graph". Each point on the block plan at which three or more facilities meet, represents a vertex in the layout graph. One of these facilities may be the exterior of the layout. The boundaries of the facilities connecting these points are the edges of the graph, and the facilities are its faces. The layout graph is always planar since in the block-plan facility boundaries do not overlap, and thus the edges of the layout graph intersects only at its vertices.

Facilities that are expressed by the *REL* chart can also be represented as a connected weighted graph, where the vertices and edges of the graph represent the facilities and their relationships, respectively. The graph is undirected because of the symmetry of the relationship chart. It is also weighted since each edge has associated with it one of the ratings: *A, E, I, O, U*, and *X*. Since the primary objective of the facility layout problem is to maximise the adjacencies achieved in terms of weight and number, the layout analyst can apply the Maximal Planar Weighted Graph *(MPWG)* concept. The formulation of developing a *MPWG* is as follows [Nozari 1981], with the objective:

$$\textit{Maximize} \qquad Z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} x_{ij} \qquad \textit{Equation 2. 13}$$

subject to

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} z_{ij} = 3n - 6, \qquad \textit{Equation 2. 14}$$

$$x_{ij} = \{0, 1\} \qquad \forall \quad i, j \qquad \text{Equation 2. 15}$$

where: z_{ij} = the weight of the relationship between facilities *i* and *j*, and

$$x_{ij} = \begin{cases} 1, & \text{if facilities } i \text{ and } j \text{ are adjacent} \\ 0, & \text{otherwise} \end{cases} .$$

Different subsets of elements of the relationship chart may be selected for the formation of the *MPWG*, thereby resulting in a number of graphs with different weights. A number of edge combinations equal to:

$$_{n(n-1)/2}C_{3n-6} = \binom{n(n-1)/2}{3n-6} \qquad \text{Equation 2. 16}$$

can be evaluated and tested for planarity in order to obtain an optimal solution. The possible number of tests for planarity is a combination, instead of a permutation, because the planar graph contains edges not arcs. That is, the flow from node *i* to node *j* is the same as the flow from node *j* to node *i*. However, as for the machine layout problems, this statement is false, since the number of parts flow from machine *A* to machine *B* is not necessarily the same as the number of parts flow from machine *B* to machine *A*. The expression *n(n-1)/2* indicates that the flow matrix is a symmetric matrix, and *3n-6* is the upper bound for the number of edges allowed.

In summary, there are three steps for developing a block layout representing a facility layout arrangement by the planar graph concept:

1. developing a *MPWG* from the facility relationship to indicate which facilities will be adjacent,
2. constructing the dual graph of the *MPWG* to represent facilities as adjacent regions having specific boundaries, and
3. converting the dual graph into a block layout where the facilities have a regular shape with specific areas.

Developing an optimal *MPWG*, however, is a difficult combinatorial problem, which has been reported to be *NP*-complete (Non Polynomial time solvable-complete problem) [Foulds 1983]. Optimal solutions can be obtained by branch and bound method. However, this solution technique requires an excessive amount of *CPU* time when the number of nodes is greater than 15. Hence, interest has turned to developing heuristic solution techniques for the problem.

b) Graph Heuristic Solution Methods

Testing a graph for planarity is the first problem that has to be solved before the planar graph concept can be applied to the facility layout problem. Unfortunately, this testing is difficult because of its *NP*-complete nature [Giffin 1986]. A substantial advance has been made by Hopcroft and Tarjan [1974], who developed a polynomial-time (linear-time) planarity testing algorithm. Since Hopcroft-Tarjan's algorithm is difficult to implement, recent heuristics have been developed to avoid planarity testing.

Deltahedron Heuristic

It can be shown that the faces or regions of a *MPG* are all triangles, including the exterior face. Therefore, a *MPG* is equivalent to a polyhedron with all faces triangular. Such a polyhedron is sometimes called a deltahedron, hence the name of the method. The deltahedron method was developed by Foulds and Robinson [1978]. It has the advantage of being easy to follow compared to the Hopcroft-Tarjan algorithm [1974]. The deltahedron heuristic solution procedure builds up a *MPG*, one vertex at a time, for implementation of the graph planarity test. The algorithm for the deltahedron heuristic is as follows:

1. *Given a travel chart (a from-to matrix), for each column, (i), sum up all values, (w_{ij}), in rows, (j). That is, for each vertex i, calculate a value M(i), where:*

$$M(i) = \sum_{i=1}^{n} w_{ij} \qquad \text{Equation 2. 17}$$

2. *Arrange the vertices in order of increasing M values. Connections are broken arbitrarily. The first four vertices are chosen.*
3. *Choose the edge of highest weight, say, e(a, b). Choose vertex c so that the triangle abc has the highest weight. Choose vertex d so that the induced complete subgraph on a, b, c, d has the highest weight.*
4. *Form a complete graph from these initial four vertices, a, b, c, and d. This graph is called a tetrahedron.*
5. *Insert a new vertex i to the triangle that causes the largest increase in the weight of the graph. (Note that the tetrahedron graph has four triangles, vertices abc, acd, and bcd.)*
6. *Repeat step (5) until no more vertices remain.*

Such approaches do not specify location aspect and the flow information of the layout. It only provides adjacencies amongst facilities.

String Processing Heuristic

Another procedure which avoids planarity testing is String Processing *(SP)* [Moore 1976]. This heuristic solution method is similar to the above deltahedal procedure, except that *SP* starts with a maximum spanning tree (presented within Appendix B) as the initial graph instead of the tetrahedral graph. The *SP* heuristic method depends on representing and processing a graph as a string of symbols that denotes its vertices. The string is manipulated according to a set of rules that results in partitioning it into substrings equivalent to subgraphs of the sought graph. Each partition is equivalent to inserting an edge between two vertices. The partitioning of the string is repeated to create a triangular face. In order to guarantee planarity, each edge should appear in exactly two of the substrings representing the triangular faces. Procedures that rely on *SP* start with a string that represents a maximum spanning tree, then manipulates it to create a *MPWG*. Rules for determining the order of partitioning should be such that it aids in maximising the total weight of the selected edges. A drawback to this heuristic solution technique, however, is that it is not very applicable to the machine layout problems, since its structure lacks both the flow and location aspects, which are essential to the machine layout problems.

c) Maximum Spanning Tree Approach

The use of a maximum spanning tree as a design skeleton for the facility layout problem has been of interest to a number of layout researchers [Carrie 1974, Moore 1975]. Carrie [1974] first proposed such an approach to the problem. These researchers suggested using the maximum spanning tree based on the direct flows between the cells. The maximum spanning tree has the advantage that it can be efficiently determined (even without a computer). It is a spanning tree whose sum of edge weights is maximal. Seeking the most important paths of flow, it is a well established concept, and several algorithms have been published which utilise maximum spanning trees. However, it does not provide the layout analysis from any location information and more importantly, it is similar to the adjacency graph approaches discussed previously, in that it does not consider all the flows in arriving at a design skeleton. In addition, and most importantly of all, because the maximum spanning tree has edges instead of arcs, it cannot capture the flow direction, which is important to the machine layout problem. From this, it is clear that an undirected graph such as the maximum spanning tree cannot capture flow direction. If the concept of the flow structure can be captured, the maximum spanning tree can be a useful solution technique to solve the machine layout problem. A maximum spanning tree with direction is called a maximum directed spanning tree.

2.3.2. Machine Layout Problems

Machine layout problems involves the arrangement of machines on a factory floor so that the total time required to transfer material between each pair of machines is minimised. As discussed in Section 2.1, both machine and facility layout problems seek the best arrangement and configuration of facilities and machines respectively. The major difference between these two location problems is that the facility layout problem lacks the flow structure, which is essential to solving the machine layout problem. A good machine layout design should have a well-defined flow structure and be able to capture the locations of the machines.

2.3.2.1. Heuristic Solution Methods

Researchers [Francis 1974, Kusiak 1985] have mentioned various machine layout configurations, but the actual creation of such layouts are not addressed in these references. Amongst different types of machine layout configurations, the straight-line flow is the simplest form of flow structure. Researchers such as Hillier [1963], Carrie [1975], Vakharia [1990], and Burbidge [1995] used heuristic solution methods to find the suboptimal flowline layout sequence. Hillier's heuristic algorithms minimised both backtracking and by-passing movements.

One of Carrie's approaches for avoiding the back track movement is to introduce as many machines of each type as are necessary to avoid back tracking altogether, but such an extreme case yields uneconomic work loads and costs on some machines. Vakharia designed *CM* systems eliminating back tracking moves. He grouped all machines with back tracks into one cell, and all machines with either in-sequences or by-pass routing into another, and then finds machine flowline sequences for these cells. The results did not provide maximum overall machine adjacencies and did not contribute to the overall shop layout.

Burbidge's line analysis plans the layout of the machines in a group and contributes to the planning of

FMS's. However, the layout is quite subjective, since the actual machine assignments were not based on material flow volume. Clearly, machine pairs with high flow between them should be positioned close to each other. To further minimise both back tracking and by-passing movements, alternative machine layout configurations such as *U, O,* and *L* shapes, other than the simple straight-line flow, should be considered. Such options were not considered by the researchers referenced in this section.

2.3.3. Multiple-Objective Layout Methods

The traditional, or static layout problem assumes that all data such as the number of departments, areas and flows are constant. However, conditions of companies are constantly changing, so most layout projects are redesigns rather than new facility developments "from scratch". Nicol [1983] concluded that "radical layout changes happen frequently and that management should therefore take this into account in their forward planning". The design of a facility can have a significant impact on the ease and cost of expansion. Tompkins and White [1984] recommend developing a flexible layout; one that can easily be modified.

Clearly a better approach to flexible layout design is to directly incorporate known future data changes into the layout problem. To do this, a series of layout problems are developed over a number of discrete time intervals. These discrete intervals can match a company's planning period, phases of project, or new product introductions. Discrete intervals also simplify data collection. For this scenario there is a series of single period or static problems, each with its own flow matrix and block diagram. An additional 'rearrangement cost' term in the objective function ties the static problems together. A solution to this Dynamic Layout Problem *(DLP)* minimises flow costs and rearrangement costs for a series of static layout problems. In a *DLP*, rearrangement costs are added whenever an area contains different departments in consecutive time periods. Rearrangement costs are in dollars per square meter and must include all the costs involved in a facilities project. If rearrangements are performed on off-shifts or weekends, wage premiums must be used to determine costs. One must also consider either the lost production cost or overtime costs to make up production on off-shifts.

There are two limiting cases to the *DLP*. Firstly, if the rearrangement costs are much higher than flow costs, then there would be no rearrangements, which allows one to combine all the flows and solve the problem as a static layout problem. Secondly, if the flow costs are much higher than rearrangement costs, then rearrangements are trivial, allowing one to solve a series of independent static layouts. The *DLP* is required when one must balance the trade-off between increase flow cost of inefficient layouts and added rearrangement costs.

Traditionally, facility layouts have been solved only for one goal, either qualitative or quantitative measurements of the layout. The most commonly used qualitative objectives are the adjacency requirements between facilities and safety regulations considerations. The quantitative goals primarily involve the minimisation of material handling costs, or the maximisation of a system's throughput.

Since the 1970's, several multi objective approaches have been proposed. These methods use weighting factors to bring individual goals, both qualitative and quantitative, into one optimisation scheme. In these methods, the multiple criteria are either combined as a single factor in the objective function, or treated as separate groups of factors. These separate groups of subgroups are considered hierarchically during the layout evaluation and construction. However, weighting factor methods have a serious shortcoming, since the determination of the proper numerical values for each weighting factor remains a subjective matter. To address this shortcoming, some researchers turn to treating secondary objectives as side constraints and solving the multi-goal problem dynamically. Some of the studies that use the aforementioned approaches are Jacobs 1987, Cambron 1991, and Shin 1992.

2.4. Artificial Intelligence Techniques (Modern Approaches)

Artificial Intelligence *(AI)* is a research field concerned primarily with emphasising theoretical problem solving, which is not necessarily applicable to the real situation. As mentioned before, one major branch of *AI* is Expert Systems *(ES)*. An *ES* employed in *CM* consists of facts that are stored in the knowledge base and heuristics that are utilised by an inference engine. A person who inserts the rules into an *ES* for a particular domain is called a "knowledge engineer". Such a person develops a system that uses similar knowledge and inference strategies to simulate the expert's behaviour.

2.4.1. Neural Networks

Neural networks, a recent development in artificial intelligence, is a distributed information processing system composed of many simple computational elements (nodes) interacting across weighted connections. Neural networks are analogous to the working of the brain and the nervous system of human beings. These networks can learn and adapt themselves with input from the actual processes. They achieve good performance with a high computation rate using their parallel processing feature and their ability to learn. The values of the weightings and the topology of the connections internally represent knowledge learnt. Learning involves modifying the connection weightings. Neural networks have proved effective at solving problems in a variety of areas, such as image processing and speech recognition. The application of neural networks to *CM* is relatively new and has recently attracted the attention of a few researchers [Moon 1990, Karparthi 1991, Venugopal 1992].

The most important advantage of the neural network in *CM* is its extremely efficient computational performance. This approach was found appropriate for solving large-scale cellular problems, but the disadvantage of using it in *CM* is that the order of the problem presentation affects the performance to a large degree.

2.4.2. Simulated Annealing *(SA)* Methods

A new technique called simulated annealing (will be explained in more details within Appendix C) has recently received much attention in solving combinatorial optimisation problems, and is based on classification and coding. The *SA* algorithm developed by Kirkapatrick [1983], is a general purpose combinatorial optimisation technique based on the annealing phenomena. *SA* methods have been successfully applied to solving combinatorial problems such as the *TSP* (Travelling Salesman Problem), the *QAP*, and the design of large-scale electronic integrated circuits etc. Although the simulated annealing approach does not guarantee an overall optimum solution in practice, it provides the means of moving away from the local optima occasionally during the search process.

The method of *SA* is directly applicable to those optimisation problems where the decision variables can be expressed as points in a discrete, *n*-dimensional space. Typical applications include the travelling salesman problem [Press 1988]. To apply the *SA* method; the solution-space, the appropriate procedure to vary the decision variables within this space, the initial 'melting' temperature *t* and the 'annealing schedule' (the manner in which the temperature *t* is to be reduced during the solution process), must be defined.

The Jajodia [1990] approach represents facilities by equidimensional square blocks and uses the Manhattan formula to compute the distance between entities. Then *SA* is employed to reconfigure the block assignment on the shop floor grid. These two simplifications do not consider the physical constraints of the problem and may lead to infeasible solutions. Jajodia, et al. [1992] compared *CRAFT*, *SA* algorithms, and other heuristic algorithms. They also arrived at the conclusion that the *SA* algorithm was better than the others, with respect to both solution quality and efficiency. Similar comparisons in the literature can be found in the work of Kouvelis and Chiang [1992], who also provided improved solutions by restricting entities with high material flow to be adjacent.

The heuristic method of Proth and Souilah [1992] represented facilities as rectangular blocks and considered the distance between facilities. They acknowledged physical constraints for intracell layout, but did not consider reassignment of the parts for the appropriate machines for further improvement and intercell layout. This last issue of the reassignment of parts is very important, and the physical constraints and part reassignment are fully considered in further developments of the proposed research methodology.

Alfa [1992] formulated a model for solving facility layout problems in *CMS's*. Alfa's model assumes that the location of the cells is predetermined. Hence, it only solves the intracell layout problems, utilising the modified penalty algorithm of Heragu and Kusiak [1991] to generate the initial configuration, with *SA* being then applied to improve this configuration. Das [1993] also formulated a model for solving intercell layout problems. His model does not consider intracell layout, however it does influence the final intercell layout.

Tam [1992] proposed a *SA* procedure to allocate space to manufacturing cells on a floor plan. His procedure takes into account the dimensions and shapes of individual cells as well as occupied areas on the floor plan. By developing a slicing tree (from the graph theory) as the guidance for cell relative locations, the *SA* procedure minimises the intercell flow whilst satisfying the various geometric constraints. However, the clustering process used in creating the slicing trees does not consider the flow directions. Although Tam emphasised that the geometric constraints (the area and shape requirements of each cell) vary according to the flow of parts through the cell, no attempt was given to specify efficient flow patterns for the system.

In 1994 Chen investigated the application of simulated annealing based algorithms to the machine cell formation problem, which partitions the set of machines into some (unknown) number of machine cells. This was done to maximise the total machine similarities within cells, subject to the cell size constraint, which restricts the number of machines in each cell. Two simulated annealing based algorithms were proposed and extensive computational tests were carried out to evaluate their performance with a graph partitioning based heuristic.

The Bazargan-Lari and Kaebernick papers [1996, 1997 and 1998] illustrated the *SA* process of developing the final intercell layout designs by providing layout configurations, showing the impact of each design on the material handling distances. In this work, the continuous layout model with the assumption that the departments are of predetermined geometrical shape and the actual dimensions of the shape can vary continuously, is used. The layout model is defined using only rectangular shaped departments/facilities which allow the consideration of material handling distances but not travelling costs, and proposes only two activities (move and swap) to generate neighbouring actions.

Wang et al. [1998] proposed *SA* algorithms for solving facility layouts in *CMS*. Their continuous model assumes only *U* shaped cells and does not consider costings in the material handling distance's objective function. However, they do not indicate how the cell layout is produced or defined (graphical presentation) and neither is any layout configuration presented.

According to an article in the August 1995 issue of *IIE* Solutions [1995] the LayOPT software package incorporates an algorithm for layout generation in addition to layout evaluation. LayOPT is produced by the Production Modelling Corporation and is an implementation of *MULTIPLE* [Bozer 1994] and *SABLE* [Bozer1995]. The attributes of this software package are comprehensively analysed in the article, and therefore the analysis will not be presented here.

MULTIPLE, a single or multi-floor improvement type algorithm developed by Bozer, Meller and Erlebacher [1994] uses a discrete representation, and extends *CRAFT* by applying space filling curves to single floor or multifloor facility layout problems. *MULTIPLE* improves upon *CRAFT* by increasing the number of exchanges considered at each iteration. In addition, *MULTIPLE* can restrict the irregularity of department shapes by using an irregularity measure, based on the perimeter and area of each department. However, since it uses a discrete representation, the department shapes may not be rectangular. *MULTIPLE*, like *CRAFT*, is a steepest-descent search and may be affected by the initial layout. *SABLE* [Bozer 1996] extends *MULTIPLE* by employing a simulated-annealing based search, and by generalising the department-exchange algorithm. *SABLE* is shown to produce lower cost layout solutions than *MULTIPLE*.

To date, no researcher has demonstrated any model that solves both intercell and intracell facility layout problems for *CMS*. The problem of shop floor layout can be considered analogous to that of chip placement on a microprocessor circuit board, since both involve the placement of manufacturing resources in a given two dimensional discrete space. This analogy will be exploited in the proposed research in order to generate efficient alternatives for the layout of a manufacturing shop. The machine and cell placement problem is similar, although they are addressed at different levels in the hierarchy of the manufacturing facility. Thus, they can be solved using an approach which will be described in Chapter 6.

2.4.3. Knowledge Based Expert System *(KBES)* Approach

These studies have shown that the detailed arrangement of facilities layout is subjective. The design factors such as multi-goals, closeness ratings, material flow pattern selections, and aisle designations are not easy to quantify. It has been reported that computerised methods do not always out perform visual methods [Black 1995, Scriabin 1985].

Unlike computerised facility planning methods, human designers have a special ability in visual pattern recognition and context-based reasoning. A number of researchers have developed methodologies which allow layout designers to play interactive roles in the computer-aided facilities layout process [Jacobs 1987, Banerjee 1992 and Webster 1992]. The latest generation of interactive approaches for facilities layout design uses optimisation techniques to generate design skeletons, from which the designer interactively produces satisfactory layouts with computer-aided expert systems [Montreul 1993 and Langevin 1994].

Applying *KBES* to facility layout problems has some drawbacks. Expert systems cannot reason from axioms or general theories, they do not learn, and thus they are limited to using the facts and heuristic's that they were "taught" by human experts. Their performance deteriorates rapidly when the problems extend beyond the narrow task they were designed to perform. From the present problem (layout) point of view, because it has been shown that it is a *NP*-complete problem, the rules developed in the knowledge base of the *ES* are difficult. To show that the *ES* facility layout methods operate with "reliable" rules, it would either have to compare with an exact procedure or with other heuristic solution methods.

Recent researchers that have applied *KBES* to solve the machine layout problems are Kusiak [1989] and Kusiak and Heragu [1990]. They developed two heuristic algorithms for the "Knowledge Based Machine Layout *(KBML)*", which can only solve optimally those problems in which the number of machines is less than four. For such a limitation, there is no need to use *KBML's* to solve machine layout problems. Their relationship matrix in the input data is redundant, since the elements in the flow matrix will quantitatively show a layout analysis to find what machines, with high volumes between them, should be located together. In addition, the way the *KBML's* rule selects the material handling system and type of machine layout is not convincing. For example, if a robot is used, then the circular layout should be implemented, or vice versa. If a cluster layout arrangement is used, then a robot is selected. Finally, a major drawback to *KBML* is that the input flow matrix is symmetric. From a machine layout point of view, this is not always true. For instance, a material flow volume from machine *i* to machine *j* might not be the same as flow from machine *j* to machine *i*. That is, the flow matrix for the machine layout problems will generally be asymmetric. In spite of these drawbacks, the *KBES* approach to machine layout problems is promising.

Black [1995] stated that the use of integrated manufacturing production systems is a strategy for the factory with a future. This is based on the linked *CMS* (eg. a linked *FMC* is presented in *Figure 2.7*), which provides for a continuous flow or smooth movement of materials through the plant. The linked *CMS* is the newest manufacturing system. It is composed of manufacturing cells linked by a pull system for material control. In the cells, operations and processes are grouped according to the sequence that is needed to make a group of products. This arrangement is much like that of the flow shop but is designed for flexibility.

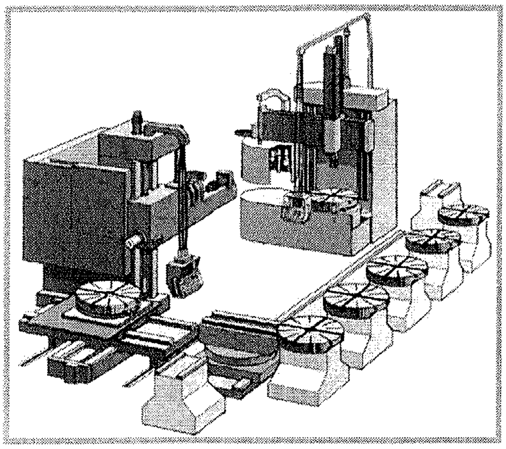


Figure 2. 7 – Flexible Manufacturing Cells (FMC) [Black 1995]

In 1992, Webster, et al. developed an interactive computer-based *FMS* design tool. Thirteen pre-specified layout types were stored in their program library. Based on a selected layout type, the developed design tool exhaustively searches the solutions space for non-inferior layouts, which suits the known part sequence requirements of the given manufacturing system. Their solution method was an improvement over an existing layout. No effort was made to seek optimisation of the layout design.

Although there are numerous references on general facility layout problems, the physical layout of machine cells in a modern manufacturing system has not been sufficiently studied. A cellular layout problem is relatively more complicated than a general layout problem. The difficulty of solving cellular layout problems stems from the fact that the location of each cell depends on the locations of other cells, as well as the geometric characteristics of these cells. Only a few papers have directly tackled this issue.

2.5. Justification Techniques

The basic problem is that many of the advantages of the new manufacturing technologies lie not in the area of cost reductions, which are always paramount, but rather in more "strategic" areas, such as reduction in inventory, throughput time, and space requirement on the factory floor, as well as simpler scheduling and better quality. Since these new manufacturing systems are largely equipment based, and manufacturing equipment has historically been justified on the basis of cost reduction or capacity expansion, these systems are expected to be justified on these same measures with the respect to layout methodologies.

When stand-alone systems are linked together into cells, such in *CM* or *FMS*, then an intermediate level of integration is achieved that exhibits an energy linkage between the independent systems. Of all these advanced manufacturing technologies, two characteristics make their justification process more complex than has been required in the past. Firstly, these technologies are much more flexible, in most cases reprogrammable, than equipment has ever been before. However, the advantage of this flexibility is not easily captured in simple economic justification procedures. The second characteristic of new technologies that requires special consideration is this aftermentioned interlinked energy. Users often report qualitative benefits from linked systems, such as faster response to customer requests, that are deemed far more important than normal cost savings.

There exists a number of formula and approaches that firms use for the economic justification of equipment. Examples include breakeven analysis, *DCF* (Discounted Cashflow) techniques, *MAPI* (Machinery and Allied Products Institute) methods, and other approaches such as incremental rate of return, accounting rate of return, net present value, *ROI* (Return on Investment), and payback.

Almost all literature available on the topic of justification of advanced manufacturing technology seems to converge on the failure of discounted cashflow (*DCF*) techniques to adequately consider strategic benefits of new manufacturing technology. Goldhar [1985] pointed out that traditional capital budgeting techniques are based on scale economics, cost-per-unit productivity criteria, and lack strategic evaluation capabilities. Gould [1982] argues that since a system consideration is necessary in judging capabilities of computer-based manufacturing systems, their performance should not be evaluated in a traditional way. Kaplan [1993] advocates that *DCF* is not applicable in the justification of new technology because improved quality and flexibility are not quantified. Whilst Primrose and Leonard [1986] presented a qualitative approach to the *FMS* justification problem.

Traditional justification procedures, which make sole use of financial criteria such as *ROI* and payback period, fail to justify manufacturing systems based on advanced technology because of the following reasons:

1. It is difficult to quantify long-term strategic benefits, such as improvements in quality, flexibility, delivery dependability of providing fast response to market needs, and valuable experience gained by the company in using advanced technology.
2. It is difficult to quantify the individual qualitative benefits created by the integration of several systems such as *CAD/CAM, FMS* and robotic systems.
3. Life cycles of advanced manufacturing systems outlive the products they produce, making it difficult to assess appropriate planning horizons for justification purposes.
4. Greater financial risks in terms of future costs, and organisational risks in terms of unanticipated

changes in the infrastructure of the company, are also difficult to assess. Deciding the appropriate level of new technology for an optimal combination with the existing traditional or new manufacturing technology also complicates the issue.

Chakravarty [1992] showed a framework for acquiring new technology as follows:

- Strategic evaluation – here consideration is given to: choice of the competitive strategy; specification of the market requirements; choice of the manufacturing system requirements; choice of the manufacturing system configuration; identification of the organisational constraints; iterative strategic evaluation (if needed).
- Operational evaluation - factors to consider are obtaining technical and operational data, and vendor information for alternative systems proposals; comparison and ranking of the alternative proposals; simulation studies of top three/four different operational scenarios; selection of two/three systems for financial evaluation.

2.6. Cost - Utilisation

Methodologies such as optimised production technology *(OPT)* and the theory of constraints *(TOC)* concentrate on the location of constraints and bottlenecks in the production process [Ronen and Starr 1990]. A constraint is defined as anything that prevents the system from achieving a better performance measure versus its goal [Goldratt 1990]. The cost-utilisation model developed by Borovits and Ein-Dor [1977] allows analysis of the utilisation of the resources in relation to their cost. The model was originally used as an economic tool to analyse computer systems hardware and designed to deal with deterministic systems.

Ronen and Spector [1992] in their paper presents a graphic model combining operational measures of performance for analysis and design of operational systems. The model combines two approaches, the Pareto and the theory of constraints *(TOC).* The Pareto approach concentrates on the important and costly elements of the organisation. *TOC* focuses on the organisation constraints. The model enables in-depth analysis, and at the same time it is easy to use. It allows the manager to identify problems and improve operating systems by:

- locating and classifying external, internal, and policy constraints; classifying the subsystems by their cost; identifying the right/desirable place for the constraint; analysing the present load structure of the system's components; suggesting a better load structure and positioning of subsystems within the system; performing a sensitive analysis concerning the effect of changes (investments in resources, change in product mix or market demand) on the system's structure and constraints; determining indices for comparison between alternative flows of operational systems; monitoring production trends; analysing statistical deviations in the system's load profile and assisting managers to achieve a global view of their system.

2.7. Computer Simulation

Computer simulation is the appropriate research methodology for operations with two independent resources (labour and machines) and in individual functions (cells). Simulation is also the only methodology that is robust enough to systematically examine the role and impact of product complexity and other essential variables on factory performance. Furthermore, simulation is compliant with the vast body of established research in evaluating *CM* performance. The evaluation criteria might be the determination of which station (a station can be a department for a facility layout problem, or machine for a machine layout problem) causes the bottleneck, or the utilisation for each station. Discrete event simulation is especially of interest to facility planners or to machine layout designers. The few researchers that applied simulation to facility layout problems are Zoller [1972] and Armour [1963]. Their input data is similar to that of *CRAFT*, discussed earlier. The main difference is that simulation is able to evaluate the system performance, and based on simulation output, the analysts can arrange appropriate facilities that can further minimise the travel time or to maximise the utilisation depending upon the objective.

Most of the literature, however, uses simulation as a tool to compare the performance levels between *CM* cells or *FMCs*, with traditional job shop manufacturing or functional layouts. Cummings [1980] compared work in progress (*WIP*), transit time, and lateness of jobs between *CM* cells and job shop layouts. The results showed that *CM* out-performs the functional layouts and that *WIP*, setup time, and transit time are reduced. Flynn and Jacobs [1986], also using simulation, comparing the performance of a functionally organised job shop with a factory using only manufacturing cells as defined by an actual firm. The initial *CM* layout, which they produced, was derived from *CRAFT*. Their simulation results showed that although *CM* exhibited superior performances in terms of average move time and average set-up time, the traditional job shops had superior performance in queue related variables (average queue length, average waiting time, *WIP* inventory, etc). The reason was the breaking-down of functional machines into many different cells increased the average delays in processing. One aspect of Flynn and Jacobs' work is that their initial *CRAFT* based layout does affect the outcome of the simulation output.

Another simulation case study on performance improvement of *CM* cells was done by Sassani [1990]. Sassani stated that product mix, product design, market situation, and other technical factors can affect the *CM* cells performance. One of the assumptions made was that the distances between machines could be neglected, however distance does effect the total travel time and thus costs. One should not assume that distance has no affect on the simulation outcome, simulation outputs are very layout dependent.

Simulators (*Figure 2.8* shows a constant material flow handling system model with pickup, transport and drop-off distance dependent operations) have been used to evaluate *CM* cells, but none have examined the hybrid industrial situation where cells manufacture less than 50% of the factory output, and the demand for products varies. Nisanci et al. [1981] provided an insight into the performance of non-cell work in their simulation of shoe operations. However, because the study was limited to an actual company, they were unable to vary part configurations and cell size. This research identified a need for better understanding of the total manufacturing operation because, although the cells showed performance gains, the non-cells functional areas showed performance losses.

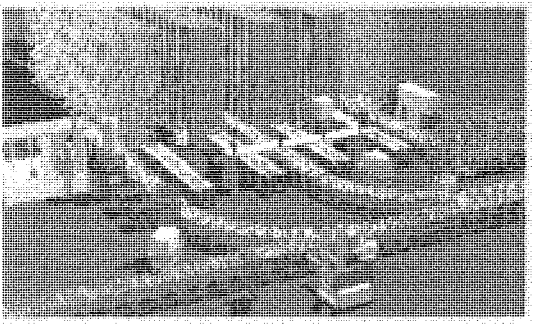


Figure 2. 8 - Witness simulation model

Computer simulation is an excellent tool to evaluate a system of interest. However, when one compares *CM* cells with other traditional systems, assumptions and initial layouts have to be carefully selected. One performance criterion that has not been addressed in the literature is that alternative machine layout configurations have a significant impact on the simulation results. A generalised solution technique to solve the layout problem using simulation is very difficult, since simulation is not a solution technique. In addition, finding all possible initial layout arrangements, which is one of the required input data, is unrealistic. This is true especially when the size of *n* (number of parts) increases. Furthermore, the output of the simulation is path dependent, (where the final layout obtained depends upon the initial specific layout) which is the same as *CRAFT*. As long as the layout analysts remember that computer simulation is used to evaluate a planning phase, then simulation can serve a very useful purpose.

2.8. Conclusions

One of the basic concepts of *CM* is to decompose manufacturing systems into subsystems. As a consequence of this, parts can be grouped into part families, where each part family corresponds to a group of machines called a machine cell. These concepts were originally proposed by Mitrofanov, and developed in the western world by Burbidge and others.

Several techniques have been developed to carry out the cell construction process, one of the earliest being Burbidge's *PFA* method. More recently, mathematical methods have been implemented, of which clustering analysis is the most commonly applied technique. An assumption underlying the use of clustering techniques is that homogenous clusters actually exist in the data. The basic approach of this technique is to construct algorithms and heuristics, which group machines into clusters based on certain attributes, such as processes. Two distinct types of clustering models have been used, namely, matrix representations and graph representations, the former being the most commonly used.

A number of integer programming clustering models have also been developed to allow for the setting of constraints on the number of cells, and the maximum number of parts allowed in each cell. The objective functions in these models are typically designed to minimise the travel distance between parts in the cell construction, where distance in this context refers to part disparity in terms of their association with like machines, or like processes. Many of these models have been shown to work well on small to moderate sized problems, for more realistic problems there are difficulties in defining a suitable distance matrix. However, more importantly, problems occur with increased computational complexity, since the number of variables and constraints are each typically of the order of *n,* where *n* is the number of parts.

Unfortunately, binary data that is often used to represent the machine-part relationships in matrix form does not capture information on intracell or intercell travel distances, flow directions, flow volumes, machine locations and feasible material handling system configurations. Thus, the diagonal arrangement of the cells in the matrix is incapable of showing how the exemption operations will create intercell flows in the actual layout. The off-diagonal cluster dispersion caused by the shared machines incorrectly suggests machine duplication amongst the cells.

The machine (intracell) layout problem however, has received substantially less attention. Although it is closely related to the facility (intercell) layout problem, certain practical issues prevent direct application of the methods, discussed in the previous paragraphs, for its solution. For example, most of the facility methods consider only equidimensional facilities, which is clearly an unrealistic assumption in the case of machine layout. In addition, most of these methods require that the locations for facility placement, as well as the distances between them, be known a priori, which is not always possible.

From this literature review, one can see that common sense, trial and error, and intuition is also needed to improve existing solution methodologies and techniques. Recent development trends of cellular layout approaches have been moving towards embedding optimisation techniques into heuristic hybrid procedures, to strengthen their effectiveness in layout design.

CHAPTER 3

Integrated *DCMS's* Layout Design Methodologies

3.1. Introduction

This chapter describes the concept of dynamic cellular layout where a combination of *CM* cells (or *GT* cells) and functional layouts are adopted for the machines. This approach is suggested as a result of the critical analysis of *CM* design approaches, which addresses the cell design problem and the associated interactions. These interactions suggest that *PF* formation and machine capacity calculations alone cannot accomplish the design of novel *CM* cells to provide the required flexibility.

A new approach is suggested in this research for *CF* (Cell Formation) which integrates machine grouping and layout design but neglects part family formation, ie. part families with overlapping machine requirements are assumed to be merged to eliminate some machine sharing. This method proposes retaining the shared machines in functional sections, with a "dynamic" assignment of machines to *PF*. It identifies machine groups and the intercell flows which cause the machine sharing problem. However, it utilises the flexibility provided by the layout and handling solutions to minimise the intercell flow delays. This reduces the number of shared machines that must be assigned to two or more cells. Simultaneously, this method provides approximate intracell and intercell layout configurations by utilising a Maximum Weighted Directed Spanning Tree *(MWDRST)* approach. This is the starting point for a more detailed layout design analysis. Thus, appropriate material handling to avoid machine duplication for backtrack flows within a cell, or intercell flows between adjacent cells, is implicitly assumed. However this approach, by exploiting layout and handling strategies, questions the traditional machine duplication solution.

3.2. Critical Analysis of Cellular Manufacturing Approaches

The traditional approach to *CM* requires that an independent cell be designed for each part family. This creates a problem in deciding the integer number of each shared machine, which must be assigned to each cell. Usually, each machine in a cell experiences overloading or under utilisation. Furthermore, none of the current methods addresses the problem of stable cell compositions for product variety manufacturing. Thus, it is difficult to develop a method, which has the ability to integrate the five problems ie. machine grouping, intracell layout, intercell layout, machine sharing and an approximate handling system configuration. Examples of traditional cellular layouts are given in *Figure 3.1.*

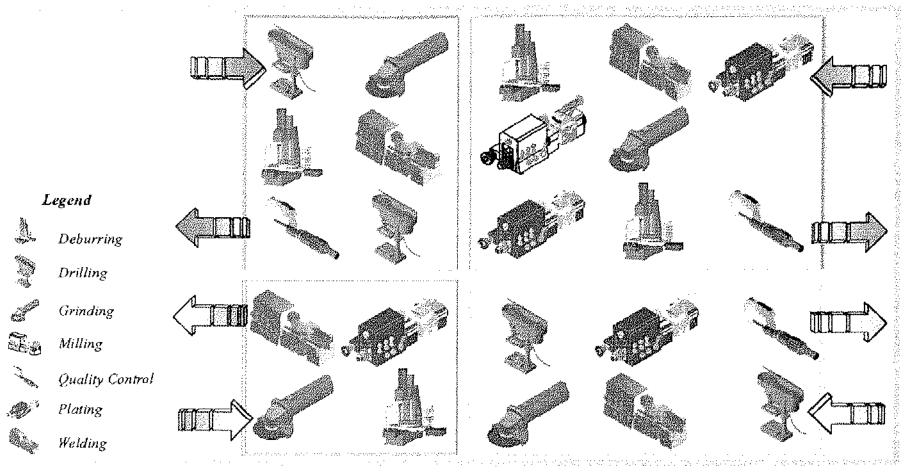


Figure 3. 1 - A traditional (four cells) cellular layout

Modern factory cells may be designed to be autonomous, based on the current mix of parts and of demand volumes. Numerous factors which precipitate intercell flows between these cells are: machine failures, the need to keep expensive one-off type machines loaded, changes in part mix or production quantities for part families, non-integer machine requirements of bottleneck machines required by two or more part families, feasibility of using handling systems to induce intercell moves between adjacent cells, alternative routing for parts when identical machines become duplicated in cells,...etc.

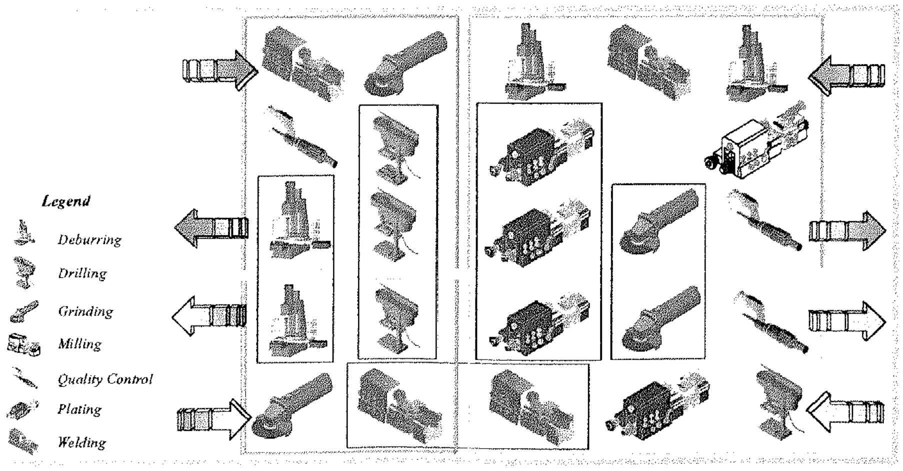


Figure 3. 2. – Four cells with machine sharing

The interactions between some of the important problems in *CF* often make it difficult to identify permanent machine-part compositions for the cells. Additionally, using the standard machine-part matrix clustering representation for cell design, it is possible to get the incorrect impression that cells can be created only through duplication of the shared machine types. The sizes, shapes and adjacencies of the different machine groups are lost in the pure linear arrangement for all the machines in the matrix output solution. Clearly a linear layout structure fails to model the possibility of placing cells that must share the same machines adjacent to each other to minimise intercell flow distances. This machine sharing (*Figure*

3.2) by intercell flows that are readily feasible in a shopfloor environment would completely avoid physical duplication of the machine types. Furthermore, the diagonal arrangement of the cells in the solution matrix is incapable of showing how the exception operations will create intercell flows in the actual layout. From a layout perspective the cells could be placed around a common facility cell, containing all shared machine types.

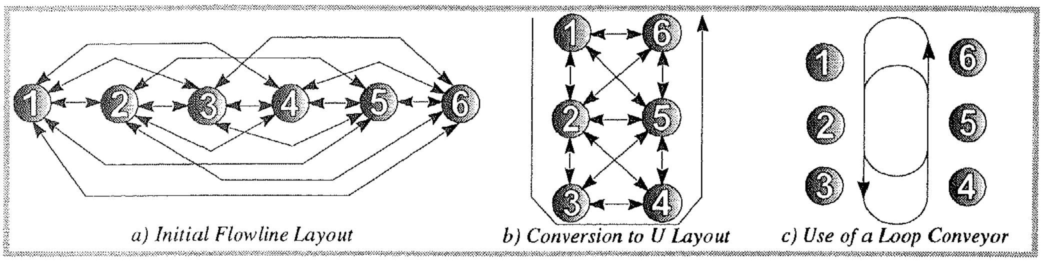
3.2.1. Problems with the Traditional Approaches

A traditional approach to minimising backtracks has been the division and distribution of machines in process specialised departments (functional layouts) into several *PF* based cells arranged in a cellular layout. This creates the classic machine duplication problem discussed extensively in the literature on *CM*. From a historical perspective, the pioneering paper on *PFA* by Burbidge [1963] was the first to specifically discuss machine duplication to divide and distribute the departments in a functional layout in order to design a cellular layout. As regards to machine duplication to create independent cells, Burbidge wrote "... compare the plant requirements for the different packs (or *PF*) and reintegrate to produce the desired number of groups, with the minimum number of machine duplications between groups". Burbidge converted the part routings into a 0 - 1 machine-part matrix and used manual sorting of this matrix to identify the part families and machine groups to constitute the various cells.

Numerous problems arise when the traditional approach is used. The part families may not be obvious from the matrix output of the cluster analysis and matrix diagonalisation methods used for *GA* (Group Analysis). This makes it difficult to clearly identify the number of machine groups, the overlapping in machine requirements amongst the part families, and the particular family to which a part may belong. Nor does the 0 - 1 data capture the design and manufacturing attributes necessary for a more accurate part family formation. In addition, these analytical methods are computational difficult due to the large number of parts involved in the analysis.

The *GA* algorithm assumes a pure linear ordering of all the machines in their matrix output. Also, their binary data representation of the operation sequences neglect the sequence of flow among machines, multiple visits to the same machine by a part, permitted backtracking or crossflows among machines inside the cell, etc. Furthermore, *LA* (Line Analysis) incorrectly assumes availability of all machines when developing an intracell layout for a part family. Hence, practical intracell and intercell layout and handling configurations cannot be modelled by *PFA*. The reason is that an important intermediate stage - shop layout analysis - is missing. This stage must relate *GA, LA*, the locations of the cells, the accessibility of the shared machine in each of these cells to the other cells, and the machine sharing problems. Such integration is attempted in the cell formation method proposed in this research.

GA experiences a machine sharing problem involving duplication among cells to minimise intercell flows. However *LA* experiences a machine sharing problem involving machine duplication at two or more stations within a cell, to reduce intracell flow backtracking. *LA* can only be performed after group analysis determines machine availability for each cell. Both problems are further complicated by the fact that only an integer number of machines can be assigned, even if noninteger machine requirements are computed. An inability to assign the desired numbers of shared machines to each cell will create intercell flow or intracell backtracking flow problems. Intracell machine duplication problems can be avoided by adopting U, L, S, W, or Z, ie. multiple parallel line layout configurations. These would reduce the travel distances amongst the machines and thereby prevent machine duplication.



a) Initial Flowline Layout b) Conversion to U Layout c) Use of a Loop Conveyor

Figure 3. 3 - Avoiding intracell machine duplication

Hence, if the machine sharing and intercell flow problems prevent the formation of independent cells, the scope for eliminating intracell and intercell machine duplication by adopting layout and handling strategies must be explored. For example, *Figure 3.3(a)* shows a six-machine *GT* flowline with flows occurring between all the machines. If a strict flowline layout is desired, then machines 1 and 2 may need to be duplicated near the locations of machines 5 and 6, or vice-versa. However, as shown in *Figure 3.3(b),* the adoption of a U-layout reduces the average travel distances when parts must backtrack to previous machines. This is because backtrack flows are converted to crossover flows among adjacent machines. *Figure 3.3(c)* shows how a loop conveyor system would further reduce the travel times among nonadjacent stations in the original flowline. This would allow identical machines to be located together for better utilisation, thereby minimising the number of machines of this type required within the cell.

If cells having shared machines could be located adjacent to each other, then the intercell flow distances would be comparable to intracell values. Thus intercell flow is encouraged without physically distributing identical machines amongst the two cells. Suitable relocations of machines common to the two cells, which would make them accessible to both the cells from the aisle, would then be indicated. This need for including layout and handling solutions to minimise the need for duplication of shared machines among two or more cells in a *CF* method is another focus of this research. It is based on several critical problems in *CF*, as suggested by the *PFA* model.

3.3. Main Modules (Problems) in Cell Manufacturing System Design

If independent cells are desired for the part families, the machines which are needed in each cell can be easily identified. However, this does not mean that all the machines required by a cell can be located in it. Layout and machine distribution problems can arise if two or more independent cells share machines. Hence, a fundamental argument against the traditional approach is that part family formation, and the distribution of shared machines, constitute only two of the four problems in cell formation (see references presented in Chapter 2). If these problems are considered together, the importance of relating layout decisions to those of machine grouping and part family formation in a cell formation model will be realised. Based on Burbidge's *PFA,* the cell formation can be broken into four distinct problems. The solution of each of these problems usually affects that of the others. The objective of the research proposal, presented in section 3.5, is to develop integrated layout design procedures which will facilitate the design of dynamic (multiple) cellular manufacturing systems (*DCMS*), which include all four CM problems; a scenario hitherto not considered.

The **first problem** is the **identification of part families**. Each part is essentially produced by a set or sequence of machines. If the machine sets for the parts in a family are merged, then the machine group forms a cell. The traditional approach has been to design the same number of cells and part families. This allows each machine group to be associated with a suitable part family.

The **second problem** is the **distribution of machines**. The cells may have common machine requirements if:

- several part families require operations on the same type of machines, and
- there is only one machine of a given type so that it cannot be assigned to any one cell.

In this case, a compromise between the independence of the cells, the intercell flows, and machine utilisation is necessary. However, if certain machines are found to process only one part family, they can be placed in a cell.

The **third problem** is that of **intracell layout**. Within a cell, the machine should be arranged in a flowline ie. the parts flow from one machine to the next in the overall sequence of operations. All the parts need not visit each machine. Such unidirectional material flow may not always be possible and backtracking may be necessary. This can be accommodated by using a suitable handling system or by duplicating identical machine at two or more stations. The proximity of all the machines required for the part family ensures that operations that are performed within the cell experience minimum handling.

The **fourth problem** is that of **intercell layout**. This problem arises if the distribution of machines creates a difficulty in machine duplication. The intercell layout must be designed to locate cells with a common machine requirement close to the material handling facilities. Likewise, if the cells are part of an assembly sequence.

The traditional approach, when applied to part families and distribution of machines, neglects the layout design issues addressed in intracell layout and intercell layout. In order to consider these problems together, an alternative view of the cell formation is proposed. This suggests minimising throughput time and material flow in the shop, rather than manufacturing all the part families in independent cells. Hence, it encourages intercell flows amongst adjacent cells to maximise the utilisation of the shared machines. This converts the problem into one of designing a hybrid layout.

3.4. Interactions between the Modules (Problems)

Interactions between the four problems of cell formation suggest that in the attempt to create independent cells for part families, the sharing of machines may not always be justified. For instance, in adopting a cellular configuration the system is deprived of flexibility to respond to changes in part mix, demand, process plans and/or designs, machine failures, machine tool technology, etc. Instead, if a hybrid layout is designed to minimise material flow with "dynamic" grouping of machines (with families of parts), the benefits of cell and part family formation could still be achieved, provided effective material handling and data processing is available. These interactions are discussed in the following sections.

3.4.1 Part Families and Distribution of Machines

The solution for *PF's* shows which machines are shared. Parts using only the unshared machines will not feature in the intercell flows, or in the capacity calculations. Hence, the distribution of machines is simplified because of the reduction in the number of shared machines that need to be distributed amongst two or more cells. However, the machine sharing decisions, based on intracell layout and intercell layout, influence the distribution of machines. This, in turn, determines the number of cells and overlaps amongst *PF's*.

3.4.2 Part Families, Distribution of Machines and Intracell Layout

Intracell layout can influence machine distribution decisions. For example, two part families may be using the same machine type. However, from a flow perspective, the same machine may feature in different positions in the flowline for each family. Parts in family 1 may use the machine either at the beginning or the end of their operation sequences. Parts in family 2 may require this machine in the middle of their operation sequences. In this case the machine should be assigned to family 2 to simplify the intercell flow scheduling.

If part families are solved by a cluster analysis of the operation sequences, instead of a binary machine-part matrix, then intracell layout can be solved. Intracell machine sharing decisions depend on the particular intracell layout configurations such as loop, multiloop, U, L, S, W or Z shapes, etc. Each of these will generate different intracell handling times and machine duplication requirements.

3.4.3 Part Families, Distribution of Machines and Intercell Layout

Traditionally, machine distribution decisions are made by computing the number of machines whose capacity is equal to the load imposed by the part family. Several cells requiring the same machine type could have noninteger machine requirements. However, only an integer number of machines can be assigned to any cell. This means that the same machine type could be overloaded or idle in adjacent cells. Hence, intercell flows must be allowed and the intercell layout must locate cells which have the same shared machines as key machines near to each other. Clearly, in cellular manufacturing design, cells with high machine utilisation, production flexibility and no intercell flows are difficult to achieve.

Expensive single machines such as *CNC* turning or machining centres, may not be practical to locate in a cell. To keep their utilisation high, companies may prefer loading parts from several families on such machines. Intercell flows to such a machines cannot be eliminated. Thus, the layout can be planned to have such machines centrally located in a common facilities section. Similarly, incompatible processes such as heat treatment, plating or presswork must normally be excluded from the cells due to machine costs, environmental problems or incompatibility with other processes. Unless an alternative compatible process is identified, they need to be located away from the cells. An appropriate layout must be designed which isolates these facilities from the cells. The increase in distances for parts travelling to and from these facilities is unavoidable. Thus for minimisation of delays the intercell flow must rely on efficient intercell flow scheduling and handling systems.

A self contained cell cannot be designed for parts which use either a majority of incompatible processes, or subcontract work, or single expensive machines shared by other cells. Also merging the few remaining machines in such a cell with another cell will make scheduling in the larger cell more difficult. Instead, since intercell flows are unavoidable, these machines could be absorbed into a common facilities section, which is equivalent to a functional layout. However, whilst utilisation of the machines in such a cell is high, the larger scheduling problems are equivalent to those of a complicated job shop.

3.4.4 Intracell Layout and Intercell Layout

If the flowline of a cell has backtracking it creates a machine distribution problem within the cell. If machines of the same type are located at two or more stations within the cell, machine utilisation and production flexibility is affected. Process planning could help to eliminate the operations which cause backtracking, or the use of a suitable material handling system could allow parts to cycle within the cell from previous stations. However, if backtracking is unavoidable, it increases the number of machines required within the cell. Hence, the intracell layout design will depend on how many machines are made available to the cell after intercell machine distributions are made.

In addition, the intracell configuration, obtained by solving the intracell layout, determines the locations of the shared machines within a cell. Furthermore, the arrangement of the machines along the perimeter of the cell will limit the accessibility of the other cells to its shared machines. Hence, the problems of locating the cells in the shop, and determining which sides of the cells should be adjacent to each other, are related.

3.4. Alternative-Enhanced Approaches to Cellular Manufacturing

It may be noted, that as early as 1952, Ireson recognised that a manufacturing facility layout might need to have a combination of the product grouping (or *PF* formation) and process specialization attributes of the cellular and functional layouts. He stated that "...the combination method of departmentalisation is accomplished by a functional layout of machines in long, narrow departments, with the products flowing at right angles to the departments .... (but) ... there must be sufficient similarity in the products and the steps of production so that such a plan can be followed without excessive backtracking of parts".

By using simulation techniques, several researchers [Ang 1984, Flynn 1987, Gupta 1982] have investigated the utility of a hybrid layout for batch manufacturing. In reality this is similar to the concept of dynamic cells which is a combination of functional and cellular layouts. Thus, having a functional layout for shared machine offers high machine utilisation and loading flexibility. Furthermore, organising some machines unique to a part family into cells will lead to reduced throughput times and handling costs. The throughput delays due to some intercell flows can be eliminated by allowing some machine duplication amongst cells, whilst other intercell flow delays can be reduced by efficient intercell transferees of some batches of parts. An example of four such (dynamic) cells is illustrated in *Figure 3.4.*

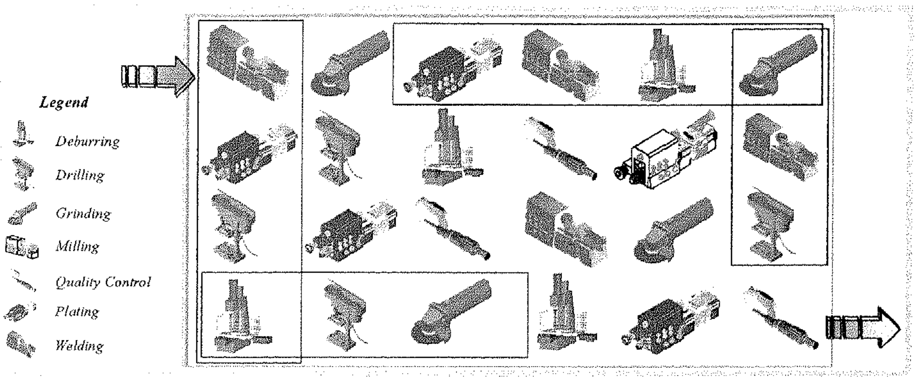


Figure 3. 4 - A dynamic cellular layout with machine sharing

This hybrid configuration is adequate for cellular manufacturing since these researchers (Ang, Flyn and Gupta) only considered the optimum layout of a single group of machines. In their papers, decisions concerning machine duplication were limited to whether identical machines could be placed at two or more stations within the cell. If an intercell layout has to be incorporated with intracell layout, an additional type of flow is created when the cells have common machine requirements. Additional flows may occur between any pair of flowlines if the machine loads are such that an integer number of machines cannot be assigned to each cell. This machine allocation problem makes intracell and intercell layout design strongly related to the machine sharing problem. When intercell flows exist it is more important to eliminate these flows by machine duplication, since otherwise these shared machines would have even higher queuing delays for the parts involved. This might reduce the number of machines available within a cell for duplication at two or more stations, to reduce intracell-handling times. Finally, the problems of machine utilisation and allocations of shared machines at the intercell and intracell levels are interrelated. With current advances in handling system capabilities, the intracell machine duplication problem can be considered to be similar to that of intercell flows.

Designing a hybrid layout can be done with a limited amount of input data. Intracell layout and intercell layout can be solved using only machine-specific data, such as the flow information captured in the travel chart. This chart simply aggregates the operation sequences and batch quantities of all the parts. Detailed part information is not required unless part families and distribution of machines need to be solved. The suggested method of partitioning the flow network in the travel chart is expected to be sufficient for:

1. locating machines and their supporting equipment to single part families in cells,
2. locating shared machines in functionally organised sections accessible to all the cells if the intercell flows are between adjacent cells,

3. distributing some shared machines amongst nonadjacent cells if a functional layout for them creates intercell flows,
4. designing an approximate layout within the cells where backtracking is eliminated by allowing bi-directional flow,
5. planning the overall shop layout of all the flowlines, based on overlapping of their machine requirements to reduce intercell flows, and
6. based on shop layout (5), suggesting the paths for the material handling system re the intercell flows that will occur.

Holographic layouts [Montreuil 1996] and fractal layouts [Venkatadri 1997] are non-traditional layouts that extend the idea of the traditional functional layout, since they distribute identical machines at multiple locations on the factory floor. In particular, fractal layouts divide the manufacturing facility into multiple, nearly identical machine cells. Each cell contains a heterogeneous mix of machines and is called a fractal. For instance, one fractal may contain a drill press, two machining centres, a turret lathe, welding station, a large press, a small press and a test station. Arriving jobs are assigned to the fractal with the largest available resource capacity relative to the job requirements. A fundamental limitation of the method for designing a fractal layout is that it does not consider the routings of the products for which the layout is being designed. Hence, for each product routing, it is required to determine the exact allocation of each operation to one of several identical machines distributed (or duplicated) at multiple locations in the layout. A related idea, that of giving flexibility to a manufacturing layout by distributing identical machines at several non adjacent locations on the shop floor, is discussed by Webster [1980].

Holonic layouts [Askin 1996] appear as random arrangements of machines. No specific cell boundaries exist; instead each machine acts as a "holon", which is an autonomous entity capable of broadcasting its availability and bidding its services. In a holonic layout the multiple machines of a type are spread throughout the facility. The intent is that, for any possible part routing that may be developed for a job, the job can be accommodated with a routing to adjacent machines in the facility. As with the fractal layout, the fundamental limitation of the method for designing a holonic layout is that it does not consider the routings of the products for which the layout is being designed.

Several authors have focussed on decomposing a layout into a set of independent or interacting flowlines. Vakharia, et al [1990] presented a *CF* method based on analysis of the operation sequences of the parts. Their method duplicates machines to create a system of flowline cells with minimum intercell flows. More recently Askin [1998] proposed an enhanced algorithm to solve the same problem, whilst Ho [1993] presented a layout design technique that exploits the similarity of product assembly sequences in a product family to design a network type layout for a multi product flowline. Moodie [1994] discussed the case of the design of a network of manufacturing cells using product sequence similarity analysis where all cells have a flowline layout.

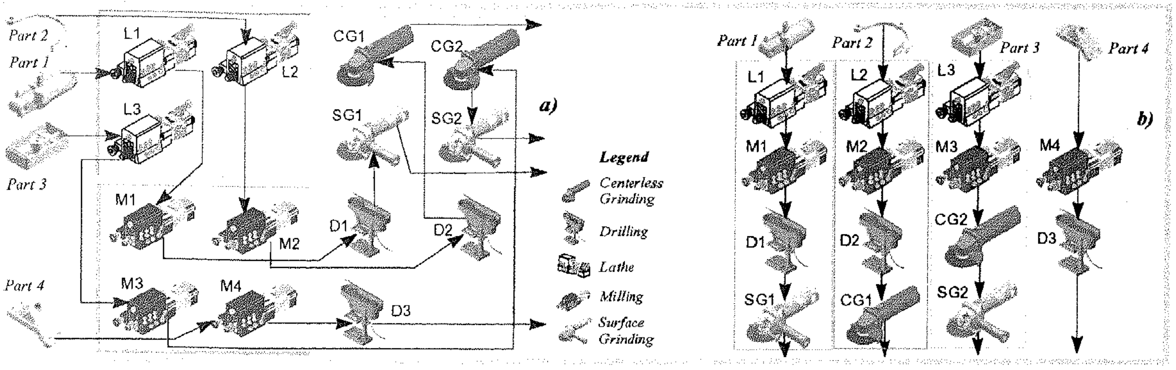


Figure 3. 5 - a) Functional layout and b) Cellular layout [Gallagher 1973]

The concept of a virtual manufacturing cell has been proposed by the National Bureau of Standards to address specific control problems in the design of their Automated Manufacturing Research Facility [Simpson 1982]. However, no significant exploration and progress in this field has been made until recently. It extends the concept of the traditional cell by allowing time sharing of machines with other virtual cells that produce different *PF's*, but have overlapping resource requirements. The cell is no longer identifiable as a fixed physical group of machines. The *CF* method suggested in this thesis is a step in this direction as it assumes intercell machine sharing to be feasible. Finally, such a virtual cell layout relaxes the traditional view that a cell must be dedicated to a *PF*. Parts from a family may visit cells other than the cell that they have been assigned to initially. In addition, cells need not be designed such that they contain all machines necessary to produce a *PF*. In this non-traditional layout, cells containing machines from the same department are located so as to allow regrouping of the machines duplicated in several cells into a process department, as in the original functional layout. Based on *Figure 3.7* [Gallagher 1973], *Figure 3.6* demonstrates the concept of a virtual cellular layout that combines the properties of the standard functional layout of *Figure 3.5(a)* with the cellular layout of *Figure 3.5(b)*.

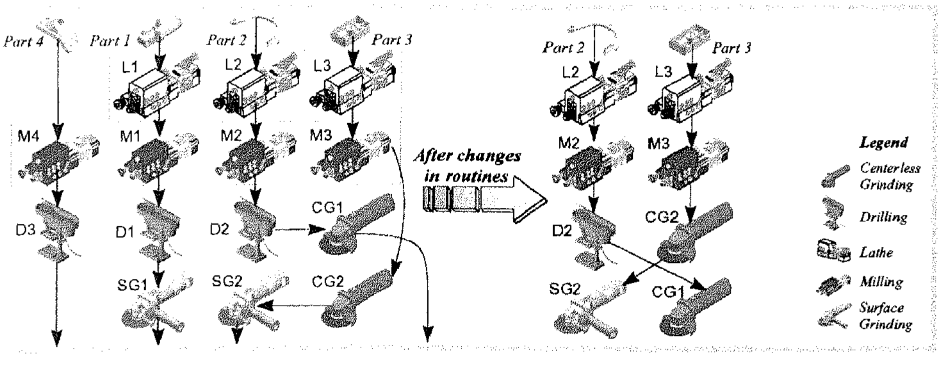


Figure 3. 6 - Virtual cellular layout

The flowlines in *Figure 3.5 (b)* have been permutated in order to bring the identical machines next to each other. The machines remain dedicated to their part families. However, they are retained in functional layouts to allow flexibility in machine reassignments when machines failure or the part mix and/or demands change. Suitable aisle configurations and handling capabilities are assumed which would allow permitted random flows. For example, the shafts with keyways flow directly from the milling machines (M) to the centerless grinders (CG), bypassing the drilling machines (D).

A recent study by Drolett [1994] demonstrated the feasibility and desirability of virtual cells for small batch production, where a new scheme for scheduling and control of such a system is proposed. He proposed a linear programming model for creating and scheduling virtual cells considering machines, fixtures and critical tools. The mathematical model in fact is a Gantt chart, with two constraints. The first constraint concentrates on the detailed horizon which represents the creation of variables and the creation of virtual cells for jobs at any period of time. The second constraint concentrates on the aggregate horizon were specific routes for virtual cells are not considered. In addition, the same author [1990] states a basic definition and terminology of virtual cellular manufacturing.

However, the framework presented by Drolett does not include the bottom nor the top of the control hierarchy. In fact improving it requires fulfilling the gap remaining at the bottom level (ie. the machine level), and at the top level (ie. strategic and aggregate planning level). Furthermore, integrated layout design is another aspect that should be looked at more seriously. Possible areas for improving Drolett's study are:

1. Dynamic resource's classification - the combinatorial complexity makes it impossible to plan the utilisation of every single resource. Similar to what a schedular would do, it is best to concentrate, at least at the planning stage, on tightly constrained resources. However, a resource could be tightly constrained (critical) at one time and lightly constrained at another time. Thus resources (ie. tools, fixture, pallets and machines) can be monitored and when the utilisation of a particular resource reaches a certain level, its classification would automatically change to critical. Then a utilisation profile graph would be attached to it and this resource would be considered in the scheduling process. If its utilisation became sparse, one could change its classification and continue monitoring it.
2. Control algorithms for *AGV's* - given the quantity of virtual cells coexisting in the system at any given time, the possibility of material flow congestion is limited. In fact, congestion can occur only when cells overlap or cross each other. In these cases several strategies can be implemented to reduce the risk of congestion.
3. Physical relocation of machines – with attention to the utilisation of the virtual cellular concept in an environment where a dynamic physical reconfiguration is possible. A few movable machines in a virtual cellular manufacturing system could reduce the distance flow significantly. In such a case, an economic function which considers the setup cost (ie. cost of moving the machines) and the cost saving in material handling, due to the imminent proximity of machines, could be used as decision criteria during the virtual cell creation phase. The virtual cellular concept can be used in systems which support dynamic physical reconfigurations. However, new algorithms which take advantage of dynamic reconfiguration, will have to be developed, since the advent of integrated, modular, movable machines are expected in the future.
4. Schedule the unexpected event - good scheduling strategies should deal with stochastic occurrences of failures and their duration. Besides the system's state and the job order's list, the control algorithm

should consider the probability of failures for each resource. Statistics regarding the frequency and duration of failures on each machine could be updated during the system's evolution. A scheduling algorithm, which deals with the probability of occurrence of unexpected events, would be a step forward.

In 1998 Sarker proposed a method to create virtual cells by applying the double sweep algorithm in the k-shortest (graph theory algorithm, presented in Appendix A) path problem. Here the objective of the virtual cell creation was to minimise the total throughput time of a given set of jobs by assigning the job to the appropriate processors. Thus the results generated from this method include not only the optimal candidate for a virtual cell with the shortest throughput time, but also the secondary candidates with alternatives routes. The author did not suggest a sorting (factor) procedure for the shortest paths, nor did he address the problem of layout design.

All of these layouts are developed from the initial machine part grouping analyses used to design independent cells. However, during the layout phase, creative layout strategies are adopted to place the duplicated machines as if they had been retained in functional departments. These layouts are realised by modifying (a) the machine-part compositions of the cells, or (b) the shapes of the cells, or (c) the orientations of the cells, or (d) the locations of the cells. Instead of a pure cellular or functional layout, these layouts represent a partial conversion to a cellular layout, with novel fusion of functional grouping with several shared machine types, limited physical duplication of shared machines, and intercell flows. Other variations of the concept of hybrid layouts are: where an existing layout is replaced by a combination of manufacturing cells and individual machines [Harhalakis 1996], cascading cells [Tilsley 1997] and remainder cells [Shunk 1995]. Terminology and explanations of these approaches can be found in these references.

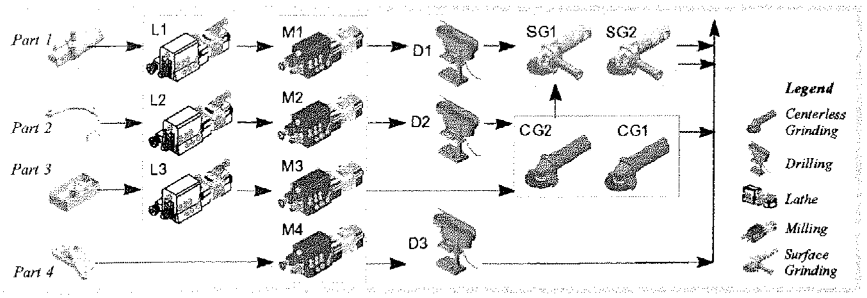


Figure 3. 7 - Dynamic cells in functional layout

From the presented analyses of these approaches it can be seen that the concept of dynamic cells *(Figure 3.7)* seems promising. Thus a concept which relies on the development of algorithms to deal with complex, multi-virtual interaction and integrated layout design will clearly support a viable programme of research. The concept will also maintain the low investment required for *CM* cells, whilst gaining some flexibility and efficiency associated with *FMS's*. Thus by organising some machine types, unique to a *PF*, into dynamic cells, the benefits of reduced throughput times and handling cost for those parts can be gained. The throughput delays due to some intercell flows can be eliminated by allowing some machine duplication amongst cells, whilst other intercell flow delays can be reduced by efficient intercell transfers of some batches of parts.

The overall material flow network of the shop can then be decomposed into a set of branched flowlines. Cells, which have several machines in common, can be placed adjacent to each other and feasible functional groupings can then be obtained by permutating the flowlines. The proposed research method can then finally attempt to merge those flowlines, which use the same sequence of machines, and create functional sections for the machines they share. Effectively resulting in a dynamic layout of functional sections and machine groups for those machines that are specific to a single part family.

Subproblems	*Objectives*	*Problems and Tasks*
Cell Formation	• Minimising intercell flow. • Maximising processing similarities of parts within cells. • Minimising the total (operation) cost.	1. All cell formation procedure with the attributes and considerations such as: • flexible processing & routing consideration, • cost awareness, • detailed grouping results including part & machine content of cells, total costs and assignment of operations, and outer-cell operations. 2. Cell formation problems in different situations.
Layout	• Effective intercell flow. • Effective intracell flow.	1. Flow matrices estimation. 2. Cell layout problems. 3. Machine layout problems. 4. Flow path considerations: • flow path configuration, • flow path directions. 5. Input/Output points to location problems considering flow path configuration, intercell flow, and intracell flow.
Integration	• Consistency between the solutions of different control levels (modules).	1. Integration between cell formation and layout problems. 2. Integration between cell layout and machine layout problems.
Simulation (Verification and Design)	• Ensuring the validity and quality of final solution.	1. A methodology that is capable of: • modelling a multiple-cell environment, • simulating different design configuration, and • analysing and verifying the tested systems. 2. Performance analysis. 3. Detecting causes of problems. 4. Re-design justification function.

Table 3. 1 - Summary of the research objectives and the tasks of all problems

3.5. Integrated Cellular Manufacturing Layout Methodology - Research Proposal

The objective of this research proposal is to develop integrated layout design procedures which will facilitate the design of dynamic (multiple) cellular manufacturing systems (*DCMS*) with improved material flow handling efficiency (referenced in section 3.4). In a *DCMS*, there are two types of material flow: the first is the flow between cells (intercell flow); and the second type is the flow between the machines within cells (intracell flow). The amount of intercell flow is affected by the cell formation. Intracell flow movements cannot be avoided and hence affect cell formation. Some of the factors which affect material flow efficiency are: material flow controls; the layout of machines and cells; and material handling systems.

This part of the research will focus on those problems that are related to the layout arrangements of manufacturing facilities. As shown in *Table 3.1* which summarises the objectives and tasks of each problem, these objectives: include cell formations to decide which parts and machines should be grouped together; layout problems to arrange machine, cells, connecting flow paths, and input/output points of cells; the integrated design problems to maintain the consistency between the results achieved in cell formation and layout problems; and the verification and redesign problems to ensure the quality and feasibility of the final solution. Some of the challenges in this study come from allowing choices in process and routes. Here, traditionally, if even more than one process plan is considered for each part, standard part/machine grouping techniques cannot be applied. Similarly, when multiple routings are considered, techniques that rely on knowing the exact flow between facilities cannot be used either. The following sections will describe the problems and outline the objectives and tasks required for each problem.

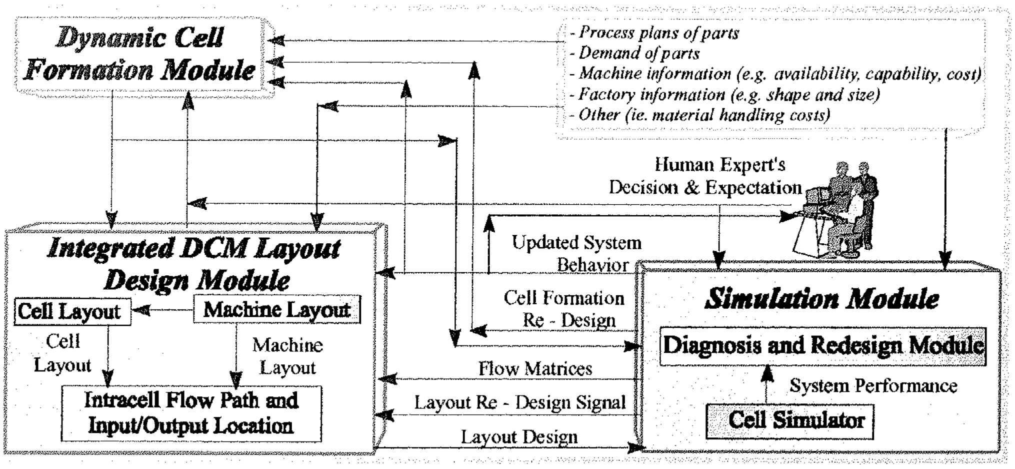

Dynamic Cell Formation Module
- Process plans of parts
- Demand of parts
- Machine information (e.g. availability, capability, cost)
- Factory information (e.g. shape and size)
- Other (ie. material handling costs)
Human Expert's Decision & Expectation
Integrated DCM Layout Design Module
Cell Layout
Machine Layout
Cell Layout
Machine Layout
Intracell Flow Path and Input/Output Location
Updated System Behavior
Cell Formation Re - Design
Flow Matrices
Layout Re - Design Signal
Layout Design
Simulation Module
Diagnosis and Redesign Module
System Performance
Cell Simulator

Figure 3. 8 - The structure of the design procedure

Figure 3.8 defines the relationship between different design modules. It outlines the tasks in each module, the information flow between modules, the design process, and the decision and control variables. Although the integration of different design modules at each individual design module is required, it's verification and redesign is still necessary. This is due to the stochastic nature of the flexible approach of this research. *Figure 3.9* shows the major steps of the entire design process which will be explained in the later section of this Chapter.

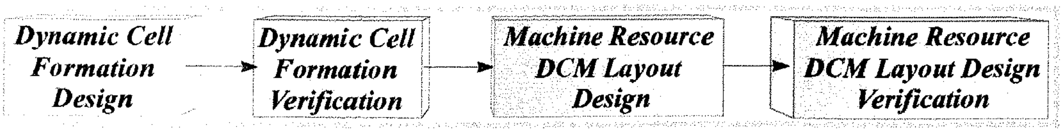

Dynamic Cell Formation Design
Dynamic Cell Formation Verification
Machine Resource DCM Layout Design
Machine Resource DCM Layout Design Verification

Figure 3. 9 - The main stages of the design procedure

3.5.1. Cell Formation Module - Problem

As mentioned before, the recommended solution should be able to consider the processing and routing flexibility of parts if they are required. One difficulty in solving the cell formation problem in a flexible system, is that much information that was usually at hand in a traditional system, is no longer available in the flexible system. Secondly, the solutions should meet the production goal and resource (machine and tool) constraints. Thirdly, the solutions should be cost-effective, ie. it should include cost minimisation as an objective function. Finally, in addition to the basic grouping information, the solutions should provide information such as, the assignments of each machine, the total cost, etc. The information obtained at this stage will then become useful for the layout problems.

Once the cell formation result has been formed, the next step is to verify the result. Simulation will be used for this purpose; the cell formation re-design procedure will be discussed later. *Figure 3.10* defines the relationships of the cell formation module with the other modules and the design process for the cell formation module.

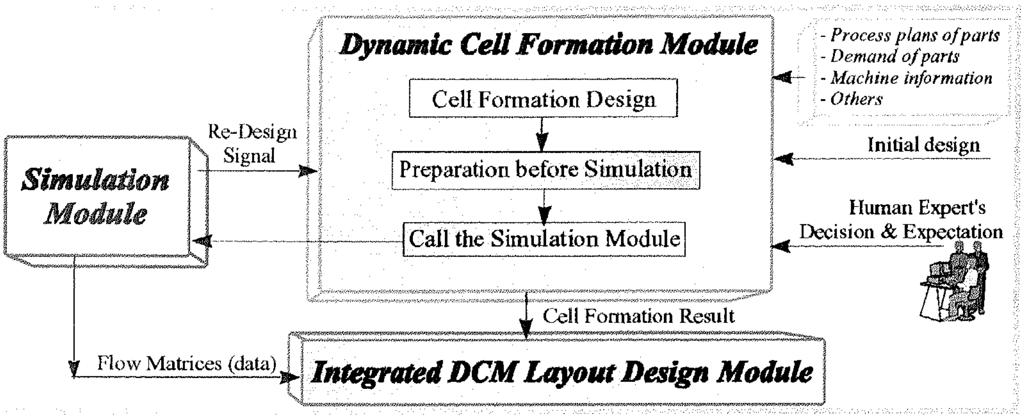


Figure 3. 10 - The designs steps of the cell formation module and their relationships with other modules

3.5.2. Layout Module - Problem

Problems which are investigated here include machine layout problems, cell layout problems, flow path layout problems, and cell input/output location problems. The overall objective is to achieve efficient intercell and intracell material flow through the layout arrangements of the manufacturing facilities.

Flow	Personnel Movement Material Transfer (Parts and Tools) Information Flow Power Transfer
Flexibility	Excess capacity for reassignment of units Scope for expansion and modification
Non-Flow	Floor Space Utilisation Safety Ease of Supervision Sharing of Machine Frequency of Machine Failure Air Conditioning Personnel Needs

Table 3. 2 - Typical criteria for facilities layout

3.5.2.1. Objectives

The common objective adopted by many layout solution procedures is the minimisation of total flow distances. Some procedures deal with the minimisation of material handling costs, which are often taken as functions of total flow distance and the types of material handling systems used. As stated earlier, because of the ill-structured nature of layout problems, many qualitative characteristics need to be considered. Some of them are 'positive' factors that require closeness of facilities, such as ease of supervision and communication, whilst some are negative factors, such as remoteness of facilities with respect to noise, dust, and safety. Common layout criteria can be classified into three groups: flow, flexibility and non-flow (*Table 3.2)*. The layout approaches usually employ a combination of different criteria as their objective.

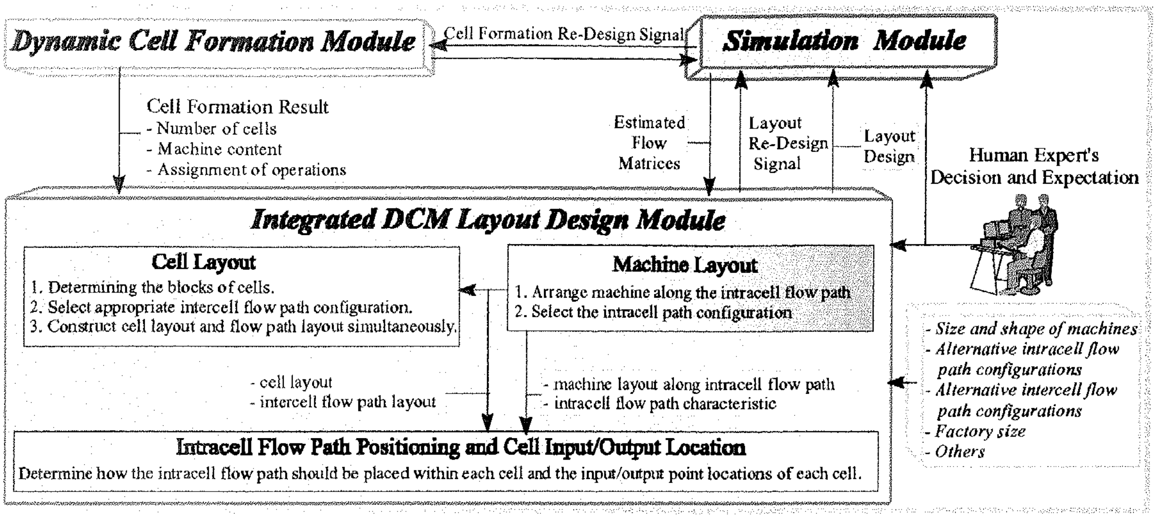

Dynamic Cell Formation Module
Cell Formation Re-Design Signal
Simulation Module
Cell Formation Result
- Number of cells
- Machine content
- Assignment of operations
Estimated Flow Matrices
Layout Re-Design Signal
Layout Design
Human Expert's Decision and Expectation
Integrated DCM Layout Design Module
Cell Layout
1. Determining the blocks of cells.
2. Select appropriate intercell flow path configuration.
3. Construct cell layout and flow path layout simultaneously.
Machine Layout
1. Arrange machine along the intracell flow path
2. Select the intracell path configuration
- Size and shape of machines
- Alternative intracell flow path configurations
- Alternative intercell flow path configurations
- Factory size
- Others
- cell layout
- intercell flow path layout
- machine layout along intracell flow path
- intracell flow path characteristic
Intracell Flow Path Positioning and Cell Input/Output Location
Determine how the intracell flow path should be placed within each cell and the input/output point locations of each cell.

Figure 3. 11 - Integrated layout module and its relationships with other modules

3.5.3. Integration of Different Layout Design Modules

The objective here is to ensure that the result obtained in one module can be maintained throughout the entire design process. For example, the result from the cell formation procedure will provide not only the machines in the cells, but also the assignment of operations to the machines. In other words, the objective for the cell formation is maintained for the subsequent design modules stage. Other relevant tasks in these modules include: identifying the relationships between these modules (such as the effects of one module on others) and ensuring the required information for every module can be obtained appropriately.

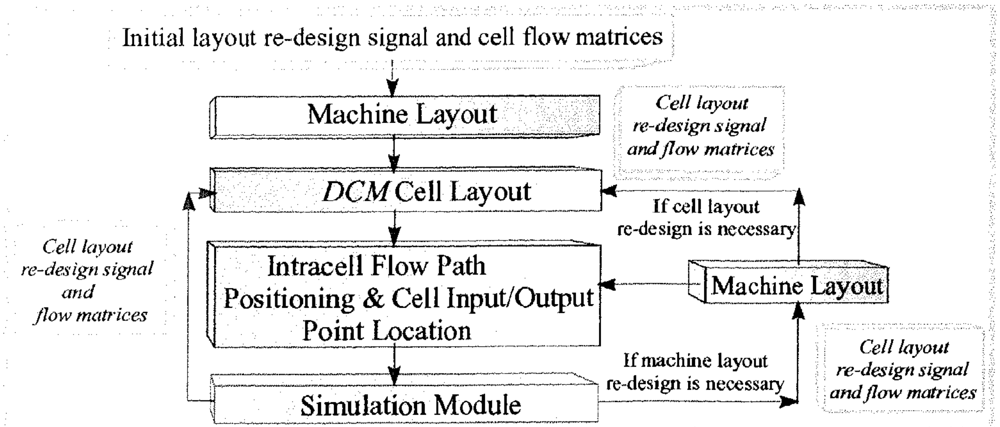

Initial layout re-design signal and cell flow matrices
Machine Layout
Cell layout re-design signal and flow matrices
DCM Cell Layout
If cell layout re-design is necessary
Cell layout re-design signal and flow matrices
Intracell Flow Path Positioning & Cell Input/Output Point Location
Machine Layout
If machine layout re-design is necessary
Cell layout re-design signal and flow matrices
Simulation Module

Figure 3. 12 - Design steps of integrated layout design module

The suggested integrated layout procedure (module) is summarised in *Figure 3.11,* which shows the interactions within the integrated layout design procedure module. This procedure contains three components: the machine layout, the cell layout and the paths within the cell, including the location of the cells input/output. *Figure 3.12* shows a step by step design procedure of the integrated layout design module.

In *Figure 3.13* the machine layout module is shown, were the main purpose is to determine the arrangement of the machines along the intracell flow paths. The objective function can be a single objective or multiple objective function. However, the main purpose of the cell layout module is to minimise the total intercell flow distance by appropriate arranging the cells and the connecting intracell flow paths. *Figure 3.13* also defines the relationships of the machine layout module with other modules.

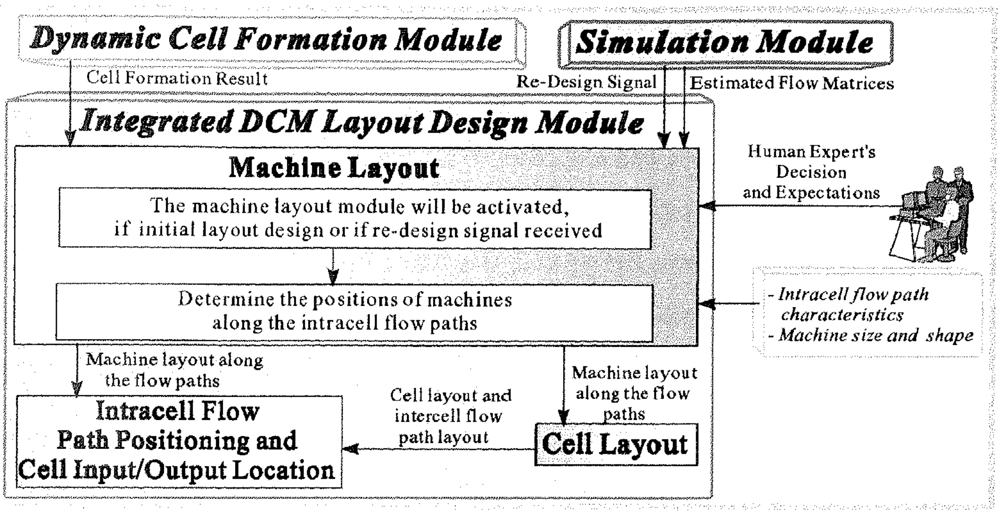

Dynamic Cell Formation Module
Simulation Module
Cell Formation Result
Re-Design Signal
Estimated Flow Matrices
Integrated DCM Layout Design Module
Machine Layout
The machine layout module will be activated, if initial layout design or if re-design signal received
Determine the positions of machines along the intracell flow paths
Human Expert's Decision and Expectations
- Intracell flow path characteristics
- Machine size and shape
Machine layout along the flow paths
Machine layout along the flow paths
Cell layout and intercell flow path layout
Intracell Flow Path Positioning and Cell Input/Output Location
Cell Layout

Figure 3. 13 - The relationships of the machine layout with other modules

With the size and shape of the factory known, the number of cells comes from the cell formation module. Whilst the intercell flow matrix comes from the simulation experiments. However the size and shape of a cell depends on the total space required for the intracell facilities (machines, flow paths, handling devices and storage etc.) within the cell, and the intracell flow path configuration. The decision is based on which feature will provide easier layout solutions, and more efficient material flow between intercell and intracell flowpaths. *Figure 3.14* illustrates the relationships of this component, and the cell layout model procedure, with other components and modules.

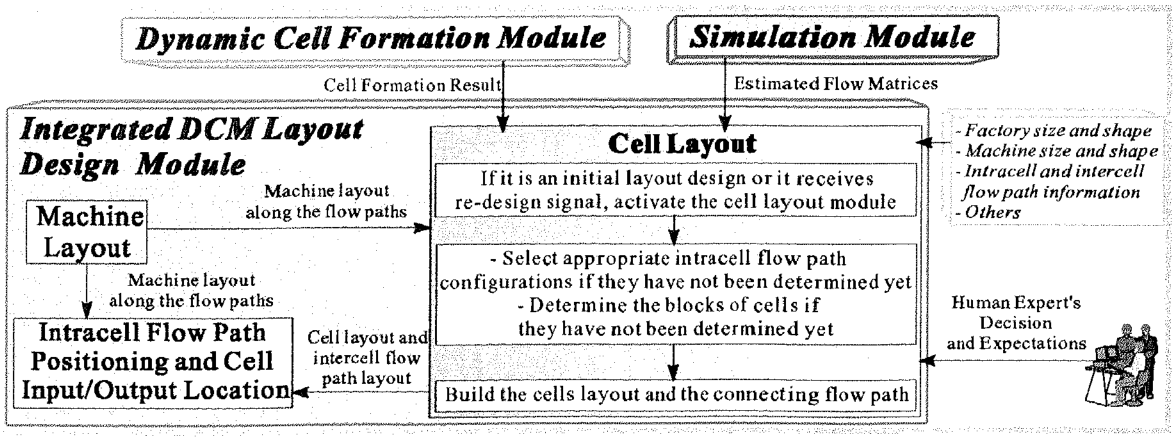

Dynamic Cell Formation Module
Simulation Module
Cell Formation Result
Estimated Flow Matrices
Integrated DCM Layout Design Module
Cell Layout
- Factory size and shape
- Machine size and shape
- Intracell and intercell flowpath information
- Others
If it is an initial layout design or it receives re-design signal, activate the cell layout module
Machine Layout
Machine layout along the flow paths
Machine layout along the flow paths
- Select appropriate intracell flow path configurations if they have not been determined yet
- Determine the blocks of cells if they have not been determined yet
Human Expert's Decision and Expectations
Intracell Flow Path Positioning and Cell Input/Output Location
Cell layout and intercell flow path layout
Build the cells layout and the connecting flow path

Figure 3. 14 - The relationships between the cell layout model and other modules

3.5.4. Simulation (validation) Module - Problem

The purpose of the simulation module is to ensure the validity and quality of the final solution. Due to the stochastic nature of the flexible approach of this research, the integration of different design modules at each individual design module is required. Furthermore it's verification and redesign is still necessary. In addition, as described previously, much information is not specific here and needs to be estimated. This implies that a solution may not always be satisfactory.

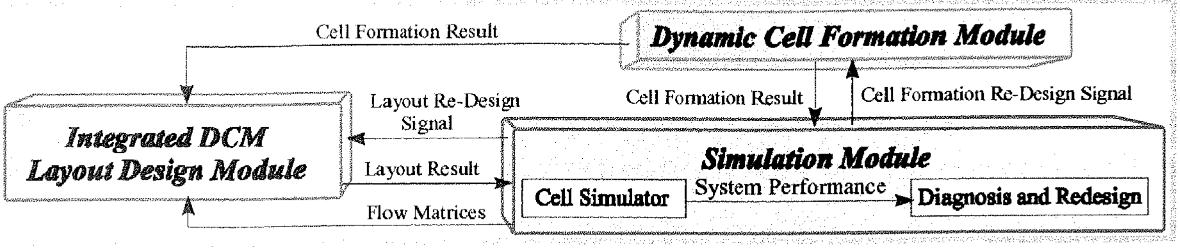


Figure 3. 15 - The components of the simulation (verification and re-design) module and their relationships

To resolve these problems, a simulation module (verification and redesign scheme) is required that can identify the causes of the problems and address them by a re-design of the system. The simulation module contains two components: a simulator, and a diagnosis and redesign component. *Figure 3.15* shows the relationships between these two components and the relationships between the verification and redesign module and other modules.

3.5.4.1. Cell Simulator

Simulation is a powerful tool to verify the proposed cell designs. It is especially useful for many complex and real-world systems with a stochastic nature that cannot be accurately described by mathematical models. There are many other advantages of simulation. Some of these are:

- Simulation can be used to help analyse a proposed system performance, even with sketchy input data.
- Once a system model has been built, it can be used repeatedly to analyse proposed system re-designs.
- Data from simulation models are less costly to obtain than similar data from real systems.
- Analytical models usually require many simplifying assumptions to make them mathematical tractable; simulation models have no such restrictions.

The simulator must possess the following characteristics:

- be able to model a dynamic, multiple-cell manufacturing system,
- be able to simulate under different layout configurations and flow control strategies, and
- provide all the necessary information for the analysis and verification of the proposed systems.

An example of a simulation model using "Witness" is shown in *Figure 3.16*.

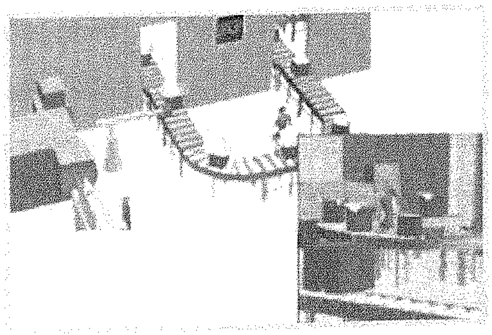


Figure 3. 16 - Simulation model using "Witness"

3.5.4.2. Analysis, Diagnosis, and Redesign

By analysing the simulation results, any discrepancy between the performance of the proposed system and the expected performance can be detected and the problems identified. After the problems have been identified, the causes of the problems can be diagnosed and appropriate remedies prescribed. In addition, to prevent unnecessary re-designs, this procedure should have the ability to determine whether another re-design and verification cycle is necessary.

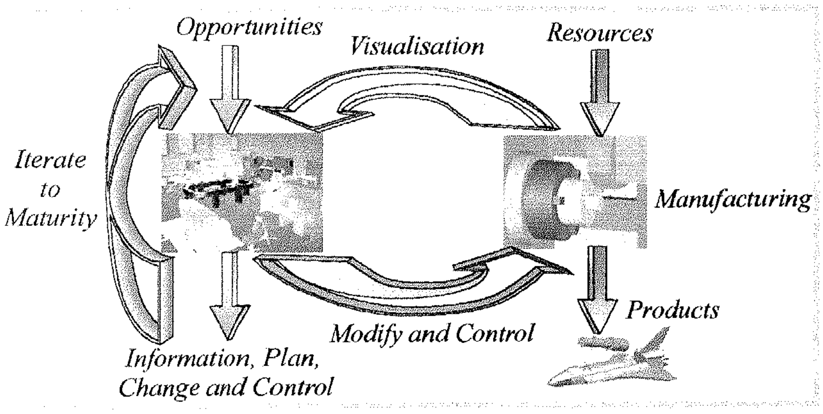


Figure 3. 17 - Research methodology for design of DCMS's

3.6. Research Methodologies for Analysis & Design of *DCMS's*

The proposed research methodology (*Figure 3.17*) develops a flow-based approach for the formation of dynamic manufacturing cells, which integrates machine grouping, shop layout design and intercell flow handling. Part families with overlapping machine requirements are assumed to be merged to eliminate the need to duplicate shared machines amongst competing cells. Machines shared by several cells are assumed to be retained in functional sections if these cells can be located adjacent to each other. This adjacency of the cells allows the machine groups for the part families to be dynamic, ie. the handling system links machines in adjacent cells in order to define the cell for a part family, without necessitating rigid physical relocation to group the machines. Thus, the machine groups can be "dynamic" ie. parts from several families can be loaded on a particular machine shared by several cells, or can enable identical machines to be placed in a functional layout for production flexibility. Another approach to flexible layout design is to directly incorporate known forecast data changes into the layout problem. Machines retained in a functional layout can be dedicated to parts from a family without physically distributing these shared machines amongst two or more cells. Simply by assigning different tools and fixtures to a machine will dedicate the machine to different part families in successive production periods with suitable material handling resources.

It is also assumed that shared machines be grouped in a functional section that is equally accessible to all associated cells. This may require rearranging the internal layout of the cells, so that shared machines are positioned close to each other, and/or readjusting the overall layout of the cells, so that the intercell distances between overlapping cells are minimised.

Legend: *Y = Directly addresses subproblem* *N = Does not do..........* *P = Addresses subproblem indirectly* *M = Machine-part matrix* *S = Operation sequences* *L = Load calculations* *T = Travel chart*	***Matrix Methods***	***Cluster Analysis***	***Overlapping Clusters***	***Multivariate Statistic***	***Pattern Recognition***	***Travel Chart Clustering***	***Graph Theory***	***Mathematical Programming***
Input Data	**MST**	**MT**	**MT**	**MT**	**MS**	**TL**	**MT**	**MLT**
Machine Groups	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**
Part Families	**Y**	**Y**	**Y**	**Y**	**Y**	**P**	**Y**	**Y**
Machine Sharing	**P**	**P**	**Y**	**P**	**Y**	**P**	**P**	**P**
Intracell Layout	**N**	**P**	**N**	**N**	**Y**	**P**	**P**	**N**
Intercell Layout	**P**	**P**	**N**	**N**	**P**	**P**	**P**	**N**
Handling Links	**N**	**N**	**N**	**N**	**P**	**P**	**P**	**N**
Multiple Data Sets	**N**	**N**	**Y**	**N**	**N**	**N**	**N**	**P**

Table 3. 3 - Capability analysis of cell formation methods

The traditional implementation of cellular manufacturing becomes problematical under more variables and dynamic conditions. It is unclear how a cellular arrangement could accommodate part families with fluctuating demand and/or changeable part mix composition. It is also doubtful as to how independent cells can be constructed for parts with relatively short life cycles and/or with frequent design and manufacturing changes. Furthermore, traditional independent cells appear to result in inefficiencies due to duplication of processing capacity (machines) amongst cells with similar requirements. Since production volumes for part families rarely correspond to integer machine requirements, duplication of the same machines amongst different cells can result in poor capacity utilisation and high costs.

The fact that intercell flows are discouraged in traditional *CM* highlights the additional inefficiencies that can result from not taking advantage of alternative part routing that are increasingly becoming feasible by manufacturing integration. This lack of cooperation amongst cells also increases the vulnerability of the manufacturing system to machine failure and labour shortage. *Table 3.3* gives a review of the capabilities of the different types of cell formation methods. Their evaluation is based on the ability to address the stated cell formation problems.

Unlike the static cell, which is defined by a fixed grouping of machines, the dynamic cell extends the traditional cell layout by allowing time sharing of machines with other cells that produce different part families. The dynamic cellular approach, whereby the principles of family based manufacturing can be implemented with the flexibility required to meet current demand patterns, has been recognised as an effective means of enhancing the efficiency of a discrete parts manufacturing shop. The design of such a dynamic cellular facility (*Figure 3.18*) can be formulated into three stages:

1. grouping of the production machines into cells,
2. allocation of the machine cells to areas within the shop-floor (intercell or facility layout), and
3. layout of the machines within each cell (intracell or machine layout).

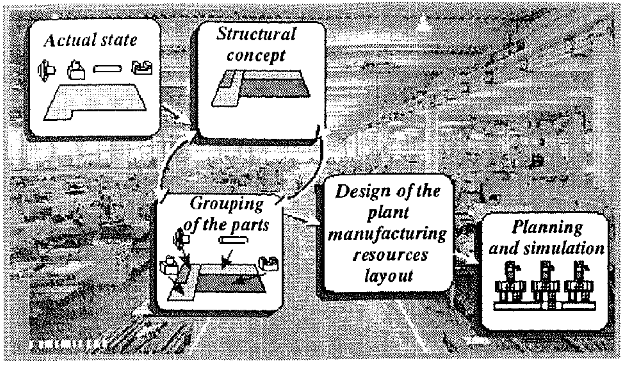


Figure 3. 18 - DCMSs layout design methodology

3.7. Capabilities of Research Methodologies

The suggested cell formation method must combine the capabilities of several analytical techniques. This is because the problems that have been discussed have different structures. The feasibility of some of the most useful analytical methods described in the literature review to solve these problems is presented in the following sections.

3.7.1. Operations Sequence Approach

The operations sequence approach groups parts, based on their operational sequences, whilst machine grouping is done simultaneously with part family formation. However, for a large number of parts, these methods impose a greater computational burden than traditional cluster analysis. Also, the clustering dendogram showing the part families cannot directly indicate the linear or branched flowline structure within a cell, or how the cells will be located with respect to each other in the shop. Whilst machine duplication is possible, the clustering dendogram fails to relate this machine sharing to the overall shop layout. Finally, these methods mainly address part families and intracell layout, whilst distribution of machines is partially addressed and intercell layout ignored.

3.7.2. Clustering Approach and Matrix Diagonalisation

Cluster analysis and matrix diagonalisation are useful for identifying the bottleneck (or shared) machines and parts (parts which use only one machine or only the bottleneck machines) and so on. The bottleneck diagonalisation can indicate the potential number of cells. However, the efficiency of these methods reduces if the clusters are not easily distinguishable, or when the intracell and intercell layout must be planned. The reason is because operation sequences, flow volumes and flow directions are ignored in the symmetric similarity coefficient matrix. The matrix representation of the groups is inappropriate for two-dimensional layout design. It fails to model travel distances or machine sharing requirements when cells can be placed adjacent to each other. Their emphasis on solving both machine grouping and part family formation with the same method, neglecting layout aspects, is inappropriate. To solve distribution of machines, a clear-cut knowledge of part family assignments is necessary. This is not always possible if a large number of shared machines encourages multiple family memberships for the same part. Finally, these methods solve part families but ignore distribution of machines, and intracell and intercell layouts.

Overlapping cluster analysis can handle multiple input matrices with similarity coefficients or symmetric travel charts. Given a particular number of clusters, the machines that must be duplicated amongst the cells, can be identified. Parts, which belong to several families, can also be identified using this method. However, the difficulty of this method is to determine the number of clusters desired and that the flow directions cannot be captured. Thus, the clusters cannot directly indicate the relationships between the intracell and intercell layouts if shared machines are duplicated amongst the clusters. Thus, this method can interactively solve part families and distribution of machines, but fails to relate them to the layout and handling issues in intracell and intercell layouts.

3.7.3. Graph Theoretical Approach

Graph theoretical methods can consider flow volumes and asymmetric travel charts to output intracell and intercell layouts simultaneously. The maximal spanning tree (undirected tree), strong components of a graph and planar graphs are extremely useful structures for representing machine groups and cellular layouts. The paths in the Maximal Weighted Directed Spanning Tree implicitly groups machines sharing high flow volumes, and ensures the shortest feasible travel distances. Since the arcs in a path are directed they automatically represent *CM* flowlines with minium distances for backtrack flows. Finally, distribution of machines, and intracell and intercell layouts, can be solved together, because intercell flows between adjacent cells or backtrack flows on the same flowline can be modelled. Limited machine distribution decisions can also be made using only the flow volumes. However, if intercell flows are observed between nonadjacent cells, then part families and distribution of machines for those parts must be solved using more detailed machining times for each part operation. The complete operational listing of the parts must be considered to solve part families completely. Finally, this method needs to be complemented by the operation sequences grouping and cluster analysis methods.

3.7.4. Mathematical Programming Approach

Mathematical programming methods are useful for planning the assignment of shared machines and parts experiencing intercell flows. They provide flexibility for modelling machine capacity constraints. The quadratic assignment problem structure can be used to fit a symmetric travel chart to a variety of layout configurations. It attempts to group machines connected by high flow volumes in order to minimise the travel distances for those parts. Furthermore, it can address the machine sharing and distribution problem after an initial layout and grouping of machines have been identified. The limitations of the *QAP* are that it assumes a symmetric travel chart, and is computationally intractable for a large number of machines. However, these methods can solve distribution of machines, and improve upon initial solutions obtained for part families, intracell layout and intercell layout by other methods.

3.7.5. Research Methodology

The proposed research uses a travel chart as input data because it is the only data representation which will allow the integration of the machine groupings, machine sharing and intracell layout and intercell layout problems for the distribution of machines. It uses mathematical programming models to combine the necessary graph theoretical programming models to encapsulated the required graph theoretical and combinatorial optimisation concepts. If independent cells exist, they will be indicated by the absence of intercell flows. Machine sharing for backtracking within a cell will be accommodated by the appropriate intracell handling system. An intercell layout of parallel flowlines will allow crisscrossing flows between adjacent lines for all relevant shared machines. Machine duplication for some intercell flows will be further avoided by permutating those flowlines to make them adjacent to each other across an intermediate aisle. This will leave the parts and machines involved in the remaining intercell flows within the nonadjacent flowlines. These proposals can be seen to reduce part families and distribution of machines.

3.8. Conclusions

In this chapter the objectives, which comprise the cell formation problem, are described. The interactions between these problems encourages the design of a dynamic layout to identify the machine groups, intracell and intercell layouts simultaneously. The capabilities of existing methods for solving these different problems are also discussed. The suggested method uses a combination of graph theoretical and mathematical programming models in order to have flow based cluster analytic capabilities. This method will be presented in the following Chapter 4, with an experimental study in Chapter 5. To improve the capability of the proposed research methodology some modern optimisation techniques (simulated annealing and evolution algorithm) will be utilised in Chapter 5. Finally, in Chapter 6 a proposed methodology, detailing a dynamic cellular layout's total cost will be optimised (minimised), resulting in a new optimised (applying *SA*) dynamic shop floor layout.

CHAPTER 4

Mathematical Modelling of Dynamic Cells Layout

4.1. Introduction

This section describes proposed mathematical model-approaches for flow decomposition of machine grouping and flowline layout design of dynamic cells. The theory of networks and graphs is applied to model this problem and some basic relevant definitions and notations from network analysis and graph theory texts [Gibbons 1985; Harary 1967, 1973 & 1980; Lawler 1976 & 1985; Aho 1974, 1980 & 1983 and Tarjan 1985 & 1983] are presented and used. Based on the network analysis and graph structures observed, a mathematical programming solution approach is developed.

Section 4.5.1 presents a programming model to minimise travel distances for Forward *(F)* and Backward *(B)* arcs, and section 4.5.2 discusses a mathematical programming model derived from section 4.5.1. This latter model optimises orientation of the weighted directed rooted tree (*WDRT*) in order to minimise travel distance and machine duplication for the Crisscrossing *(C)* arcs. In Chapter 6, section 6.1, a mathematical model is presented for economical machine duplication, whilst in section 6.2 an optimising a *DCM* layout from the stage three and design (machine placement) of the shop layout, using a Simulated Annealing algorithm *(SA),* is also developed.

It will be demonstrated that the sequential use of these models efficiently integrates the machine grouping and layout design issues discussed in the previous Chapter 3. To illustrate the model's use an example is presented in Chapter 5, and an approach for developing an approximate layout from the output is also explained. A review of graph theory and network analysis, justifying proofs and a review of some *NP*-complete problems is also discussed (in Appendix A).

4.2. Definition from Graph Theory

From the introduction to this chapter, which mentions network analysis and graph theory texts, a directed graph, denoted by $D(V,A)$, is a collection of n nodes, denoted by $v_1,.., v_n$ $(v_i \in V, |V| = n)$ and m arcs, denoted by $a_1,..., a_m$ $(a_j \in A, |A| = m)$, which join the nodes. An arc, denoted by a_{ij}, is an ordered pair of nodes (v_i, v_j) represented by an arrow connecting the tail (initial) node v_i to the head (terminal) node v_j (*Figure 4.1 and 4.2*).

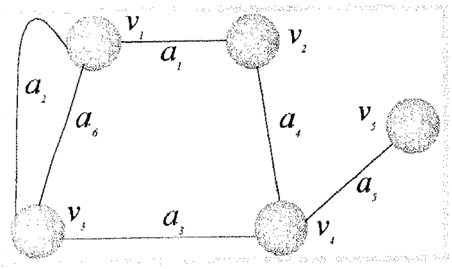


Figure 4. 1 - Undirected graph

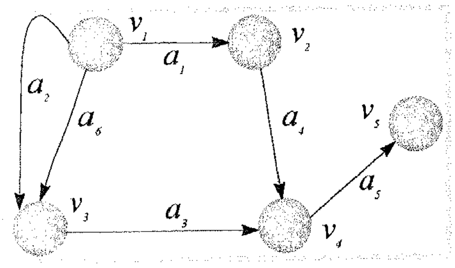


Figure 4. 2 - Directed graph

Any arc is said to be incident from node v_i and incident to node v_j. An edge joining the two nodes either has no direction or can be replaced by two arcs pointing in opposite directions from the node, a_{ij} or a_{ji}. A loop is an arc whose head and tail nodes are the same. The weight of arc (v_i, v_j), denoted as c_{ij} (cost), t_{ij} (time), f_{ij} (flow volume), represents a relationship between nodes v_i and v_j, taken in the direction v_i to v_j.

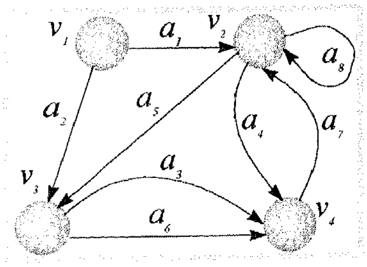


Figure 4. 3 - WDRT

A *WDRT* (*Figure 4.3*) denoted by $T(V_t, A_t)$, is a connected graph which contains no circuits (defined in Appendix A). By definition, if T is a *WDRT* of order m, then it contains $(m-1)$ arcs. It contains no circuits, but the addition of any arc creates exactly one circuit (or cycle), called a fundamental circuit (or cycle). If v_i and v_j are distinct nodes of T, then there is exactly one path between v_i and v_j.

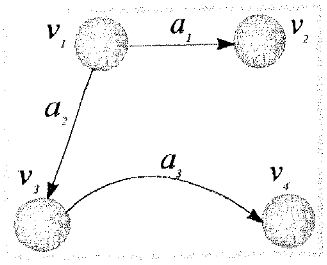


Figure 4. 4 - Corresponding-spanning WDRT

A spanning *WDRT* (*Figure 4.4*) of digraph $D(V,A)$ is a partial graph $D(V_t = V, A_t \subset A)$, ie. all pairs of nodes are connected by a path or chain. An out-tree is a *WDRT* in which all the arcs point away from a root node, which is the only node with zero in-degree. In the case of an in-tree, all arcs are directed towards a sink node, which is the only node with zero out-degree.

Undirected Graphs	Directed Graphs
Edges, *E*	Arcs, *A*
Vertices, *V*	Vertices, *V*
Graph, G (V, E)	Directed graph, *G (V, A)*
Loop	Loop
Mapping	Directed mapping
Adjacent edges	Adjacent arcs
Parallel edges	Strictly parallel arcs
Isomorphic graph	Isomorphic graph
Edge sequence	Arc sequence
Length	Length
Chain	Path
Circuit	Cycle
Simple chain	Simple path
Simple circuit	Simple cycle
Connected	Strongly connected

Table 4. 1 - Terminology for directed and undirected graphs

Table 4.1 shows the terms used in undirected graphs and the corresponding terms in directed graphs. Detailed reviews of graph theory and network analysis terminology, and definitions used in this research are presented in Appendix A.

4.3 Arcs Classification in the *WDRT*

Developing of this classification is in order to apply the concepts of Carrie [1975 and 1976] and Hillier [1963] on unidirectional flow line design to intercell layout design. The initial design of undirectional flow lines is necessary because only then can backtrack flows be identified. Their work concerned the design of the optimum layout sequence for a number of machines through which a variety of products is processed. Since they considered a single cell, they classified the flows along the flow line as in-sequence, bypass and backtrack. The heuristics proposed by Hillier used objective functions, which are maximisations of the number of in-sequence movements and minimisations of the amount of backtracking on the line.

Hillier's work helps to incorporate the aspects of material handling system capability into the layout design problem. Carrie enhanced these heuristics in a computer program, which adopted a variety of strategies, such as machine duplication at multi-functional stations, alternate routing of operations and allowance for material handling to minimise flow backtracking on the line. With respect to intracell layout, their classification of the flows is sufficient. Thus a conveyor connecting the machines in the flow line can provide for easy and rapid in-sequence and bypass flows. Backtrack flows require the services of a material handler for which queuing might be involved, since this would take more time than in-sequence moves. Carrie's machine duplication decisions were limited to a single cell. However, when applied to intercell layout, an additional type of arc is necessary in order to classify the flows which may occur between any pair of flow lines. When intercell flows are also considered, it is more important to minimise the intercell flows, since these will have even higher queuing delays for the parts involved.

Intracell machine duplication and throughput delays then become secondary compared to intercell flow delays and machine duplication.

Classification of all possible flow arcs in the digraph by various authors are as follows:

- an asymmetric Travel Chart for a "shop", also suggested by Aho [1974 and 1983],
- dominators in a cyclic digraphs, suggested by Ahuja [1974 and 1993],
- strong components in a digraph, suggested by Tarjan [1977], and
- optimum branching in a digraph, suggested by Tarjan [1977 and 1985].

The machine grouping, intracell flowline layout, and intercell layout planning problems can be integrated through the structure of the Maximal Weighted Directed Spanning Tree (*MWDRST*). This suits the in-sequence, bypass and backtrack classification (for intracell flows) and crisscrossing (for intercell flows) classification of flows in a cellular system. Hence, the *MWDRST* was preferred over several undirected graph structures such as cut trees Montreul [1989], maximal spanning trees [Carrie 1975, Heragu 1989 &1990], minimal spanning trees [Srinivasan 1994] and maximal planar graphs [Carrie 1978, Foulds 1985], used to solve the more general problem of facilities (or only intercell) layout design. The *MWDRST* has the clustering property of the maximal spanning tree, and it gives the flowline layout for the group of machines in each path. Since nodes in the *MWDRST* can have the outdegree greater than one, this combines the concepts of a functional layout with a tree layout for the shop floor when flowlines sharing machines are merged. Owing to its directed tree property, it conforms to the in-sequence bypass or backtrack (for intracell flows) and crisscross (for intercell flows) categorisation of flows in the travel chart for the shop floor. Being a planar graph, it allows a permutation of its branches at the branching nodes, such that the distances corresponding to high volume intercell flows can be minimised after the initial tree layout is obtained. From the above discussion, the complete set of arcs *A* in *D(V, A)*, other than those in *T*, can be decompressed into two sets (explained in more detail in Appendix A):

$$A_t = \{a_{ij} \in T\}, \text{ and } A_{\bar{T}} = \{a_{ij} \in \bar{T}\},$$

Based on the paths *T*, the chords in $\bar{T}$ (Appendix A) can be classified into one of three sets: *F* (forward), *B* (backward) and *C* (crisscrossing) arcs, where $F \cup U \cup C = A_{\bar{T}}$. In each class the arcs can be further classified as necessary or redundant, depending upon whether they necessitate machine duplication or not, respectively. If intracell flow delays are considered smaller in magnitude than intercell flow delays, then layout and handling solutions can make machine duplication for the (intracell) arcs in $F \cup B$ (intracell) flows redundant. This would ease the machine duplication problem for the necessary arcs in *C* (intercell) flows, ie. arcs connecting nonadjacent paths in the *WDRT*. The following sections will explain in more detail this classification.

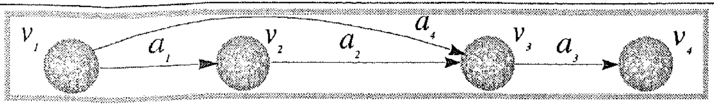


Figure 4. 5 - Tree arcs and only forward arcs in WDRT

4.3.1. Forward Arcs

An earlier definition for the length of a path has been the sum of the weights of the k (part or pendant node) arcs it contains. A k - path between nodes v_i and v_j on some path in T, denoted by p_{ij}^k, where ($v_i \in p_{ij}$ and $v_j \in p_{ij}$), is a path with k arcs (edges). Any arc $a_{ij} \in T$, or $\in D(V,A)$, is therefore a p_{ij}^k. However, if node v_j is reachable from v_i by a path in T containing two or more arcs, then $a_{ij} \in \overline{T} \Rightarrow \exists p_{ij}^k$ for $(n-1) \geq k \geq 2$ in T. This could make a_{ij} a redundant forward arc in $\overline{T}$, as well as $D(V, A)$, if the length of the path does not correspond to a significant handling delay between the consecutive machines. There may be need for machine duplication to reduce travel time for the flow f_{ij}, since it can simply bypass all intermediate machines in the existing flow line or path in T from v_i to v_j. It can also be said that the forward arcs (*Figure 4.5*) on any path in T makes the path a unilateral component, because an arc and at least one path from the corresponding set, connects each pair of nodes.

Carrie's work resorts to intracell duplication of identical machines in several cells to eliminate backtrack flows. This is subject to sufficient machine utilisation being ensured in all cells. However, the critical assumption made in his work is that the intercell machine duplication problem does not constrain the machine availability within the particular flow line being designed. In contrast, this proposed research gives a higher priority to the elimination of intercell flows by machine duplication.

The set of arcs in F can be classified as redundant $(a_{ij} \in F_r)$, or necessary $(a_{ij} \in F_n)$, where $F_r \cup F_n = F$. F_r contains arcs that do not merit machine duplication. The distance that parts included in these arcs must flow, can be made less than k by a layout adjustment, or by using a suitable handling system. From a layout perspective, a U-layout effectively can halve the length of the initial flow line based on the path connecting machines i and j, without upsetting the previously unidirectional flow. Instead of placing all machines in a line, as indicated by the paths in T, it is suggested that machines in the modified layout are placed on both sides of a central aisle separating two shorter flow lines. In any new layout, parts can move across the aisle to a machine in the opposite flow line instead of having to bypass several machines, when a pure linear unidirectional flow line layout is used. This modified layout prevents intracell machine duplication without increasing the time that parts travel between nonadjacent machines for consecutive operations.

However, conversion from a pure linear unidirectional intracell flow line layout to a L, U, S, W or Z shaped layout reduces the intercell accessibility to shared machines. Issues such as:

- how many machines in each flow line,
- locations at which these flow line direction changes, and
- left or right directions in which each machine should be located,

must also be addressed.

Theoretically, several traditional *NP*-complete problems must be solved together, in order to consider the effects of intracell and intercell flow distances simultaneously, because intracell machine locations and intercell layouts must be determined together. This research assumes that intracell flow delays due to flowline layout, are preferable to the higher intercell flow delays due to reduced machine adjacencies when having L, U, S, or Z-shaped layouts for cells.

The arcs in F_n correspond to k-path arcs, where $k \geq k_{optim}$. Thus, if an arc in F corresponds to a path in T with $(k_{optim}+ 1)$, or more arcs, then travel times can become excessive if machine duplication is not done. Optimal values of k depends on several factors, and in addition, it makes the integrated intracell and intercell layout problem the standard two-dimensional facility layout problem. This has been proved to be difficult to solve for a large number of departments (or machines), when their locations are not known previously.

Forward arcs corresponding to paths in T with higher k values can also belong to F_r. As an example once issued from, say, the raw material store, batches of parts included in arcs from F_r must traverse the k-path distance, without queuing at the intermediate nodes. There is no need to duplicate the machines in the tail nodes of these arcs in order to bring them adjacent to the machines in the head nodes, or vice-versa. It will not reduce material handling times for those parts. Hence, it is possible to avoid machine duplication for arcs in F by making $F_n = \Phi$ (explained in more detail within Appendix A.2). For example, it is claimed that "anywhere from one to fifteen machines are grouped together, typically in a U-shaped layout" [Sule 1988 see p122] giving k as high a value as *14*, if only linear flows are allowed. With the developments in automated handling systems, network-type flow path configurations could be encouraged, thereby allowing higher k values. This argument supporting the use of larger k values to define arcs in F_r can be seen in the layouts where all identical machining centers have been placed together in a functional layout, with no intracell machine duplication, thereby supporting the new *CM* philosophy of dynamic cells. Secondly, this type of layout configuration will minimise the intracell flow delays without allowing machine duplication. Hence, if machine duplication for arcs in F are avoided, it will leave more machines for eliminating intercell flows.

Another reason for avoiding machine duplication for arcs in F_n is that, even after machine duplication, the additional flow line will essentially still remain adjacent to the existing flow line from which it was derived. Hence, by eliminating the earlier functional layout with all identical machines located in one cell, intracell machine utilisation would reduce. So, machine duplication for arcs in F_n are preferable avoided, since the intracell flows can be managed by a handling system.

Machine duplication for arcs in F can also be avoided by replacing the existing conventional machines with *CNC* machines (key machine concept), and thus travel times for arcs in F_n will be eliminated by introducing *CNC* machines which perform consecutive operations on the same part. The effective length of the flowline, originally comprised of only conventional machines, is then drastically reduced. Based on these arguments, machine duplication for arcs in F need not be considered during the layout design stage. Furthermore, if *WDRT* structures for the shop layout and flow line layouts for each cell are assumed, then these arcs will not contribute to the intercell flow delays. Only if L, U, S, W or Z-shaped intracell layouts are desired, will the lengths of these arcs feature in the two-dimensional layout problem.

4.3.2. Backward Arcs

All backward arcs (*Figure 4.6*) on a path in T are chords, which causes the paths in T to become strong components or circuits, when these arcs are added. In order to avoid machine duplication to reduce travel times for the flows in these arcs, layout as well as handling solutions must be developed. Similar to the ideas developed for forward arcs, a backward arc (v_j, v_i) on a path p_{ij}^k in T must also traverse a distance of k edges instead of arcs, i.e. it is equivalent to a k-chain, since the k-path in T is traversed in the reverse direction (a_{ji} points in the reverse direction from p_{ij}^k). Hence, B_r will contain all arcs which convert k-paths into circuits, or strong components, for the specified limiting value of k, where $B_n = (a_{ij} \in \overline{T})$ such that T contains a p_{ij}^k .

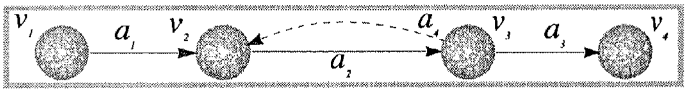


Figure 4. 6 - Tree arcs and only backward arcs in WDRT

The need for intracell machine duplication to eliminate the travel times for arcs in B may not be necessary, as discussed earlier for the forward arcs. Again, the handling capabilities of existing *FMS's* suggests that the intracell handling delays caused by these arcs is not a problem and can accommodate such intracell flows. Handling alternatives to eliminate backtracking and to prevent machine duplication within a flow line, can also be implemented to leave machines for eliminating flows arcs in C_n (crisscrossing arcs). Hence, this further justifies grouping all identical machines at one station in a *CM* flow line in the presence of backtracking.

The common representative patterns of workflow clearly indicates that loop flow patterns are viable within a cell. This pattern would also eliminate intracell machine duplication for backtracking flows within a cell, which is identical to the sequences of machines observed in the *WDRT*. Hence, the *WDRT* structure can be exploited for intracell as well as intercell layout design. As in the case of forward arcs, machine duplication for arcs in B_n will result in the additional flow line being adjacent to the existing flow line. Hence, it would be preferable to retain the duplicated machines in their functional sections, using handling systems to reduce the handling delays due to backtracking.

4.3.3 Crisscrossing Arcs

All crisscrossing *(C)* arcs *(Figure 4.7)* contained in $\overline{T}$ are chords, whose addition to T will create a fundamental circuit (or cycle) which links two paths in T. The flows on these arcs in C constitute the intercell flows which, as per the existing literature, receive the highest priority for elimination by machine duplication. These arcs could be further classified as redundant if necessary. Redundant arcs would correspond to those connecting adjacent paths in the *WDRT*, whereas the necessary ones would be spanned at these paths. The arguments for avoiding machine duplication for the redundant arcs are similar to those given earlier for *F* and *B* arcs.

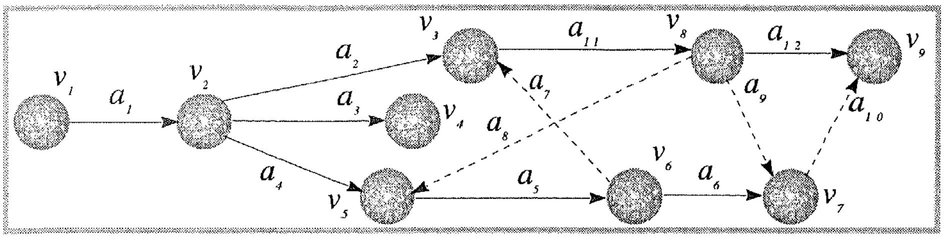


Figure 4. 7 - Tree arcs and only crisscrossing arcs in WDRT

For some necessary arc a_{ij}, in order to avoid the delays due to intercell flows, the machines to be duplicated could be:

1. node v_i and all the predecessors of node v_i in the operation sequences of parts which contain arc a_{ij} placed next to the flow line containing node v_i, or
2. node v_j and all the successors of node v_j in the operation sequences of parts which contain arc a_{ij} placed next to the flow line containing node v_j.

However, as has been emphasised, any kind of machine duplication will lead to utilisation and load balancing problems, and must be avoided. It is impossible to identify an integer allocation of machines to cells without utilisation problems, if the loads for the part families are equivalent to noninteger machine requirements. An approach to this problem, in the case where machine exists for distribution (duplication), is described in Chapter 5.

4.4. Problem Complexity

The development of a single mathematical programming model for the combined machine grouping, intercell and intracell layout design problems have proved difficult. For intracell flow line construction no single model has been able to incorporate all aspects of machine grouping which considers:

- generation of all flow lines in the form of a *MWDRST*,
- interaction between an intracell flow line configuration (flow distance for forward and backward arcs) and an intercell layout (flow distances for crisscrossing arcs),
- machine sharing for nonadjacent crisscrossing arcs, and
- optimal left to right order of the flow lines.

Thus it is proposed that the required model be a combination of cluster analysis, constrained spanning *WDRT*, permutation generation and two-dimensional facility location problems. The brief review of existing *NP*-complete problems (presented in Appendix A) will show that each of the problems listed in the last paragraph is itself a separate *NP*- problem (as normally abbreviated for non-deterministic polynomial time complete problems). However, a common property for *NP* complete problems is that no polynomial time algorithms are known for the problem. Thus if one of these problems is solvable in polynomial time, then all the problems are solvable in polynomial time.

Stages	*Steps*	*Main Tools*	***Description of Steps***
I	1	*Matrix form*	*Input* 1. Batch quantity for each part *Qk* 2. Operational sequences for each part *Sk* 3. Number of machines of each type available *Ni*
	2	*MATLAB matrix form*	Compute the distance between every pair of machines *i* and *j* and construct the distance matrix
	3	*MATLAB*	Generate initial digraph *D(v,a)*
	4	*MATLAB Chu's algorithm*	Solve first stage to generate *T* Maximum Weighted Directed Rooted Spanning Tree *(MWDRST)*
	5	*Tree draw*	Generate *path (1)... path(p)* in the *MWDRST*
	6	*Tree draw*	Identify arcs in forward, backward and crisscrossing paths of the digraph *D*
II	7	*MATLAB matrix form*	With crisscrossing paths and *path(1)...path(p),* generate combined interpath flow matrix using total flow for machines *i* and *j*, and weighting for the shared machines
	8	*MATLAB*	Solve second stage to pivot paths – to rearrange *MWDRST*
	9	*MATLAB matrix*	Plan layout based on material flowlines from second stage and path adjacencies
	10	*Tree draw*	Identify arcs in crisscrossing to plan for machine duplication in nonadjacent cells
	11	*MATLAB*	Design and analysis of approximate *DCMS's* flowline layout
III	12	*MATLAB*	Economical capacity oriented machine duplication
IV	13	*MATLAB SA*	Economical optimisation of *DCMS's* layout using *SA* algorithm minimising total material handling and duplication cost's
	14	*MATLAB SA*	Shop floor layout displacement

Table 4. 2 - Initial strategy overview of the proposed research methodology

4.5. Mathematical Formulation of the Problem

The definitions presented in the previous sections (networks and graph theory) are sufficient to give an overview of the mathematical formulation and models for the proposed dynamic cell layout design and flowline formation method, presented in *Table 4.2*. This problem is solved in four stages:

- Stage I is a mathematical programming model to generate a Maximum Weighted Directed Rooted Spanning Tree (*MWDRST*). This implicitly minimises travel distances for Forward *(F)* and Backward *(B)* arcs.
- Stage II is a mathematical programming model derived from stage I. It finds the optimal orientation of the *WDRT* in order to minimise travel distances and machine duplication for the Crisscrossing *(C)* arcs.
- Stage III is a mathematical model for economic duplication of machines from the stage II approximation layout and will be explained in Chapter 6.1.
- Stage IV is the placement of machines from the approximate flowline layout from stages II and III (shop floor layout) to generate the final *DCM* layout, applying the combinatorial optimisation *SA* algorithm. This will be discussed in Chapter 6.2.

These four main stages of the design methodology for the *DCM* flow line layout, just discussed, are further elaborated in the next sections, focusing particularly on the first two stages. The proposed stages III and IV are presented in Chapter 6. Also presented in the next chapter is an illustrative example, and an experimental study with an improved analytical procedure. A detailed mathematical presentation is given in Appendix A.

4.6. Modelling of Stage I: Steps 1 - 6

In this section, application of the graph theory, utilising Edmond's optimum branching algorithm (dual solution) to a *CM* layout is described. Chu [1965], Edmonds [1967] and Bock [1971] have independently devised an efficient algorithm to find an "optimum branching" (minimum spanning tree) in a directed graph. Chu's [1965] proof was entirely graphical; Bock [1971] solved the problem using linear programming theory and Karp [1972] gave a further proof of Edmond's [1967] algorithm, which does not depend on linear programming theory. Tarjan's [1977 and 1985] modified branching algorithm gave a new improved running time of $0(n^2)$. Bock [1971] was the first who described the optimum branching problem, and gave formulation (minimum directed spanning tree - primal problem) to the integer mathematical programming form (using matrices) as shown in *Equation 4.1*, and also suggested a procedure for dual (maximum) problem solution. This algorithm (Bock's) was the initial concept in this approach for the mathematical description of the model:

$$Minimise \quad z = \sum_{i,j \in S} c_{ij} x_{ij} \qquad \textit{Equation 4. 1}$$

Bock's objective function *(Equation 4.1)* is defined for S as the set of positive integers $1 \ldots n$, for a specified n (with S_k as subset of S having n_k elements) and Q as a set of indices of all the proper subsets of S. The indices i and j denote the initial and final nodes respectively of the directed links ij of a given network. Equivalently, i and j are row and column indices of a square matrix and c_{ij} are the link values. The variable x_{ij} has the value of 1 if the corresponding link is in the solution set and 0 if not. An example for this approach is also included in Bock's work.

In the proposed approach of this research *Equation 4.2* constitutes the first stage, which is a mathematical programming model for obtaining T, a maximum weighted directed rooted spanning tree *(MWDRST)*. This proposed model is different from the original model described by Bock [1971] who presented *Equation 4.1* for minimisation of a directed spanning tree. Thus:

$$Maximise \quad \sum_{j \neq R} \sum_{i \neq S} f_{ij} w(p_{ij}) x_{ij} \qquad \forall \; a_{ij} \in T \qquad \textit{Equation 4. 2}$$

where:

f_{ij} - $\Sigma Q_k \; \forall S_k \mid a_{ij} \in S_k \;\; i \neq S; j \neq R$ - flow from machine i to machine j (the sum of the batch quantities of all parts whose operation sequences contain machines i and j),

$w(p_{ij})$ - 1, if - $a_k \in T$ weight (length) of the path p_{ij},

X_{ij} - binary variable to represent the membership of arc a_{ij} in T,

p_{ij} - path in T tree connecting machine i to machine j,

R, S - common root (eg. raw material store) and sink (eg. finished goods store) nodes of the digraph

D(V,A) (travel chart) occurring in the operational sequences of all parts,

a_{ij} - directed arc from machine *i* to machine *j*,

S_k - operational sequence of part *k*, represented as *(R, 1, 2, 3, .., n-1, n, S)*, where *i* is the machine required for the *i-th* operation on the part,

Q_k - batch quantity for part *k*, and

T - *MWDRST* generated by the (linear) programming model in the first stage, (also denoted as *D(Vt,At))*.

The objective function given by *Equation 4.2*, is different than Bock's original definition, because if the mathematical programming model is a linear programming model, then this *Equation 4.2* is a dual problem of Bock's model. As presented the objective function in *Equation 4.2* maximises the linear function of flow *(f)* and weight *(distance w)* between machines *i* and *j* of the material handling in the flow lines.

The proposed model from stage I is programmed in *LINGO* together with the main algorithm (Chu's 1965), which is used later and programmed (Appendix D) in *MATLAB*. The results from both programs are the same as that to be presented in the illustrative study and Chapter 5. However, it should be noted that Chu's algorithm originally was given in graphical form for the minimum spanning tree (running time improved by Tarjan [1985]) and was modified for *MWDRST* and coded in *MATLAB*. Interestingly the reader can refer to Chu's original paper in this area of work, which has been already mentioned before. This and the proposed mathematical model are explained in more detail in Appendix A.

Now, after the flow line distances are maximised *(MWDRST path)*, the next step is to solve the problem of how to optimise (pivot) paths around a *MWDRST*. Optimisation of the flow line paths orientation is explained in the next section with the proposed mathematical model.

4.7. Modelling of Stage II: Steps 7 - 11

This section describes the mixed integer programming model, which finds the optimal permutation of the branches of *T* to minimise intercell flow distances with adequate definition and terminology. Further detail description of the stage two mathematical model is presented in Appendix A. The proposed model for this stage is derived from Love's model for the two-dimensional location for Euclidean distances [Love 1973], originally used for solving the *QAP* [Love 1976] (based on the Gilmore-Lower approach for solving *QAP)* as shown in *Equation 4.3* It is here adapted to solve the Optimal Linear Arrangement.

Thus:

$$\textit{Minimise} \quad \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} w_{ij}\left(R_{ij}+L_{ij}+A_{ij}+B_{ij}\right) \qquad \textit{Equation 4. 3}$$

where:

n - number of facilities and number of locations,

w_{ij} - nonnegative weight or flow between facility *i* and facility *j*,

R_{ij} - horizontal distance between facility *i* and *j*, if facility *i* is to the right of facility *j*,

L_{ij} - horizontal distance between facility *i* and *j*, if facility *i* is to the left of facility *j*,

A_{ij} - vertical distance between facility *i* and *j*, if facility *i* is above facility *j*, and

B_{ij} - vertical distance between facility *i* and *j*, if facility *i* is below facility *j*.

Interestingly, the reader can refer to Love's paper or to Francis [1976] and Karisch [1995 and 1998] for more information about *QAP* methodology and examples. However, a second stage mathematical formulation of *Equation 4.4* is presented as follows:

$$\text{Minimise} \quad \sum_{i=1}^{p-1}\sum_{j=i+1}^{p}\{F_{ij} + \sum_{k\in P_i\cap P_j} W_k\}(R_{ij}+L_{ij}) \qquad \textit{Equation 4. 4}$$

where:

P - number of paths corresponding to the pendant nodes in the *T (MWDRST)* tree,

F_{ij} - $\sum_{l}\sum_{k} f_{kl}$ $\forall\, k \notin P_i \in P_j$, $\forall\, l \in P_i \notin P_j$, $a_{kl} \in C$; sum of flow volumes of arcs that connect paths P_i and P_j (such that either the head or tail node of each arc lies on P_i or P_j),

W_k - flow weight assigned to a machine common to paths P_i and P_j,

R_{ij} - column difference between P_i and P_j, if P_i is to the right of P_j,

L_{ij} - column difference between P_i and P_j, if P_i is to the left of P_j, and

a_{ik} - assigned position *k* (path position) in the optimal sequence of pendant nodes.

Stage II was felt necessary because the progressively increasing weights used successfully in the examples solved later were obtained iteratively from trial runs for each example. As illustrated in later examples, during the permutation of the branches of *T*, two critical tasks need to be performed. The first task is necessary, although it does not influence intercell flow distances. It ensures that if a pair of paths shares paths, which share the machines at the branching nodes, then such a shared path will always permute as a group, even if the machines are at two or more levels. The second task is to permute the branches without violating the restrictions imposed during the first task. This task minimises intercell flow distances without allowing any machine sharing.

The successful use of this restricted permutation strategy will be shown in later examples, and also in an experimental study for the second stage. However, later in Chapter 5 and Appendix A, it will be seen that in the worst case the one dimensional optimal linear arrangement becomes a two dimensional *QAP,* which is a *NP*- problem. Traditionally, an exact solution of the *QAP* is computationally infeasible for fifteen or more machines [Francis 1974], necessitating the use of pairwise-interchange heuristics. This is why a *QAP* steepest descent pairwise interchanged heuristic, coded (Appendix D) in *MATLAB*, is applied in Chapter 5 (An Experimental Study).

4.7.1. Evaluation and Analysis of Approximate *DCMS* Flowline Layout

The *DCMS* flowline layouts need to be evaluated and analysed, as the main criteria for layout evaluation is kept simple to focus more on the qualitative aspect, which is a comparison of the total distances moved by all the parts in both intercell and intracell layout configurations. The objective function is presented by *Equation 4.5* as follows:

$$\Sigma f_{ij}\, d_{ij} \qquad \forall (i,j) \qquad \textit{Equation 4. 5}$$

where f_{ij} is the flow volume between machines i and j in the *DCMS* layout, and d_{ij} is the distance between machines i and j. The distance can be rectilinear, Euclidean, or along the shortest path through a flow network. This criterion for *DCMS* layout analysis for each approximate layout evaluation allows a near feasible solution during the intermediate stage solutions (second stage).

This is a simple but effective measure of the handling and layout capabilities of the proposed method. That is, the sum of the lengths of the flow arcs in the new layout configuration, weighted by their flow volumes, have to be evaluated for comparison with those shown for the original layout. This was achieved using a mathematical model build in *MATLAB* with certain known assumptions, such as rectilinear distances, unit lengths and widths for all the departments, and equal department sizes for all the layout configurations assessed.

4.8. Mathematical Justification of the Methodology

This section contains the mathematical justification of the methodology presented in the first two stages. Here the use of the *MWDRST* is compared to the Optimal Linear Arrangement *(OLA)* or the Directed Optimal Linear Arrangement (*DOLA*) analogue for cell formation and layout design (detail justification is shown in Appendix A).

The *OLA* is analogous to the Bond Energy Algorithm [McCormick 1972] or *TSP* for a symmetric chart. The *DOLA* of the travel chart is equivalent to a permutation problem or *TSP* for an asymmetric cost matrix. The linear arrangement of the machine groups that the *OLA* presents does not capture the two dimensional flow and adjacency structure of layout problems. Aspects such as machine sharing by multiple paths at a branch node, or parallel flowlines to avoid intercell machine duplication, cannot be considered by them. With the *MWDRST*, the cluster analysis properties of *MWDRST* are obtained together with the flow structure in each group. Being a planar graph, it can also be rearranged to minimise intercell flow distances.

An aisle in the actual layout can separate each pair of adjacent paths in the *MWDRST*. This would allow intercell crisscrossing or intracell forward or backward flows without machine duplication. Machines in adjacent flowlines can be grouped to form "dynamic" cells. Finally, it will be shown (Appendix A) that the average travel distances in the *MWDRST* are less than in a *DOLA*, and that the mathematical formulation justification is based on network analysis and graph theory texts [Gibbons 1985, Harary 1967, 1973 and 1980].

4.9. Summary of Initial Methodology for the Stages I and II

This section summarises the steps in the first two-stage solution strategy adopted for the proposed dynamic cell formation and layout methodology shown in *Table 4.3*. The feature of this part of the research work was to capture the directionality embedded in the operation sequences of a variety of parts produced in a facility for the *DCM* layout design. Also, to consider how to generate machine groups by identifying a flowline layout for each group, thus indicating which flowlines must be placed adjacent to each other to minimise intercell flow distances, and furthermore to determine an approximate configuration of the aisles.

Stages	*Steps*	*Main Tools*	*Description of Steps*
I	1	*Matrix form*	*Input* 1. Batch quantity for each part *Qk* 2. Operational sequences for each part *Sk* 3. Number of machines of each type available *Ni*
	2	*MATLAB matrix form*	Compute the distance between every pair of machines *i* and *j* and construct the distance matrix
	3	*MATLAB*	Generate initial digraph *D(v,a)*
	4	*MATLAB Chu's algorithm*	Solve first stage to generate *T* Maximum Weighted Directed Rooted Spanning Tree *(MWDRST)*
	5	*Tree draw*	Generate *path (1)... path(p)* in the *MWDRST*
	6	*Tree draw*	Identify arcs in forward, backward and crisscrossing paths of the digraph *D*
II	7	*MATLAB matrix form*	With crisscrossing paths and *path(1)...path(p)*, generate combined interpath flow matrix using total flow for machines *i* and *j*, and weighting for the shared machines
	8	*MATLAB*	Solve second stage to pivot (permute) paths – to rearrange *MWDRST*
	9	*MATLAB matrix*	Plan layout based on material flowlines from second stage and path adjacencies
	10	*Tree draw*	Identify arcs in crisscrossing to plan for machine duplication in nonadjacent cells
	11	*MATLAB*	Design and analysis of approximate *DCMS's* flowline layout

Table 4. 3 - Research methodology strategy: stages I and II

4.9.1. Stage I: Steps 1 - 6

Here the operational sequence and batch quantity of the parts is the only input data. To obtain the *MWDRST* the travel chart obtained from this data is the input to the Stage I model. Using the *MWDRST*, the forward and backward arcs in the original diagraph are eliminated.

4.9.2. Stage II: Steps 7 - 11

With the crisscrossing arcs only, a matrix showing the intercell flows amongst all pairs of paths in the *MWDRST* can then be developed. This matrix is the input to the *MWDRST* second stage model to permute the branches of the *MWDRST*, without dividing any of the branching nodes.

After these steps, additional steps (illustrated in Chapter 5) are required to:

1. identify all the parts whose operation sequences contain the arcs in C_n,
2. identify all the machines that must be shared,
3. obtain the setup and operational time data for each operation on a part,

4. solve a mathematical programming model to determine the assignments of shared machines to the competing cells, and
5. evaluation and analysis of an approximate *DCM* flowline layout (presented in section 5.7.1).

For mathematical modelling, *MATLAB* and *LINGO* (mathematical modelling languages) were used. This provided an environment to develop, run and modify the mathematical models, since *LINGO* is a powerful matrix generator. The program provides all the features required to generate a model in a simple, efficient, intuitive manner. *LINGO* allows the expression of a model using conventional mathematical terms such as subscript variables, sets, operations and general expressions. In addition, the user has an extensive library of mathematical, probability and financial functions, and the formulation can also include extensive comments within the model. Most importantly, *LINGO* allows syntax that is much more liberal than when expressing formulas.

The main programming tool used in this research was *MATLAB*, which is another powerful matrix generator. Using *MATLAB*, all the mathematical computing was done, which included coding and building of algorithms and heuristics for *TSP, QAP*, the Branch and Bound method, and the Mixed Integer program. This also includes combinatorial optimisation techniques, *SA* and *EA* (Evolution Algorithm) for the *TSP,* and shop layout displacement. At this stage it may be noted that results from these programs were initially tested and compared satisfactory with the operation research package *(STORM [1992]).*

4.10. Computational Experience

At this stage the first two stages of the proposed research were tested. The primary goal was the *NP* – complete problem structure of the second stage, which was a main computational bottleneck, and not computational efficient. The optimal solutions for the second stage, obtained using an integer mathematical model, was validated with good results by utilising a *LINGO* model. Since the stage one model is a linear programming model with a network structure, solutions were obtained very quickly for all travel charts applied. However, the second stage problem, with more then 4 nodes, required much longer time (a hour or more) to obtain the optimal solution. Even when the second stage model was solved with a new *MATLAB* mixed integer program, times of more then 30 minutes were recorded. This result clearly is not suitable for many industrial scenarios having non-stable product or dynamic production environs.

As referenced in the experimental study and improved procedure Chapter 5, because this methodology obtains valid intracell and intercell layouts for the first stage, to obtain the *MWDRST,* the same algorithm can be used. However, to improve the computational time of the second stage model, the steepest descent pairwise interchange heuristic (*SDPI)* was applied (some initial experimental results for the *SDPI* are shown in *Table 4.4*). It may be noted that 16 pendant nodes (problem size) for the second stage *OLE* model permutation could be equivalent to a medium size company operating with between 50-100 machines *(Table 4.4)*.

Stage 1 (problem size)	*Stage 2 (problem size)*	*Stage 2 Running time (sec)*
14	4	1
12	6	2
14	8	3
20	10	4
25	16	5
40	24	8

Table 4. 4 - Computation time when SDPI algorithm utilised in the second stage model

At the time of formatting this research methodology it should be mentioned that a similar result from literature has only recently been available. Karisch [1998] compared results from the steepest descent pairwise interchange heuristic (*SDPI*), with respect to *EA* and *SA* methods, when applied to the *QAP* problem. Again, the *SDPI* outcome still gave better performance then the other two methods. Following Karisch's positive result the *QAP* steepest descent pairwise interchange heuristic was coded (Appendix D) in *MATLAB* and is included in Chapter 5 as a part of the experimental study.

4.11. Conclusions

In this chapter the initial design procedure with mathematical modelling of the *DCM* layout, including the first two stages, is presented. The goal of this part of the research was to capture the directionality embedded in the operation sequences of a variety of parts produced in a facility for the *DCM* layout design. This goal was achieved by incorporation of directionality into the design flowline skeletons layout using a Maximum Weighted Directed Rooted Spanning Tree (*MWDRST*), which is a rooted and directed tree composed of directed flow paths linking departments. Hence it incorporates the first principle for design of the facility layout, which is the maximisation of directed flows. *MWDRST* is a planar graph concept, hence planar embedding is possible and automatically penalises backtrack flows. These are the two main features, directed flow maximisation and backtrack flow minimisation, which characterises the *MWDRST* concept.

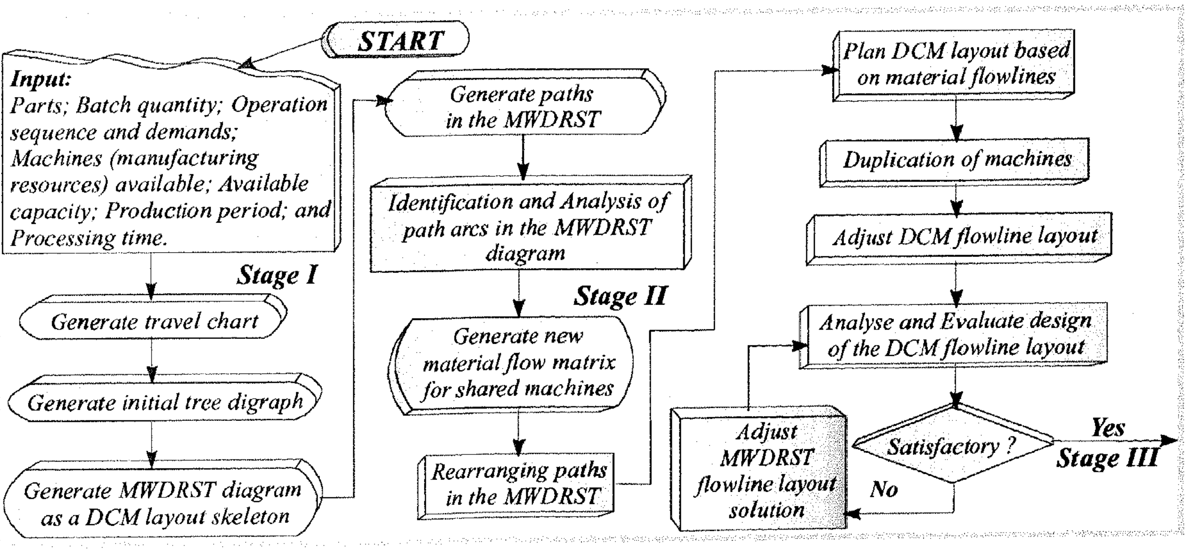

START
Input:
Parts; Batch quantity; Operation sequence and demands; Machines (manufacturing resources) available; Available capacity; Production period; and Processing time.
Stage I
Generate travel chart
Generate initial tree digraph
Generate MWDRST diagram as a DCM layout skeleton
Generate paths in the MWDRST
Identification and Analysis of path arcs in the MWDRST diagram
Stage II
Generate new material flow matrix for shared machines
Rearranging paths in the MWDRST
Plan DCM layout based on material flowlines
Duplication of machines
Adjust DCM flowline layout
Analyse and Evaluate design of the DCM flowline layout
Satisfactory ?
Yes
Stage III
No
Adjust MWDRST flowline layout solution

Figure 4. 8 - An interactive (stages I and II) DCM flowline layout framework based on graph MWDRST theory

A first two-stage mathematical programming approach was used to generate the *MWDRST*. In stage I, a linear programming model generated the *MWDRST* from the travel chart. In stage II, using only the crisscrossing arcs as input to the 0 - 1 integer- programming model, an optimal planar embedding of this *MWDRST* was obtained.

The approach presented shows that the suggested method is a unique combination of network analysis, graph theory and mathematical programming concepts. Its fundamental assumption is that machine sharing on a part family basis creates load-balancing problems. Furthermore, shared machines could be retained in functional layouts as long as machine sharing is within the cell or between adjacent cells. Finally, an interactive chart for the design of the *DCM* flowline layout (first two stages of the proposed research methodology) is presented in *Figure 4.8.*

This proposed first, of the two stages of this research, generates machine groups, identifies a flowline layout for each group, indicates which flowlines must be placed adjacent to each other to minimise intercell flow distances, and an approximate configuration of the aisles. Functional layouts, as well as machine groupings, are simultaneously encouraged. The classification of arcs used is effective for assessing whether material handling or machine sharing, is necessary to minimise intracell or intercell travel distances respectively. Examples are presented in the next chapter, and compared with three different *CM* design methods. This work shows the results obtained by this method, together with an improved procedure which employs combinatorial optimisation techniques and stochastic optimisation (*SA* and *Evolutionary Algorithm (EA)*).

CHAPTER⑤

Experimental Study of Dynamic Cells and Machines Layout

5.1. Introduction

In this chapter, to illustrate the proposed model's use, a rudimentary example and a more detailed study is presented. Furthermore, a procedure for developing an approximate layout from the model's output is also explained, along with comparisons of the proposed *MWDRST* method with three other established *CM* design methods. An improved procedure is also evaluated with respect to these three *CM* methods. Combinatorial optimisation algorithms are also employed to solve *NP*-problems as part of the improved procedures, which are programmed in the mathematical programming tool *MATLAB* whose codes are presented in Appendix D. These optimisation algorithms (such as *QAP*, *SA*, *TSP*, *2-opt* and *EA*) have been tested and applied to these *NP* examples. A short review is included in this chapter. However a more detailed explanation about combinatorial optimisation algorithms used in this research is presented in Appendix C.

5.2. An Rudimentary Example of Proposed *MWDRST* Research Model

The proposed *MWDRST* research model (*Table 4.2 and Table 4.3*) and the final *DCMS* layout (first two-stages I and II of the solution strategy) will be explained using a rudimentary example. It will be demonstrated that the sequential use of these models efficiently integrates the machine grouping and layout design issues discussed in the previous Chapters 3 and 4.

5.2.1. Stage I: Steps 1 – 6

The problem of the first stage of the research model solved here was to capture the directionality embedded in the operational sequences of a variety of parts produced in a facility for the *DCM* layout design. The operation sequence and batch quantity of the parts is the only input data. The travel chart *(Table 5.2)* obtained from this data is the input to this Stage I model illustrative example to obtain the *MWDRST*. Thus, utilising the *MWDRST* the forward and backward arcs in the original digraph can be eliminated. Here the example's initial layout for the shop is shown in *Figure 5.1* and the number of machines available of each machine type is also given.

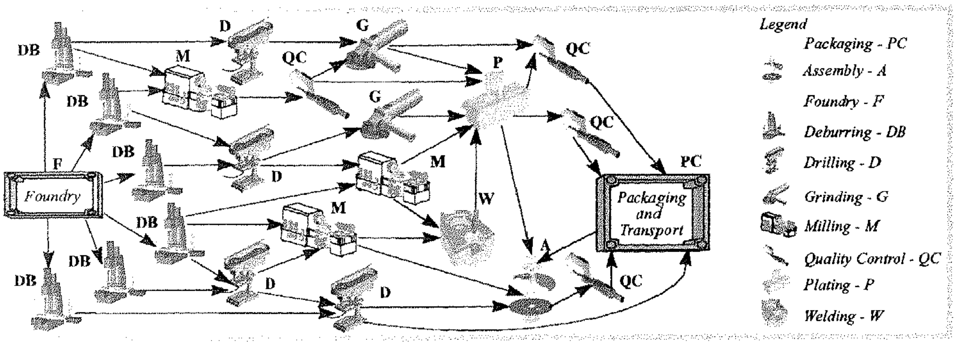


Figure 5. 1 - Initial layout

5.2.1.1. Step 1

Analysing the initial layout from *Figure 5.1*, the Foundry *(F)*, Plating *(P)*, Assembly *(A)*, and Packaging *(PC)* sections are assumed indivisible due to the type of functions that they perform and only one Welder *(W)* is available. If the fumes from welding make it incompatible with all the other sections, it must be located on the periphery of the shop. A similar location constraint applies to plating and foundry. The grinders should preferably be located together since these are usually expensive precision machine tools (also because of the occupation safety and health regulations). The quality control stations must be assumed divisible, since quality control is preferably performed at the location where some machining or assembly is done.

Part	*Operation Sequence*	*Batch Quantity*
1	F-DB-D-G-QC-PC	6
2	F-DB-M-QC-G-P-QC-PC	8
3	F-DB-M-QC-P-QC-PC	7
4	F-DB-D-G-P-QC-PC	7
5	F-DB-D-M-P-QC-PC	7
6	F-DB-M-W-P-A-QC-PC	3
7	F-W-P-A-QC-PC	3
8	F-DB-M-W-P-A-QC-PC	3
9	F-DB-M-A-QC-PC	3
10	F-DB-D-M-A-QC-PC	7
11	F-DB-D-D-A-QC-PC	4
12	F-DB-D-A-QC-PC	3
13	F-DB-D-PC-A-QC-PC	4

Table 5. 1 - Operation sequence and batch quantity data to provide a numerical solution

The operation sequences of the individual parts and the batch quantity for each part are shown in *Table 5.1*. The use of a travel chart is preferred instead of using the Muther's relationship diagram. Part 13 has a peculiar sequence in that it returns for assembly and quality control after visiting packaging once. Such "exceptions" are typical and need to be eliminated by process re-planning.

5.2.1.2. Step 2

Based on this input data (*Table 5.1*), ignoring parts setup and operation times, the travel chart shown in *Table 5.2* can be developed (utilising *MATLAB* or any spreadsheet software). This travel chart, which was arranged in logical order of the operation sequences, (actual flow matrix as shown in the *Table 5.1*) resulted from the computation of distances between every pair of machines from *Table 5.1.*

TO Machine

FROM Machine	*F*	*DB*	*D*	*G*	*QC*	*PC*	*M*	*P*	*W*	*A*
F	*	62							3	
DB		*	38				24			
D			*	13		4	14			7
G				*	6			15		
QC				8	*	65		7		
PC						*				4
M					15		*	7	6	10
P					29			*		9
W								9	*	
A					30					*

Table 5. 2 - Travel chart used for DCM layout design

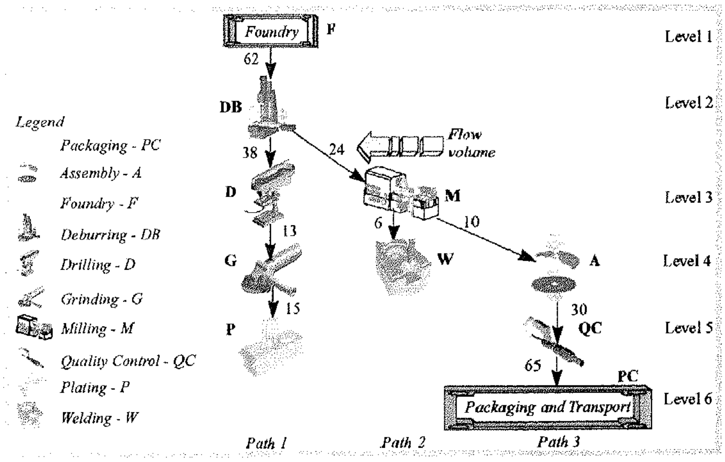


Figure 5. 2 - MWDRST maximal spanning distance – after stage I

5.2.1.3. Step 3

Based on the travel chart from *Table 5.2* the initial digraph is developed in a graphical (network flow) manner (*Figure 5.1*). Analysing this digraph, as mentioned before in step 1, and noting all arcs incident from the *PC* node, the exception operation (explained in step 1) corresponding to part 13 can be recognised by identifying all arcs incident from the *PC* node. Corresponding to the four quality control machines available, the arcs incident to and from the *QC* nodes involve *G, A, M, P* and *PC*. Hence, using the travel chart *(Table 5.2)*, indivisible bottleneck machines or shared machines which can be duplicated, or exception operations with low flow volumes, can be identified.

5.2.1.4. Step 4

Using *LINGO*, the stage I (from *Table 4.2 and 4.3*) research model, presented in Chapter 4, was solved to obtain the *MWDRST*, as shown in *Figure 5.2*. It will be shown that stage II is a three-node permutation problem, compared to the ten-node (machines) problem in the original travel chart (*Figure 5.2* and explained in Appendix A).

5.2.1.5. Step 5

Step 5 generates the *MWDRST* material flowline paths. Thus, from *Figure 5.2*, a total of three paths can be identified:

- *Path 1: F→DB →D →G →P;*

- *Path 2: F → DB → M → W*; and
- *Path 3: F → DB → M → A → QC → PC.*

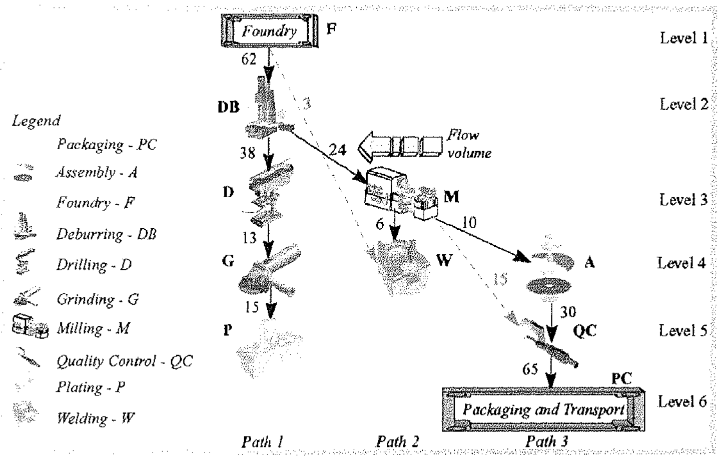


Figure 5. 3 - Only forward arcs in the MWDRST

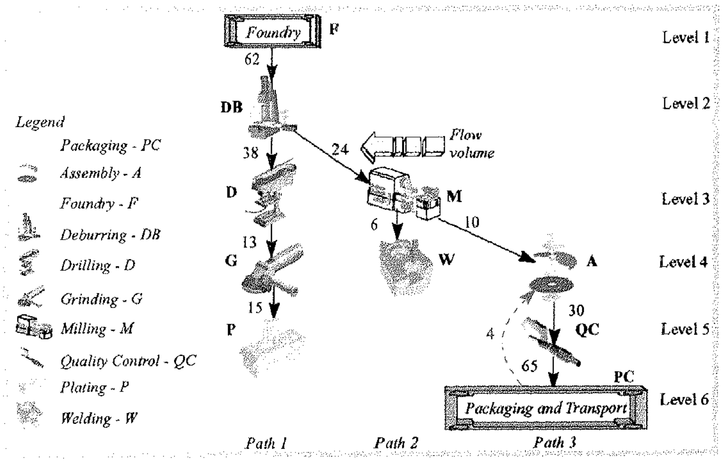


Figure 5. 4 - Arcs in MWDRST and backward arcs

	Path 1	**Path 2**	**Path 3**
Path 1			
Path 2	D-M (14), M-P (7), W-P (9)		
Path 3	D-PC (4), D-A (7), G-QC (6),QC-G (8), QC-P (7), P-QC (29), P-A (9)	0	

Table 5. 3 - Crisscrossing arcs intercell flows

5.2.1.5. Step 6

Classification of arcs in the *MWDRST* from *Figure 5.3*, shows that two forward arcs *(F → W)* and *(M → QC)*, and one backward arc *(PC → A)* exist *(Figure 5.4)*, since they are linking more than one level. The crisscrossing arcs *(Figure 5.5)*, from *Table 5.3*, suggest that machine duplication will be required.

5.2.2. Stage II – Steps 7 – 11

From the stage I research model *Figure 5.2*, machine types *DB* and *M* are the two branching nodes. Thus the head and tail node of any crisscrossing arc will each belong to a different path. Since the *MWDRST* stage II model permutates the branches of the *MWDRST* without dividing any of the branching nodes, and uses a symmetric intercell flow matrix, the direction of these arcs does not matter. This approach of solving stage II (*Table 4.2 and 4.3*) as an optimal linear arrangement problem is valid, for it is only required to place cells with high flow interactions adjacent to each other.

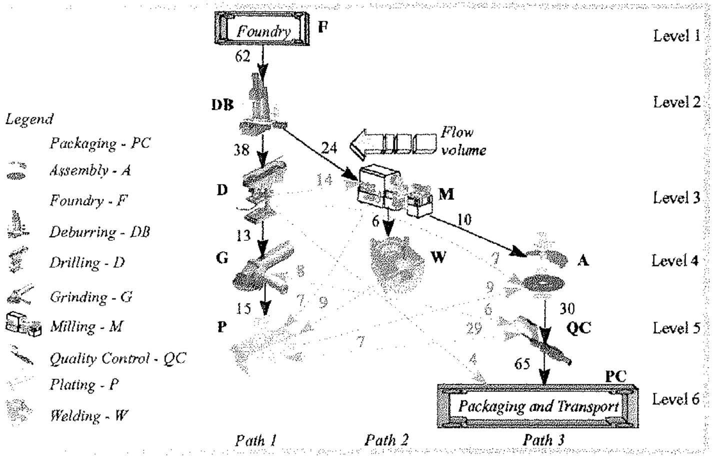


Figure 5. 5 - Arcs in MWDRST and crisscrossing arcs

	Path 1	Path 2	Path 3
Path 1			
Path 2	30		
Path 3	70	0	

Table 5. 4 - Crisscrossing arcs intercell flows

5.2.2.1. Step 7

Converting the crisscrossing arcs to edges, the matrix of intercell flows can be developed. *Table 5.4* shows these intercell flows between the three paths in the *MWDRST*. This table shows the deviation from planar graph theory, since arcs which cross each other are also considered.

5.2.2.2. Step 8

In this step, pivoting (permutations) of the *MWDRST* paths are considered. If duplication of machines *DB* and *M* is assumed, then the matrix in *Table 5.4* is the only input for the stage II model. Using only this matrix of intercell flows, the stage II result will yield the *MWDRST* shown in *Figure 5.6*. The order of the paths is *P2-P1-P3*. This will allow the cells to be independent, but requires machine *M* to be duplicated.

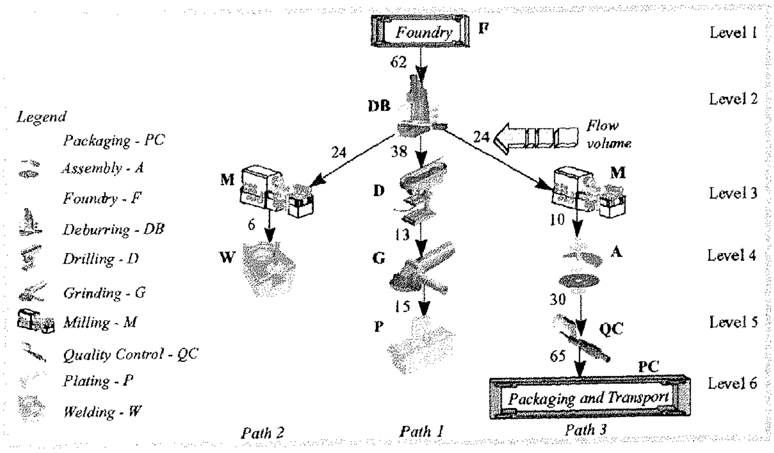


Figure 5. 6 - MWDRST - after stage II modelling

5.2.2.3. Step 9

Analysing results from step 8, the permutated branches of the *MWDRST*, it may be observed that path 2 has only one additional machine, the welding station *W*. The intercell flow from *Figure 5.5*, corresponding to machines unique to the path 2, is due only to arc *(W,P)*. Arc *(M,P)* , from *Figure 5.5*, involves a milling machine which could be placed in a functional layout. Thus to avoid machine duplication completely, stage II requires a more realistic solution. To avoid the duplication of the milling machines, paths *P2* and *P3* can be moved as a group, allowing only the permutations *P2-P3* or *P3-P2* (either *P2* will be adjacent to *P1* or *P3*, not both) as shown in *Figure 5. 6.* To force paths sharing branching nodes to be together whilst the *MWDRST* is permutated, requires the adoption of the weighting schema (from the previous section and Appendix A.9) shown in *Table 5.5.*

Machines in Cell	***Level***	***Weight Assigned***
DB	2	210
M	3	210

Table 5. 5 - Shared machine flow weights

5.2.2.4. Step 10

After step 9, where the material flowline layout is planned, the next step is to identify the crisscrossing arcs for machine duplication. The machine duplication procedure is now presented, where the values for the w_{ks} used in stage II research model are shown in *Table 5.5.* Machine *DB* was assigned a weight of *210* (from *Table 5.4) max = 70* since *p = 3*, and in order to make weight > *140*, the weight was set *=3 x 70= 210*). Thereafter, machine *M* can also be assigned a weight of *210*. In this way *420* is made greater than the weight of *DB*. This ensures that the stage II research model functions correctly, by forcing paths sharing a number of branching nodes to move as a group.

By summing the weights of machines common to both paths, any pairs are developed and shown in *Table 5.6* (similar to *Table 5.4*). Combination of *Tables 5.4* and *Tables 5.6* produce *Table 5.7*, which shows the new intercell flows for the stage II research model. The sequence of paths is now forced to be *P1-P3-P2 (or P2-P3-P1).* The corresponding *MWDRST* is shown in *Figure 5.7.*

	Path 1	***Path 2***	***Path 3***
Path 1			
Path 2	210		
Path 3	210	420	

Table 5. 6 - Intercell flows (shared machines)

	Path 1	***Path 2***	***Path 3***
Path 1			
Path 2	240		
Path 3	280	420	

Table 5. 7 - Combined intercell flows - for stage II modelling

Next the layout from the *MWDRST* and flowline skeleton shown in *Figure 5.7 and Figure 5.8* is generated. For each path, the machine type with the least number of machines available was chosen. Using this number as a bound, as many "copies" of that path were created parallel to each other. The process was repeated for the other paths (two copies of path 1 could be created and likewise each of the other two paths) and the resultant intermediate *DCM* flowline layout is shown in *Figure 5.9.*

5.2.2.5. Step 11

Finally at the stage II research model, using the crisscrossing arcs, the additional machines of all types have to be placed between paths 1 and 3 which have an intercell flow due to arc *(D,M)* (ie. two *DBs*, two *Ds* and one *M*) (*Figure 5.10*). However, their location must be based on flow directions, without losing the functional layout for each machine type. The two machines included in each crisscrossing arc can then determine the adjacencies. The quality control machines have been duplicated freely at the four locations where they are required. A part can be checked immediately after machining or assembly before it is sent to the next machine in its operational sequence. This decision is supported by one of the major benefits of *CM* quality control-cells due to the proximity of machines and immediate feedback amongst operations within the cells. The final layout is now shown in *Figure 5.10.*

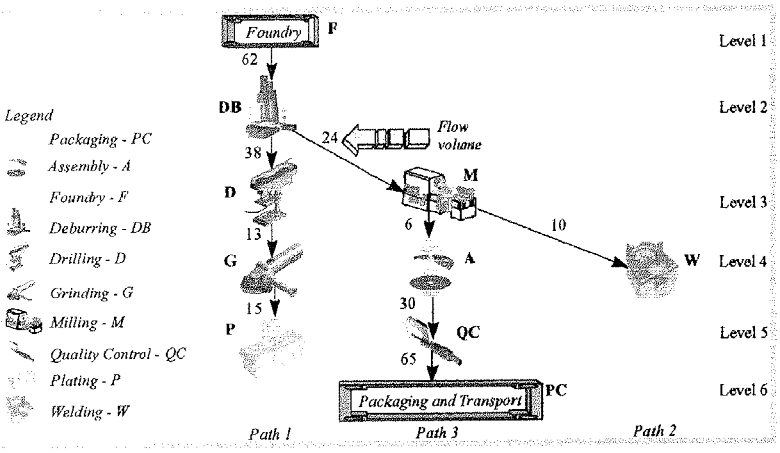


Figure 5. 7 - Stage II MWDRST with no machine duplication

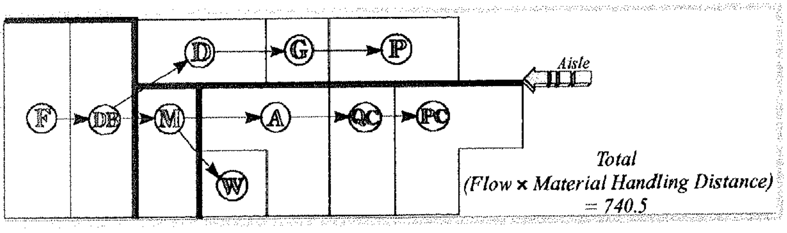


Figure 5. 8 - Flowline skeleton for MWDRT with no machine duplication

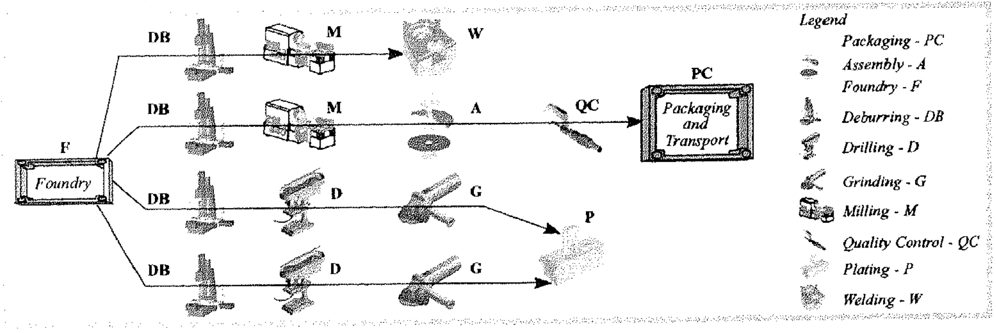


Figure 5. 9 - Intermediate layout from the final MWDRST

5.2.3. Comments

In this section of Chapter 5, an rudimentary example is presented which utilises the proposed research methodology for the design of the *DCM* flowline layout. The initial layout for the shop is shown in *Figure 5.1.* Machine locations in both layouts *(Figures 5.1 and 5.10)* create identical machine adjacencies, although the original layout has unnecessary machine duplications. With the proposed research method, a defined flow structure is obtained. This helps to retain each machine type in a functional layout and maintain "dynamic" machine groups. The machines allocated to a part can be chosen so as to avoid any intercell flows. The flowlines and crisscrossing arcs connecting them also suggest aisles for a material flow handling network.

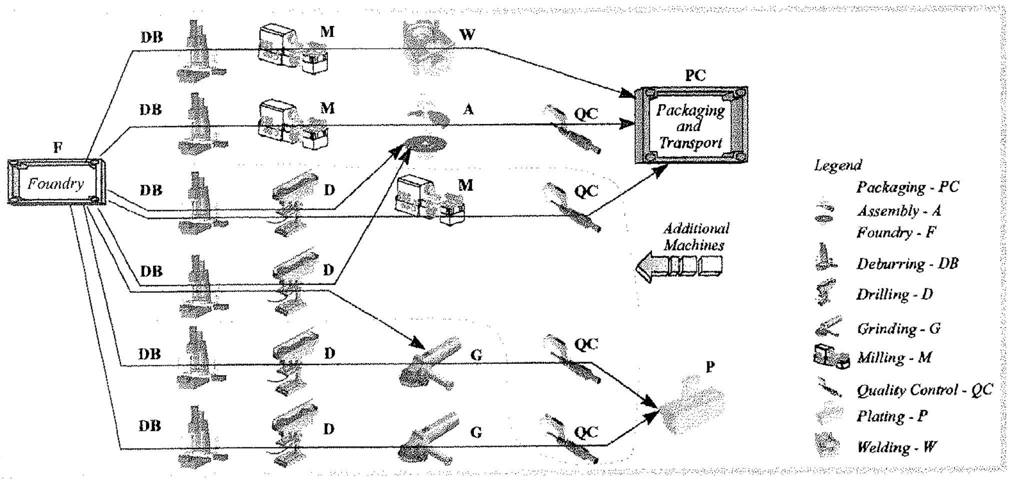


Figure 5. 10 - Final DCMS flowline layout

Generally speaking, from a manufacturing point of view, machines grouped consecutively on any path in the *MWDRST* could be replaced by *CNC* machining centres. The conventional machines that are replaced can then be assigned to the nonadjacent cells to minimise flows across the central aisle. This would eliminate some crisscrossing arcs.

5.3. Comparison of *MWDRST* with Other Flowline Graph's Adjacency Based Design Procedures

Here the comparison of the *MWDRST* proposed research methodology (stage I and stage II) for the design of *DCM* flowline layouts from Chapter 4, (*Table 4.2* and *Table 4.3)* with two other Graph Adjacency design procedures is now presented. The same rudimentary example from section 5.1, is utilised as the input. Before comparing the first two stages of the proposed *DCM* flowline layout methodology from Chapter 4, with other graph methods, it is appropriate now to recapitulate some essential facts. The three main principles for the design of effective directed flowline in a facility layout, are as follows:

1. Maximise directed material flow.
2. Minimise the total material flow.
3. Minimise the cost of the material flow.

Tompkins and White [1984] state, "A directed material flow path is an uninterrupted material flow path that progresses from the origin to the destination without backtracking and an uninterrupted material flow path is a flow path that does not intersect with other paths". It may be recollected, from Chapter 2 (the classical model for layout design), the Quadratic Assignment Problem that is embedded in *CRAFT* fulfils principles 2 and 3 but overlooks principle 1. The *MWDRST* is a rooted and directed tree composed of directed flow paths linking departments - thus it incorporates principle 1. Currently used Adjacency Graph based flowline design layout methods are:

- Maximum Spanning Tree (*MST*) (*Carrie [1976], Sirnivasan [1994] and Lin [1996]).*

- Cut Trees (*Montreuil [1993] and Kandiller [1998]).*
- Maximum Weight Planar Graphs (*MWPG*) (*Foulds and Griffin [1985]).*

5.3.1. Comparison of Proposed *MWDRST* with *MST* Procedure

To compare the illustrative example solution (from the previous section 5.1) with the Sirnivasan [1994] approach, (similar to Carrie, from the Literature Review) this solution applied *MST* (Prim's algorithm, Appendix B) to *CM* design in this example, and the program for finding *MST* was coded and executed in *MATLAB* and *LINGO.* The resulting *MST* undirectional flowline layout is shown in *Figure 5.11* and the *MST* flowline skeleton layout is shown in *Figure 5.12,* after the *MST* algorithm is applied.

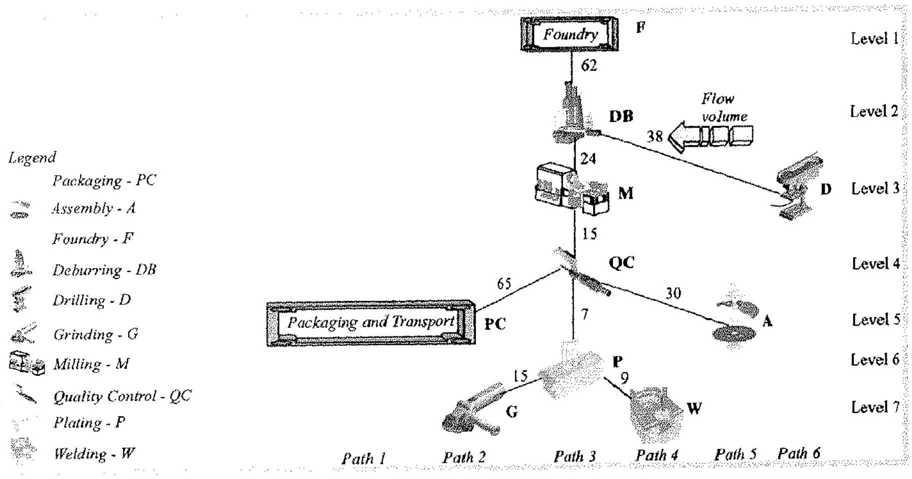


Figure 5. 11 - MST flowline layouts

It is clearly visible from section 5.1, noting the *MWDRST* example figures *(Figures 5.2, 5.6, 5.7 and 5.8)* and the *MST* solution figures *(Figures 5.11 and 5.12)* that a *MWDRST* – stage I and stage II, is a much better design (fewer paths) than a *MST* flowline skeleton procedure. This also requires using the *MST* algorithm as a base for solving the *TSP* and Branch and Bound problem, as will be shown later in this Chapter. Presented now is a short overview of the *MST* problem (an extended overview is presented in Appendix B).

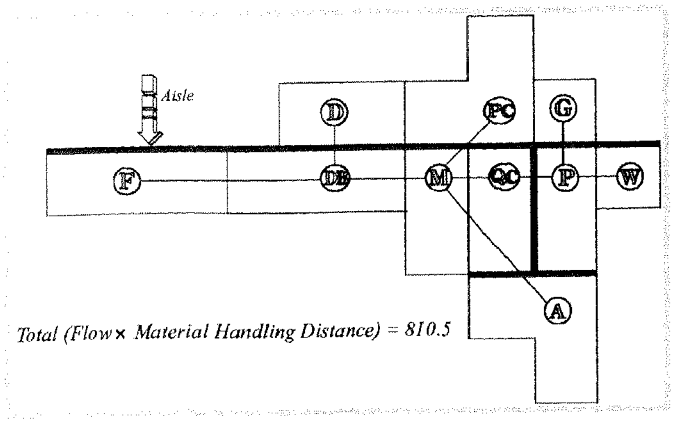


Figure 5. 12 - MST flowline skeleton layout

The *MST* problem was first formulated in 1926 by Boruvka. It was evolved during the electrification of Southern Moravia to find out the most economical layout of a power-line network. Some polynomial-time algorithms have been developed to solve this problem. Amongst them the Prim and the Dijkstra algorithms are the most famous ones, which are presented in the Appendix B in more detail and also coded in *MATLAB* (source codes also shown in Appendix D). Below is given a summary of the solution procedure:

- *Set all edges emanating from a node as dependents of that node to store the node and edge relationship*
- *Set current label 1*
- *Make multiple passes:*
  - *Delete all leaf nodes in each pass until 2 ≥ nodes remain*
  - *Set label of each deleted leaf node to current label*
  - *Increment current label by 1 with each pass*
- *Mark node with maximum label as centre node (if there are 2 nodes with maximum label, see both as centre node)*

Adding temporary nodes and edges

- *If necessary, extend length of all subtrees of centre node and all other nodes with label > 1 by adding temporary nodes and edges to equal the length of the largest subtree*

Node placement

- *Place centre node at tree of graph*
- *Place remaining nodes on concentric circles which passes from lower radius circle to higher radius ones*

Removing temporary nodes and edges

Figure 5. 13 - Cut tree algorithm [Montreuil 1993]

5.3.2. Comparison of Proposed *MWDRST* with *Cut Tree* Procedure

The cut tree approach by Montreuil [1993] represents each department by a node. Arcs represent material flow and layout rearrangement by applying the Gomory-Hu algorithm [1969]. Cut trees arcs are then manually converted into layouts, and then fed into exchange procedures to obtain better results. The algorithm for drawing the cut tree is shown in *Figure 5.13.*

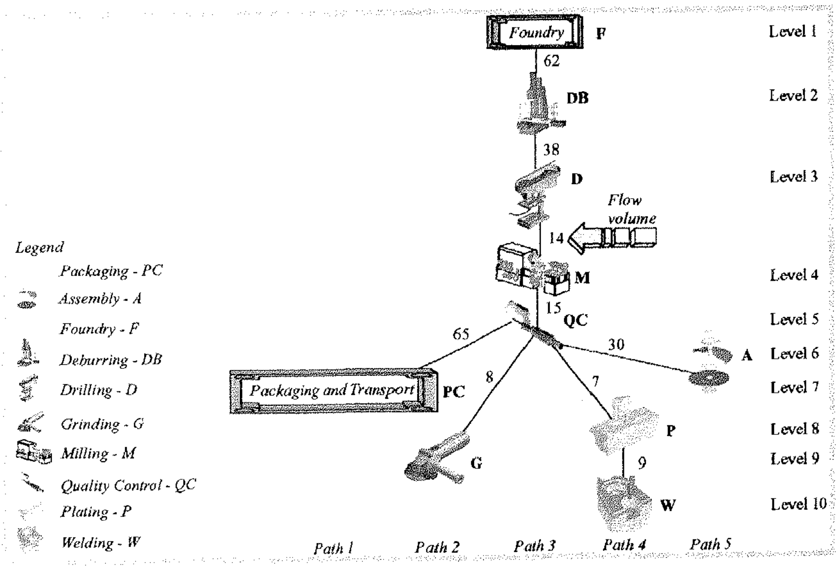


Figure 5. 14 - Cut Tree flowline layout solution

To compare the Montreuil approach with the *MWDRST* proposed research methodology the *Cut tree* (Gomory-Hu) algorithm is applied to *CM* design. In this illustrative example the program for finding the *Cut Tree* was coded and executed in *MATLAB*. The resulting *Cut tree* undirectional flowline layout is shown in *Figure 5.14* and the *Cut tree* flowline skeleton layout shown in *Figure 5.15,* after the *Cut tree* algorithm was applied.

It is clearly visible from section 5.1 noting the *MWDRST* example figures *(Figures 5.2, 5.6, 5.7 and 5.8)*, and the *Cut tree* solution figures *(Figures 5.14 and 5.15)*, that the *MWDRST* – stage I and stage II - is a much better design (fewer paths) than the *Cut tree* flowline skeleton procedure.

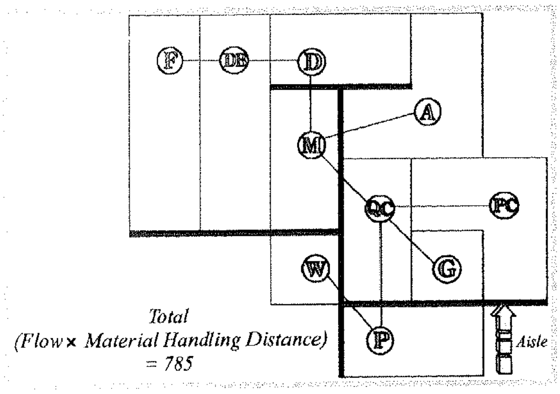


Figure 5. 15 - Cut Tree flowline skeleton layout solution

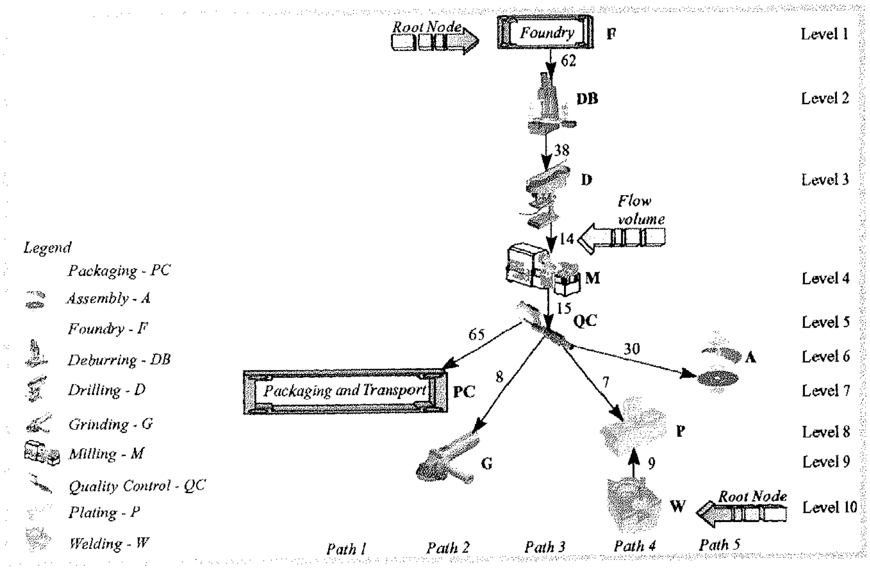


Figure 5. 16 - Cut Tree flowline directionality

5.3.3 Flowline Directionality Comparison

Flowline directionality is defined as the maximum uninterrupted length of the operational sequence of a part that is embedded in each of the design flowline skeletons. For example, the design flowline skeleton for the *MWDRST* (from *Figure 5.5*) and operation sequence (from *Table 5.1)* has a maximum path length *F-DB-D-G-QC-PC* with an operation substring *F-DB-D-G* of maximum uninterrupted length 3 units. Comparison of the flowline directionality for all three methods (*MST*, Cut Tree and *MWDRST*) and for all parts, with mean and standard deviations, is given in *Table 5.8* (the higher values of the flowline directionality the better). Standard deviations, from *Table 5.8*, are derived using the *STDEVA* (spreadsheet function) which estimates standard deviation based on an appropriate sample.

Part No.	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*Mean*	*Std. Dev.*
Cut Tree	2	1	1	2	3	3	0	1	1	3	2	2	2	1.77	0.93
MWDRST	3	2	2	4	2	3	0	3	6	2	2	2	2	2.54	1.39
MST	2	3	2	2	2	2	0	2	3	2	2	2	2	2	0.71

Table 5. 8 - Directionality comparison of the flowlines procedures

As is shown in the graph theory review (Appendix A) and explained at the beginning of this chapter, undirected graphs *MST* and *Cut tree* lack flowline directionality. To capture flowline directionality clearly the undirected graphs (*MST* and *Cut tree*) need to be converted into directed graphs. This problem can be solved with a procedure where splitting of graph vertices into input and output parts is applied to the undirected graphs. The procedure of splitting the undirected graph vertices into input and output parts, which helps to capture the directionality of flowlines, has been applied to the *MST* and *Cut tree* solutions. *Figure 5.16* shows how this splitting procedure is applied to the Cut Tree example shown in *Figure 5.14*.

Part No	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***Total***
Cut Tree	4	5	4	4	5	3	3	3	3	5	4	4	4	51
MWDRST	4	4	2	5	3	5	2	5	5	5	4	4	4	52
MST	3	6	5	5	4	4	3	4	4	4	3	3	3	51

Table 5. 9 - Sum of the total lengths

			MST			***CUT TREE***			***MWDRST***		
From	***To***	***V***	***D***	$\vec{D}$	***VD***	***D***	$\vec{D}$	***VD***	***D***	$\vec{D}$	***VD***
F	*DB*	62	1	*F*	62	1	*B*	62	1	*F*	62
F	*W*	3	6	*F*	18	2	*C*	12	2.5	*F*	12
DB	*D*	38	1.5	*C*	57	2.5	*C*	114	1.5	*C*	95
DB	*M*	24	1.5	*F*	36	2.5	*C*	36	3	*F*	36
D	*G*	13	3.5	*F*	45.5	3.5	*F*	58.5	1.5	*F*	19.5
D	*PC*	4	2.5	*F*	10	5.5	*C*	18	7.5	*C*	22
D	*M*	14	3	*C*	42	1	*F*	28	3	*C*	21
D	*A*	7	6.5	*C*	45.5	4.5	*F*	10.5	4.5	*C*	17.5
G	*QC*	6	2.5	*B*	15	2.5	*C*	9	5	*C*	15
G	*P*	15	1.5	*C*	22.5	1.5	*F*	22.5	1.5	*F*	22.5
QC	*G*	8	2.5	*F*	20	2.5	*B*	12	5	*C*	20
QC	*PC*	65	2.5	*C*	162.5	1.5	*B*	97.5	1.5	*F*	97.5
QC	*P*	7	1	*F*	7	4	*B*	14	3	*C*	14
PC	*A*	4	5	*C*	20	3	*F*	12	3	*B*	12
M	*P*	7	2	*F*	14	2	*F*	28	2.5	*C*	35
M	*W*	6	3.5	*F*	21	1.5	*F*	6	1.5	*F*	9
M	*A*	10	3.5	*C*	35	3.5	*F*	20	1.5	*F*	15
P	*QC*	29	1	*F*	29	4	*F*	43.5	3	*C*	58
P	*A*	9	2.5	*C*	22.5	2.5	*F*	31.5	1.5	*C*	13.5
W	*P*	9	1.5	*B*	13.5	3	*F*	22.5	1.5	*F*	31.5
A	*QC*	30	2.5	*C*	75	1.5	*C*	75	1.5	*F*	45
				C & B	**519.5**		***C & B***	**450**		***C & B***	**259.5**
Total weighted travel distances					***810.5***			***785***			***740.5***

Table 5. 10 Comparison of MWDRST with Cut Tree and MST (legend: F: Forward Flow; B: Backtrack Flow; C: Cross Flow; $\vec{D}$: Flow Direction; D: Distance; V: Volume of Flow; VD: Flow × Distance)

Also presented in *Table 5.9* is a comparison of the sum of the total length of substrings in the operation sequence of a part that is embedded in each of the three design flowline skeletons. How to derive the total length of substrings is shown in the *MWDRST* example (section 5.2.1.5). Thus from *Figure 5.5* the maximum path *F-DB-D-G-QC-PC* from the operational sequence (*Table 5.1*) has a maximum uninterupted path *F-DB-D-G* (length 3 units) and an embedded path *QC-PC* (length 1 unit) with a total length of 4 units.

5.3.4. Comments

Using the illustrative example comparisons compiled with respect to *MST* and the *Cut Tree*, the compared results are shown in *Table 5.10.* The main conclusions drawn from comparison with the *MWDRST* are:

- The total amount of backtrack and cross flows estimated for the layout and *MWDRST* is 50 % better than the *MST* and 42 % better than the Cut Tree (from *Table 5.8*).
- Comparing the total weighted travel distance (from *Table 5.10*) indicates that the *MWDRST* is 9 % better than the *MST* and 6 % better than the Cut Tree.
- The use of the proposed *MWDRST* is a feasible flowline design skeleton for generating a *DCM* layout for a dynamic-product facility.
- Unlike existing design flowline skeletons, the *MWDRST* embeds directed material flow paths in the layout. It classifies the flow arcs in a travel chart based on the direction of flow between different pairs of departments in the actual layout.

5.4. Comparison of *MWDRST* with Clustering Analysis Technique

5.4.1. Clustering Analysis Data

In this example data from an actual implementation of a cellular layout in the job shop of the General Electric Company, shown in *Figure 5.17* [Koenig 1981], was used. From the original paper, part families were formed using a cluster analysis to indicate the machines required to produce these families. A classification and coding analysis was performed to validate that the machines assigned to a part based on its code, matched the sequence of machines given by its route sheet. A machine loading analysis of the part families was also performed to determine the machine requirements of their cells as shown in *Table 5.11*.

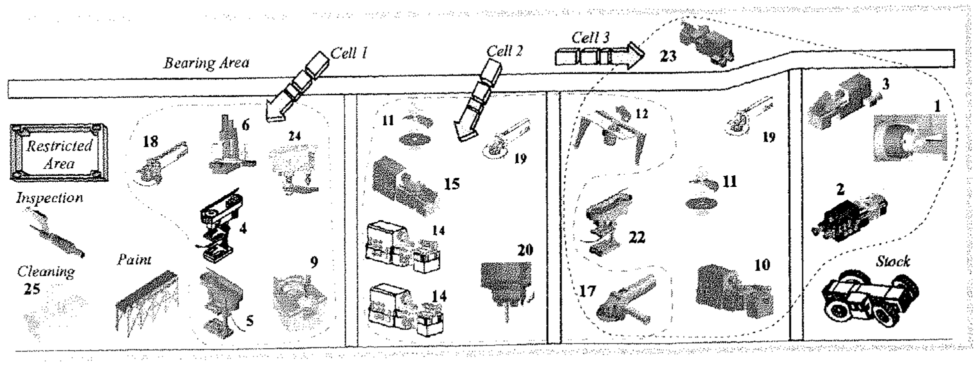


Figure 5.17 - Original GT three cell layout [Koenig 1981]

5.4.2 Stage I: Steps 1-6

5.4.2.1. Step 1

A clustering analysis data analysis performed in section 5.3.1 (machines, the sequence and machines route sheet) and *Table 5.11* was the input for the stage I research model.

		Number of machines required			
		Available capacity (1 shift/1536 hrs.)		Available capacity (2 shift/3072 hrs.)	
Machines	*Loading (hrs.)*	*Number of machines*	*Computed number*	*Actual need*	*Computed need*
1	1258	1	0.82	1	1.41
2	151	1	0.1	1	0.05
3	1	1	0.00	1	0.00
4	33	1	0.02	1	0.01
5	1153	1	0.75	1	0.37
6	2302	2	1.49	1	0.75
7	124	1	0.08	1	0.04
8	133	1	0.08	1	0.04
9	1154	1	0.75	1	0.37
10	3122	2	2.03	1	1.02
11	3689	3	2.41	2	1.21
12	2173	2	1.41	1	0.41
13	6	1	0.01	1	0.02
14	4831	4	3.14	2	1.57
15	1815	2	1.18	1	0.59
16	188	1	0.12	1	0.06
17	1771	2	1.15	1	0.57
18	360	1	0.23	1	0.11
19	5545	4	3.61	2	1.81
20	1274	1	0.83	1	0.41
21	1133	1	0.73	1	0.36
22	3580	3	2.33	2	1.23
23	248	1	0.16	1	0.08

Table 5. 11 - Machine loading results [Koenig 1981]

TO machine

FROM machine	***1***	***2***	***3***	***4***	***5***	***6***	***9***	***10***	***12***	***14***	***15***	***16***	***17***	***18***	***24***	***26***
1	229	8150							99	14					1020	
2		3	1			58	35		245	24			1635	110	2	
3																
4						30										
5		35				414	83									
6				10	18	611	795		58							
9				21	479	28			18						350	
10		1014				145		5545	112				206			
12									2563	8			100	20	217	
14						6			670	199		2	45		8	
15									4		35	6	4			10
16												63	12	14		
17		30	4			347			250	11			701		64	
18													119	28		
24		22				350		1014	352	356	27	2	104			
26																15

Table 5. 12 - Travel chart for GT layout design [Carrie 1977]

5.4.2.2. Step 2 And Step 3

Using the part lot size and their route cards, a step 2 travel chart was generated for the parts flow analysis, shown in *Table 5.12*, which is a matrix expression of the initial digraph (*D* for step 3 from Chapter 4 of the proposed research model). A comparison of the machines listed in *Table 5.11* and *Table 5.12* shows that the flow interactions of several machines were not indicated on Koening's travel chart *(Table 5.12)*. For instance, machines 26, 7, 8, 11, 13, 19, 20, 21, 22 and 23 have either not been included in the travel chart or have not been shown in the original layout. Machines 11 and 21 are multiple machines used for layout and deburring. Machine 3 is a miscellaneous works multiple machine, whilst machines 7, 8, 13 and 18 are shared work multiple machines in the bearing area. Hence, the analysis was restricted to the grouping and layout of the machines showed in the travel chart. It was assumed that machine 18, which is shared with the bearing area machines, could be relocated.

5.4.2.3. Step 4 and Step 5

The travel chart was the input to the first stage model (*Table 5.2*), programmed and employed within *MATLAB*. *Table 5.13* gives a summary of the solution report, the total weight of the *MWDRST* and the lists of the arcs contained within it. *Figure 5.18* shows this *MWDRST* rooted at machine 1. The weights (flows) of the individual arcs can be obtained from *Table 5.12*. Eight paths (or cells) can be observed, as well as the sequence of the machines within each cell (*Figure 5.18*). Path 6 does not corresponds to cell 1, paths 3 and 5 do, in the original layout, except for machine 18. However, cell 2 and cell 3 in the original layout cannot be directly identified in the remaining paths. Furthermore, the second stage has been simplified to an *OLA* (Optimal Linear Arrangement) problem in eight nodes. If the traditional method of diagonalisation of the travel chart had been used, it would have given an inferior *OLA* in 16 nodes, and would not have yielded any layout structure.

From node	***To node***	***Weight***	***From node***	***To node***	***Weight***
1	2	8150	24	14	356
1	24	1020	24	15	27
2	17	1635	6	9	795
2	18	110	9	4	21
24	6	350	9	5	479
24	10	1014	14	12	670
15	26	10	17	3	4
15	16	6	***MWDRST with value 14647***		

Table 5. 13 - Stage I MWDRST output from MATLAB

The cell sizes suggested by the *MWDRST* range from 3 to 6, and paths 1 and 4 can be eliminated from the second stage. Both paths have only one extra machine after the branch nodes 6 and 7, respectively. Their absorption would create a U shaped cell with machines 1, 2, 18, 24 and 10, as shown in *Figure 5.18*. However, this was not done prior to the second stage, since this example is extremely useful for demonstrating the total capabilities of the proposed dynamic cell formation method.

Since machines 2, 24, 9 and 15 are branching nodes, they suggest how the permutation amongst the different branches of the *MWDRST* must be limited. Machine 6, which is between machines 24 and 9, need not be given any weighting (same flowline). The overall nesting of the path permutations can be described as { [(1, 2)], [(3, 5), 4, 6, (7, 8)] }.

Since paths 1 and 2 branch from machine 2 and generate two permutations, they must remain adjacent. Also paths 3 to 8 branch from machine 24 and constitute the second major group. Hence, these two major groups of paths generate two permutations at the highest level in the *MWDRST*. However, within the second group, paths 3 and 5, which branch from machine 9, can be permutated. Similarly, paths 7 and 8, which branch from machine 15, can have two permutations, thus these pairs of paths can be permutated with paths 4 and 6.

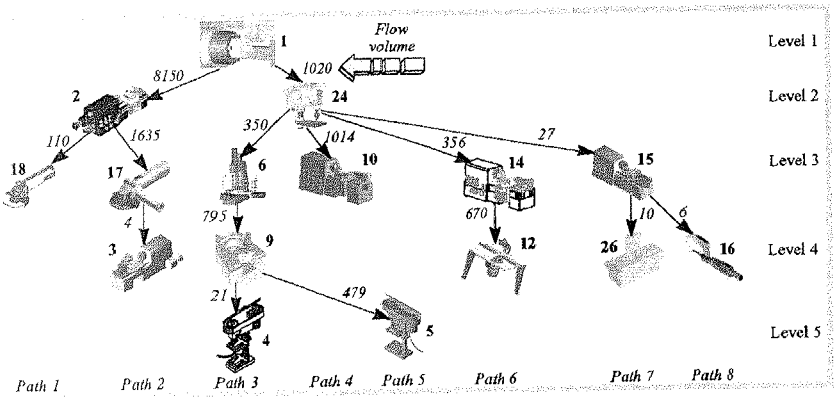


Figure 5. 18 - MWDRST - after stage I

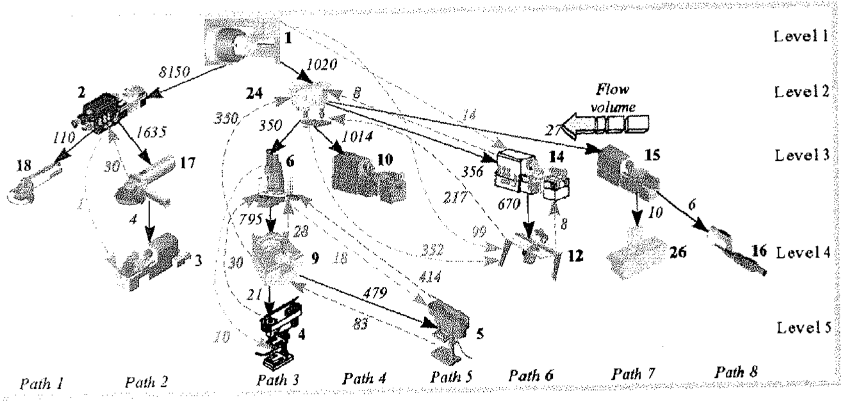


Figure 5. 19 - MWDRST forward and backward arcs

5.4.2.4. Step 6

Figure 5.19 shows forward and backward arcs on the *MWDRST* identified by inspection of the travel chart and the *MWDRST*. Since a flowline layout is assumed for each cell, these arcs can be eliminated from the second stage. Thus, the traditional two-dimensional *QAP* with rectilinear distances, simplifies to an *OLA* (one-dimensional) problem. For this condition, the two-dimensional *QAP* would have to simultaneously analyse the effect of having U or S intracell layouts on intercell layout and machine sharing, because of reducing the intracell travel distances. However, the disadvantages are that this reduces the intercell machine adjacencies and increases intercell travel distances, and would necessitate more intercell duplication of the shared machine types.

	P1	***P2***	***P3***	***P4***	***P5***	***P6***	***P7***	***P8***
P1								
P2	119							
P3	117	632						
P4	1038	1412	145					
P5	152	667	0	145				
P6	313	867	82	112	82			
P7	24	196	0	0	0	4		
P8	38	208	0	0	0	6	0	

Table 5. 14 - Intercell flows (crisscrossing)

5.4.3. Stage II: Steps 7 – 11

5.4.3.1. Step 7

The travel chart for intercell flows, *Table 5.14,* is necessary for the second stage which is developed using only the crisscrossing arcs. Thus, with respect to the definition of a crisscrossing arc, it will therefore connect any two paths in the *MWDRST*. This will contribute to the intercell flow between them.

Machine in cell	*Level*	*Weight assigned*
6	2	3000
7	2	3000
15	3	Not applied
56	3	5000
33	4	7000

Table 5. 15 - Flow weights for shared machines

	P1	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*
P1								
P2	3000							
P3	0	0						
P4	0	0	3000					
P5	0	0	10000	3000				
P6	0	0	3000	3000	3000			
P7	0	0	3000	3000	3000	3000		
P8	0	0	3000	3000	3000	3000	8000	

Table 5. 16 - Intercell flows (shared machines)

Table 5.15 shows the flow weightings, which are assigned to the machines that are branch nodes relating two or more paths. These weightings were given arbitrarily high values and were arrived at iteratively. The goal in this case was mainly to prevent paths, which share indivisible machines, from being assigned nonconsecutive positions in the *OLA* for the second stage. The cumulative intercell flow weightings, which must be added to the values in *Table 5.14,* are shown in *Table 5.16*. Note that the larger the number of shared indivisible nodes in a pair of paths, the higher their intercell flow weighting. This will ensure that these paths remain adjacent during the permutation process of the second stage.

	P1	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*
P1								
P2	3119							
P3	117	632						
P4	1038	14112	3145					
P5	152	667	10000	3145				
P6	313	867	3082	3112	3082			
P7	24	196	3000	3000	3000	3004		
P8	38	208	3000	3000	3000	3006	8000	

Table 5. 17 - Combined intercell flows for stage II

Table 5.14 and *Table 5.16* must be combined to generate the travel chart to be used for the second stage. These flow values, which will constrain the permutation of the paths in the *MWDRST* without splitting the indivisible machines, are shown in *Table 5.17*.

5.4.3.2. Step 8

Since the second stage is the *NP*-complete *OLA* problem in eight nodes, there is need to control the branch and bound process in the *LINGO* integer programming model. Therefore this resulted in writing a *MATLAB* version branch and bound integer programming method. If an upper bound for this minimisation problem is known, it can be used to eliminate non-optimal solutions in the early stages of the branch and bound search used in integer programming. This approach is used to prune the search tree of solutions with an objective value greater then the upper bound.

Prior to the execution of the second stage, an upper bound of 271462 (actually 135731 for the symmetric flow matrix) for the objective function value was obtained using a pairwise-interchange heuristics routine, written in *MATLAB*. The linear arrangement of the paths in the *MWDRST* produced by this heuristic was 1, 2, 4, 6, 5, 3, 8, 7. For the purpose of comparison, and to check the validity of the second stage model, the original ordering after the first stage was 1, 2, 3, 5, 4, 6, 7, 8.

MWDRST second stage ***With Minimum Value*** ***Of 271462***			***Column***	***Distance***	***Weight***
			R12	1	0
			R13	5	00
			R34	0	12580
Path Nodes	***Distance***	***Weight***	R35	0	4000
A18	1	76816	R36	0	12328
A27	1	12082	R38	1	0
A33	1	-65856	R56	0	12328
A46	1	155424	R68	3	0
A54	1	71632	R78	0	32000
A65	1	78820	L34	3	0
A71	1	-40448	L56	1	0
A82	1	-17008	L78	1	0

Table 5. 18 - Summary of stage II output

Table 5.18 gives a brief summary of the second stage output, showing the positions assigned to the pendant node of each path in the *OLA*. The right and left values can be obtained from these positions, with each value for each pair of paths zero. Pairs of paths, with right or left values greater than one, correspond to parts whose operational sequences contain arcs producing intercell flows between nonadjacent cells. These parts would have to travel to the end of the aisle containing their cell before visiting the other cell. This type of linear flow distance is not captured in the standard layout design programs, where the rectilinear or euclidean distances allow flows across flowlines. This gave the optimal reorientation of the *MWDRST* to develop the final layout, which is shown in *Figure 5.20.*

The layout (*Figure 5.20*) implicitly avoids machine dividing and minimises intercell flow distances for the crisscrossing arcs simultaneously. It should be noted that the linear arrangement, obtained by the *MATLAB* built *SDPI* heuristic, was expressed differently as paths 1, 2, 6, 3, 5, 4, 8, 7. This means that path 1 goes to position 1, path 2 to position 2, path3 to position 6, path 5 to position 5, path 4 to position 3, path 6 to position 4, path 7 to position 8, and path 8 to position7, when it is compared to the initial path orientation from the *MWDRST* stage I.

5.4.3.3. Step 9

Figure 5.21 shows the approximate layout developed using the new *MWDRST* after the second stage, subject to the current shape of the shop (machines 15, 16 and 26 could not be placed adjacent to machine

24, where the stock area is shown). As a means of comparison, in *Figure 5.22,* the arcs in this *MWDRST* are also shown in the original layout. Machine 18 has been shown relocated from the bearing areas since it has significant flow interactions with machines 2 and 17. Certain inconsistencies in the original layout can also be observed (machines 14 and 15 are shown located together, although there is no flow interaction between them). It appears that, due to the improper locations of machines such as 24, 18 and 10 in the original layout, a poor flow structure resulted as is observed in *Figure 5.22*. The material flow aisles were included to allow the intercell flows from *Table 5.14,* and ease of material handling for the backward and forward intracell flows.

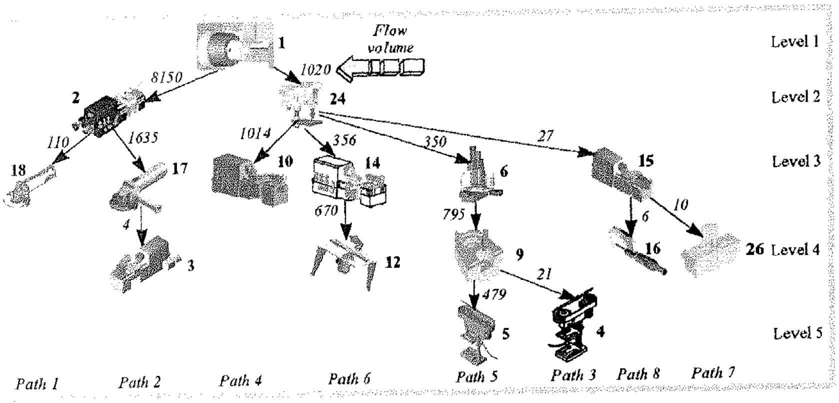


Figure 5. 20 - MWDRST at stage II

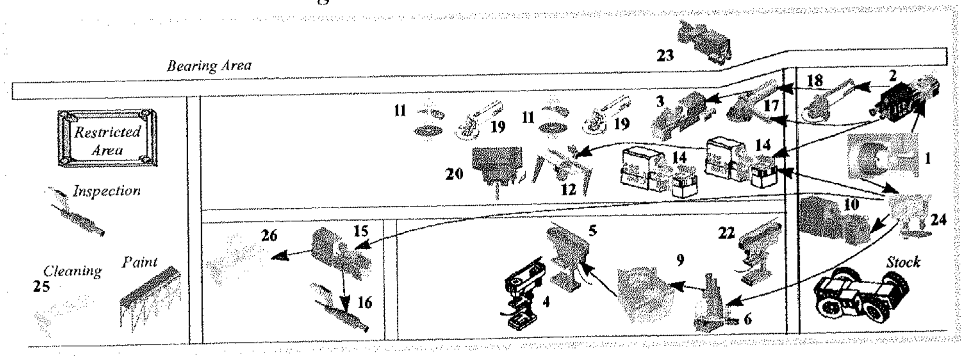


Figure 5. 21 - Layout after MWDRST stage II

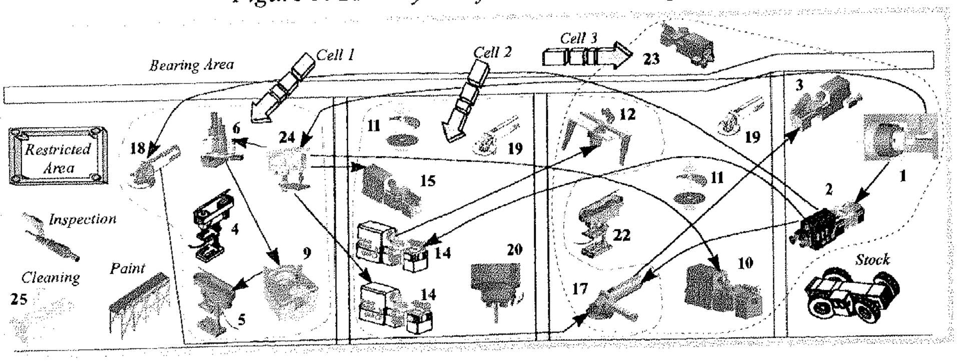


Figure 5. 22 - Larger travel distances in original (Figure 5. 17) layout

5.4.3.4. Step 10

The rounding-off of the theoretical machine requirements to integer values, from *Table 5.11,* demonstrates the problem of machine sharing when capacity calculations are performed on a part family basis. With reference to the number of machines shown in *Figure 5.17,* it is seen that the actual machine requirements for the two shift/3072 available hours case were used to plan the original layout, except for machine 14, of which two were available. Thus, this prevented machine 14 from being duplicated amongst the nonadjacent cells, because other related machines likewise could not be duplicated. However, a subsequent analysis to duplicate machines based on capacity calculations for the parts in the intercell flows was not conducted at this stage, because of the algorithm complexity and time limitations. Hence, this example becomes a simple application of both stages I and II of flowline *DCM* layout design as discussed in the previous chapter.

5.4.3.5. Step 11

The layouts in *Figure 5.21* and *Figure 5.22* then had to be compared for the total distances moved by all the parts in both layout configurations, where the objective function = $\Sigma f_{ij}\, d_{ij}$, as presented in Chapter 4. This was a simple but effective measure of the handling and layout capabilities of the proposed methodology. That is, the sum of the lengths of the flow arcs in *Figure 5.21,* weighted by their flow volumes, had to be compared with those shown for the original layout in *Figure 5.22.* This was achieved using a mathematical model built in *MATLAB,* with certain necessary assumptions such as rectilinear distances, unit lengths and widths for all the departments, and equal department sizes, for all the layout configurations evaluated.

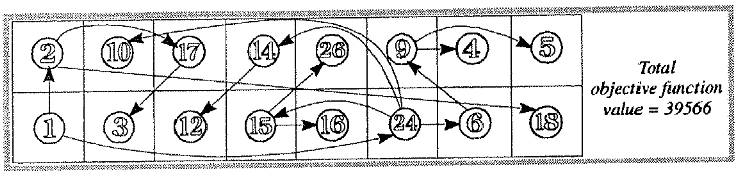


Figure 5. 23 - Original cells configuration as a branched flowline [Vakharia 1990]

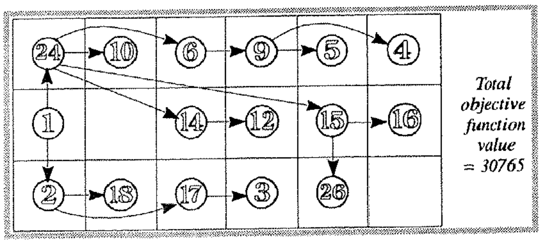


Figure 5. 24 - Final - proposed layout based on the MWDRST (restricted areas)

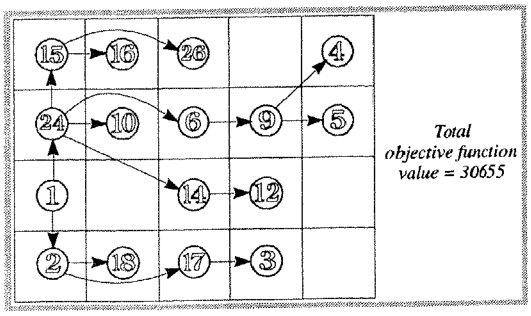


Figure 5. 25 Final – proposed layout based on the MWDRST (unrestricted areas)

Figure 5.23 – *Figure 5.33* shows the approximate rectilinear grid representation of the original layout and several alternative configurations suggested by the *MWDRST* in *Figure 5.20.* The arcs in these figures are those in the *MWDRST,* unless otherwise stated. The flow directions captured in the *MWDRST* are also shown in these grids. A large number of machines were shown in the original layout, which did not feature in the travel chart (*Figure 5.17*). Hence, the layouts shown can be considered the best

approximations within the allowed minimum travel distances. As suggested by the proposed method, the layout shown in *Figure 5.24* results in less material flow handling than the original layout shown in *Figure 5.23*. However, as shown by the layout in *Figure 5.25*, the machine groups and cell adjacencies suggested by the final *MWDRST* would help to further reduce the total flow distance.

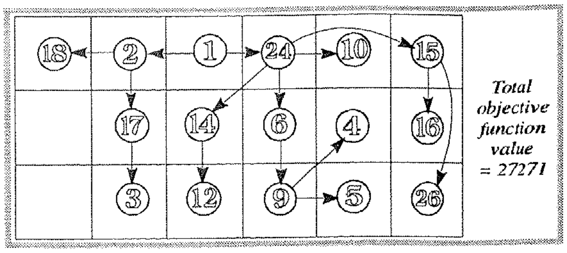


Figure 5. 26 - MWDRST intercell shapes layout

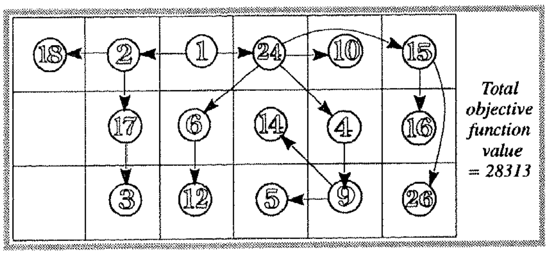


Figure 5. 27 - MWDRST final layout

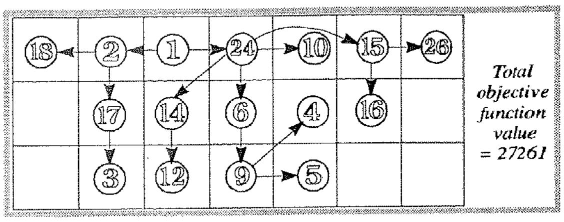


Figure 5. 28 - MWDRST U shaped layout

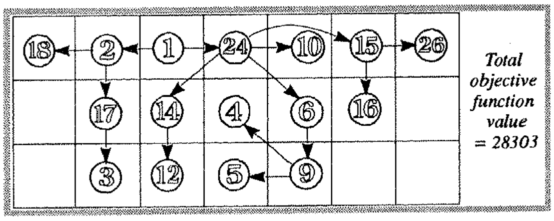


Figure 5. 29 - Another MWDRST U shaped layout

Figure 5.26 and *Figure 5.27* show the effect of evaluating a U layout for paths 3 and 5 after both modelling stages of the flowline *DCM* layout procedure have been executed. The complication that arises is that a U layout is effectively two paths instead of the original path. Instead of paths (6 – 9 - 5) and (6 – 9 - 4) attached to machine 24, paths (9 - 5) and (9 - 4) *(Figure 5.25)* have also be considered. These latter two paths can permutate with respect to each other whilst travelling from machine 24. Clearly, the layout in *Figure 5.26* results in lower handling costs than the layout in *Figure 5.27*. *Figure 5.28* and *Figure 5.29* further demonstrate the possibilities of obtaining efficient cellular layouts by merging the structure of the *MWDRST* and an intracell U layout, to reduce the lengths of certain flowlines. However, in comparison to the previous layouts, these layouts perform poorly in terms of giving an overall rectangular shape.

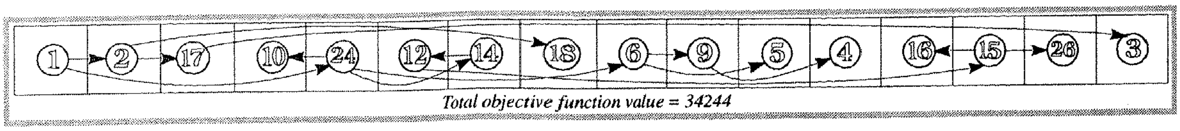


Figure 5. 30 - Undirectional (one dimensional) QAP flowline layout

Figure 5.30 is the linear one-dimensional *QAP* solution for travel chart clustering, with equal forward and backward material flow distances. Here machine 1 has been assumed as the root node and fixed as the first node in the heuristic solution to the optimal linear arrangement. Surprisingly, it yields a lower penalty than the original layout configuration implemented in the Koenig [1981] paper. Although the flows are generally unidirectional, the high total objective function (total distances) value of $34244 shows the basic disadvantage of the *OLA* model for cell formation. *Figure 5.31* is an approximate *DOLA* clustering solution to group machines based on the travel chart [Lenstra 1975] and presented in appendix C. This is a maximum weighted travelling salesman tour, starting and ending at machine 1.

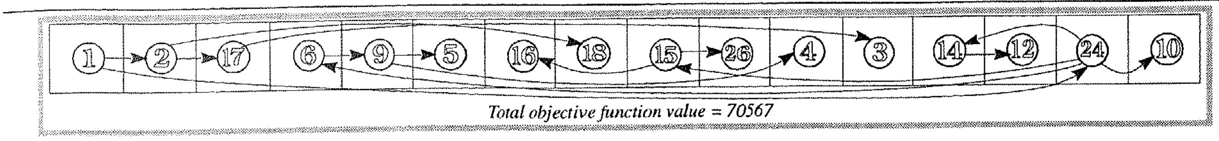


Figure 5. 31 - Directional (one dimensional) QAP flowline layout

This *TSP*-problem heuristic method was programmed in *MATLAB* (code attached in Appendix D) and is later improved - optimised with *SA* coded in *MATLAB* (*TSP* theory presented in Appendix C and code attached in Appendix D). This *TSP* arrangement forces all arcs to be directed away from the source node in order to model a pure flowline. Initially, a tour could not be found, since the graph was not completely connected. The travel chart was enhanced into a complete digraph by using *(10000 –* f_{ij}*)* instead of the original flow entries. Clearly, the prevention of backtracking to determine a pure flowline layout for the machine yields an extremely high flow penalty. Furthermore, the arcs included in the *TSP* and *MWDRST* have differing node degree constraints. *Figure 5.32* and *Figure 5.33* show the same *TSP* sequence in the form of a U layout and S layout, respectively. Whilst the U layout for the *TSP* gives a competetive solution, the disadvantage with this approach is that the initial clustering shown in the *MWDRST* is lost.

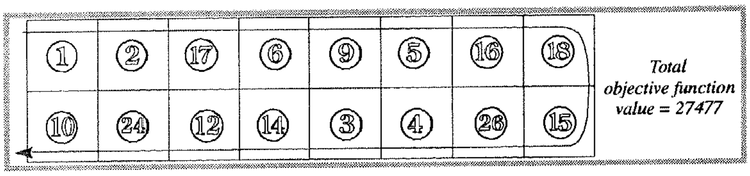


Figure 5. 32 - Directional (one dimensional) TSP flowline layout

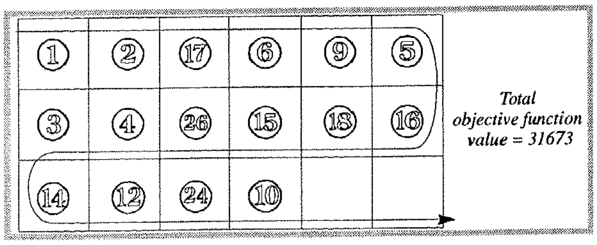


Figure 5. 33 - Directed one dimensional TSP with S layout

5.4.4. Comments

Here an important observation can be made that the optimal solution for stage II, obtained by the integer programming model, was equal to that obtained by the *MATLAB* heuristic. It has already been mentioned in Chapter 4 that computational time for the integer model is very long, not suitable for industrial cases, and from the just presented facts the *SDPI* solution gives the identical result but in quicker running time. However, Karisch [1998] recently compared results from the steepest descent pairwise interchange heuristic *(SDPI)*, with respect to *EA* and *SA* methods when applied to a *QAP* problem, with the outcome that *SDPI* still gave better performance then the other two methods. Thus, following Karisch's result the *QAP* steepest descent pairwise interchanges heuristic, coded (Appendix D) in *MATLAB* will only be applied in further examples.

Thus, the first stage has a significant effect of reducing the problem to a size where a heuristic applied to the second stage would also obtain the optimal linear arrangement. This would allow an alternative optima to be obtained and evaluated for generating other feasible layout configurations.

Furthermore, results from the proposed method in combination with other techniques and algorithms (coded in *MATLAB*), will be compared with results from packages such as *STORM*. Thus this work will quickly evaluate a finite number of *DCM* layout alternatives, which conform to the machine grouping, flowline layout, and intercell machine adjacencies given by the second stage *MWDRST*. This will help to integrate the effects of reducing the lengths of individual material flowlines by increasing the width of the shop (converting a rectangle to a square with the same area).

These results show that the proposed method correctly models the immediate adjacencies of machines, and the groups to which they could be assigned, if cells were desired. Flow distances and machine locations are usually determined subsequently. Hence, the evaluation of alternative *DCM* flowline layouts using the simple objective (total distance) function, coded in the *MATLAB* (or *EXCEL*) package is adequate for preliminary analyses.

5.5. Comparison of *MWDRST* with Operation Sequence Clustering Technique - A Detailed Study

In this section the example is derived from a paper, which proposed a cell formation method, that relates within-cell flow sequences of parts (or operation sequences) to the machine sharing problem [Vakharia 1990]. Part families with similar operational sequences are used to generate cells, which have a purely undirectional flowline structure. Interactions between intercell flows, cell size, intracell flow backtracking and machine utilisation are also considered.

Part No.	*Batch Size*	*No. of Batches per Year*	*Op1*	*Op2*	*Op3*	*Op4*	*Op5*	*Op6*	*Op7*	*M/C Operation sequences*
1	2	1	1	4	8	9				1,4,8,9
2	3	1	1	4	7	4	8	7		1,4,7,4,8,7
3	1	1	1	2	4	7	8	9		1,2,4,7,8,9
4	3	1	1	4	7	9				1,4,7,9
5	2	1	1	6	10	7	9			1,6,10,7,9
6	1	1	6	10	7	8	9			6,10,7,8,9
7	2	1	6	4	8	9				6,4,8,9
8	1	1	3	5	2	6	4	8	9	3,5,2,6,4,8,9
9	1	1	3	5	6	4	8	9		3,5,6,4,8,9
10	2	1	4	7	4	8				4,7,4,8
11	3	1	6							6
12	1	1	11	7	12					11,7,12
13	1	1	11	12						11,12
14	3	1	11	7	10					11,7,10
15	1	1	1	7	11	10	11	12		1,7,11,10,11,12
16	2	1	1	7	11	10	11	12		1,7,11,10,11,12
17	1	1	11	7	12					11,7,12
18	3	1	6	7	10					6,7,10
19	2	1	12							12

Table 5. 19 - Operation sequence and batch quantity [Vakharia 1990]

5.5.1. Stage I: Steps 1-6

5.5.1.1. Step 1

The input data used involves the operational sequences of the parts, batch quantities, setup and run times of the parts on the machines. Also considered is the available machine capacity and the number of machine of each type available for distribution amongst the cells. The operational sequences and batch quantities of the parts used in the analysis are shown in *Table 5.19.*

Machine type	*Number available*	*Total load (min.'s/day)*	*Actual machine requirements*
1	2	450	1
2	1	96	1
3	1	288	1
4	2	696	2
5	1	240	1
6	4	504	2
7	5	1280	3
8	1	288	1
9	1	468	1
10	7	1170	3
11	3	1080	3
12	1	340	1

Table 5. 20 - Machine loads and availability data [Vakharia 1990]

Table 5.20 shows the loads on the different machine types and the number of machines available of each type. If the available productive time per day per machine is 480 minutes, excessive machine redundancy can be observed in this table. This suggests that the number of machines required for changeover to independent cells, leads to an increase in the number of machines required. It will be shown that using the proposed method, the flow properties and machines of cells, can be obtained without rigid dedication of machine cells. Thus, machines can be retained in a functional layout but are assigned to one or more part families. This will ensure reduction in setup and operation times, yet allows flexibility to re-tool machines for other part families. Hence, a layout can be generated where all the advantages of functional layout, cell formation, flowline layout for cells and feasible intercell flows can be exploited. Thus this will minimise the need for rigid cell formation and machine sharing.

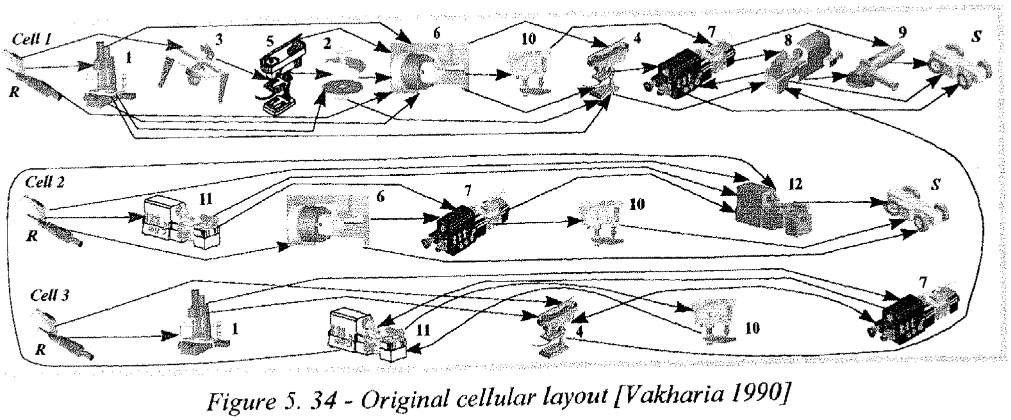


Figure 5. 34 - Original cellular layout [Vakharia 1990]

Figure 5.34 shows the intracell and intercell flows and layouts obtained in the considered example [Vakharia 1990]. The cells are not 100% independent and flows in cell 3 are bi-directional. The author state that "the fact that the underutilised machines were in the remaining cell 3 helped preserve the unidirectional flow patterns in the other cells".

However, close observation of the three flowlines showed that their machines have been placed consecutively in the flowline, and thus did not have any flow interactions. For example, machine pairs (1,

3) and (10, 4) in cell 1, or pairs (11, 6) and (10, 12) in cell 2 have been placed consecutively, although no parts move immediately from the first machine to the second. This problem of no flows between consecutive machines is expected with a pure undirectional layout. In such a layout two machines at most can be placed adjacent to any machine. So if a *TSP* or a linear *QAP* approach is used, the tendency to minimise average travel distances will make machines adjacent that may not be flowline connected. In contrast, *MWDRST* allows several machines to receive flow from a common predecessor around which they can be positioned. Furthermore, none of the machines in cell 3 had any immediate flow directions. In addition, machines 11 and 10 in cell 3, which are strongly connected, could have been placed after machine 7, to eliminate the backtracking existing in the sequence suggested by the author. A U layout could also have been designed for the flowline to allow crisscrossing flows to reduce the travel distances.

Cell No.	*Parts*	*Machine No. in cell*	*Number assigned*	*Average machine utilisation in cell (%)*
1	*1, 3 - 9*	1	1	80
		3	1	60
		5	1	50
		2	1	20
		6	1	67.50
		10	1	75
		4	1	45
		7	2	50
		8	1	60
		9	2	48.75
2	*11 - 19*	11	3	60
		6	1	37.50
		7	2	68.75
		10	2	75
		12	1	70.83
3	*2,10*	1	1	13.75
		11	1	45
		4	2	50
		10	1	18.75
		7	1	29.17

Table 5. 21 - Machine utilisation in cells [Vakharia 1990]

Table 5.21 shows the poor machine utilisation when intercell flows are restricted. Machines are distributed amongstst cells based on capacity calculations on a part family basis, only unidirectional intracell flow is allowed, and a pure flowline intracell layout is required. Cell 1 has an unusually large number of machines since it contains ten of the twelve shown in *Table 5.20.* Cell 3 caters for only two parts. The author (Vakharia 1990) has shown only machines 4 and 11 as being distributed amongst the cells *(Figure 5.34).* However, the intercell flows cause other machines such as 1, 6, 7 and 10 also to be shared. The cell based machine distribution results in higher flow requirements for machine 1, 4, 7 and 10 than the actual values in *Table 5.20.* For instance, the required pairs for machines 1, 4, 6, 7, 10 and 11 are (2:1), (2:2), (5:3), (4:3) and (4:3), respectively in the original layout. Furthermore, the poor utilisation of all the shared machines, assuming utilisation on a part family basis, can be clearly observed. For example, in cell 1, machine type 1 has 80% utilisation, which reduces to 13.75% in cell 3. In cells 1 and 2, the utilisation of machine type 6 is 67.50% and 37.50% respectively. Under a functional layout with limited intercell flows and intracell backtracking feasible, such poor machine utilisation due to physical cell formation can be avoided. Thus these critical intercell machine sharing aspects have not been addressed in this paper (Koenig 1981).

5.5.1.2.Step 2 and Step 3

Table 5.22 shows the travel chart (matrix form of the initial digraph) developed from the data of *Table 5.19*. This assumes that all operational sequences originate at an *R* (*IN* or source) node and terminate in an *S* (*OUT* or sink) node to maintain consistency with the original paper.

TO Machine

FROM Machine	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*S*	*R*
1		1		8		2	3							
2				1		1								
3					2									
4							9	11						
5		1				1								
6				4			3			3			3	
7				5				2	5	6	3	2	3	
8							3		8				2	
9													13	
10							3				3		6	
11							5			3		4		
12													8	
S														
R	14		2	2		9					6	2		

Table 5. 22 - Travel chart for CM layout design

5.5.1.3. Step 4 and Step 5

The travel chart (*Table 5.22*) was the only input to the first stage model and executed using *MATLAB*. *Table 5.23* summarises the *MATLAB* solution report. It gives the total weight of the *MWDRST* and lists the arcs.

From node	*To node*	*Weight*	*From node*	*To node*	*Weight*
R	1	14	4	7	9
R	3	2	4	8	11
R	6	9	7	10	6
R	11	6	8	9	8
1	2	1	9	S	13
1	4	8	11	12	4
3	5	2	***MWDRST with value 92***		

Table 5. 23 - Stage I output

Figure 5.35 shows the corresponding *MWDRST* rooted at the node *R*. The weightings of the individual arcs can be obtained from *Table 5.19*. Six paths are observed, with paths 4, 5, and 6 being attached to the root. Since the only branching node for these three paths is the root node (R), they could be attached to other machine nodes. Thus, from a flow perspective, the volume of backtrack flows would be reduced by the flows on at least three arcs (R to: 3, 6 and 11).

The basis for this argument is that when stock from the raw material stores reaches machines 3, 6 or 11 (first operation in the flow paths), clearly it has not been machined. Hence, there is no quality requirement arising from these first operational machines for being located together with other machines.

For the same reason, machine 2 in path 3 has not been moved. After the stock has been machined on machine 1, the second operational machine must be located closely. This will eliminate quality solution problems from the previous modelling stage. Hence, these flow paths could be connected to machines on other paths, instead of being attached to the root. Essentially, this amounts to selecting, from all arcs incident to each of machines 3, 6 and 11, these arcs which carry the highest flow volume. From a modelling perspective, arcs (R, 6) and (R, 11) were eliminated from the stage one, prior to the next stage of the model's solution.

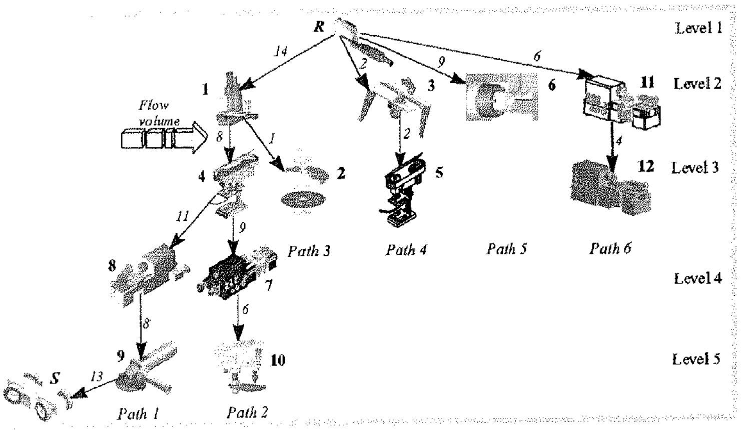


Figure 5. 35 - MWDRST – stage 1 flowline

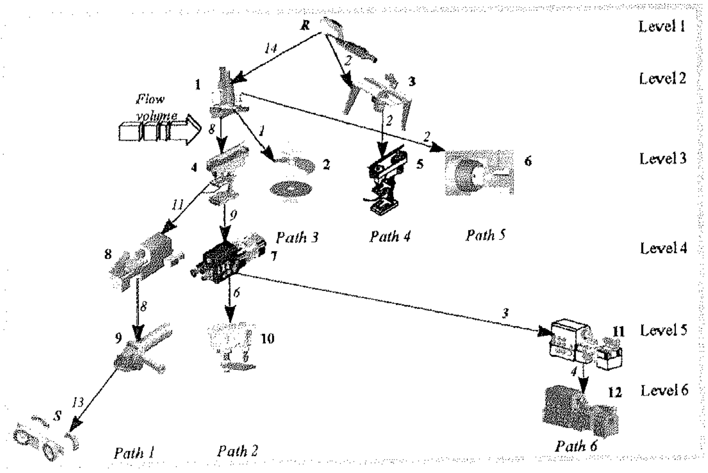


Figure 5. 36 - Modified stage 1 of MWDRST

From node	*To node*	*Weight*	*From node*	*To node*	*Weight*
R	1	14	4	7	9
R	3	2	4	8	11
7	11	3	7	10	6
1	6	2	8	9	8
1	2	1	9	S	13
1	4	8	11	12	4
3	5	2	***MWDRST with value 82***		

Table 5. 24 - Modified stage I MWDRST

Table 5.24 shows the modified set of arcs that constitutes the *MWDRST. Figure 5.36* shows the expected flow reduction of paths 5 and 6, allowing the earlier backtracking for arcs (1, 6) and (7, 11) to be eliminated. Path 4 could not be shortened since (R, 3) is the only arc incident to machine 3. Hence, the modified *MWDRST* shows an increase in the number of machines acting as branching nodes, giving the opportunity to reduce the backtrack flow distances on any flowline in the shop.

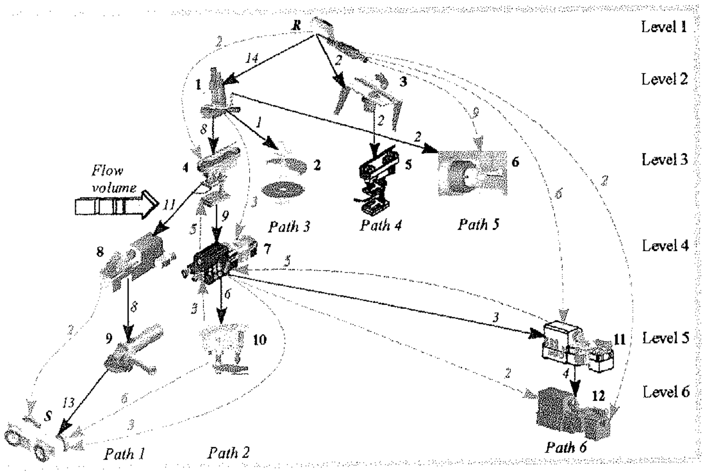


Figure 5. 37 - Forward and backward arcs in MWDRST

5.5.1.4. Step 6

Figure 5.37 shows the complete forward and backward arcs of the *MWDRST.* The effect of their flow volumes on the layout of the shop need not be analysed until some feasible layout configurations are developed after the second stage model (optimal permutations of the flow paths branches). The reason is that they involve flows within a flowline, and will not contribute to intercell flows in any way. This demonstrates the nature of the *MWDRST* structure in classifying whether flow arcs causes backtracking within a cell, or not. Unlike other cell formation methods, backtracking is based on an overall shop layout structure, instead of the operational sequence of each part.

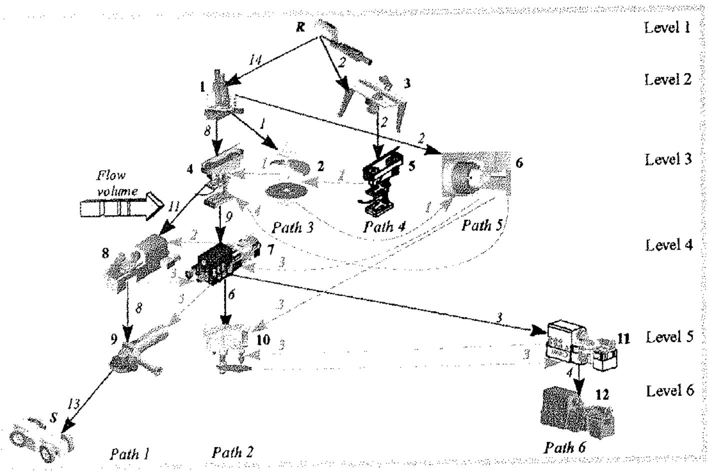


Figure 5. 38 - Initial crisscrossing arcs in MWDRST

5.5.2. Stage II: Steps 7 – 11

5.5.2.1. Step 7

The travel chart of the intercell flows necessary for the first stage, shown in *Table 5.25,* was developed by identifying the crisscrossing arcs in *Table 5.19.* The volume of intercell flows is relatively low. This suggests that the system of *DCM* flowlines obtained in the *MWDRST* has eliminated most of the flow arcs as forward or backward arcs, in the *MWDRST.* Since the number of crisscrossing arcs is small, they have been shown on the *MWDRST* in *Figure 5.38.* This permits the permutation of the branches of the

MWDRST in the second stage to be observed visually.

	P1	*P2*	*P3*	*P4*	*P5*	*P6*
P1						
P2	10					
P3	1	1				
P4	0	0	1			
P5	4	10	1	1		
P6	0	6	0	0	0	

Table 5. 25 - Intercell flows (crisscrossing)

Machine in cell	*Level*	*Weight assigned*
1	2	100
4	3	300
7	4	0

Table 5. 26 - Flow weights for shared machines

Table 5.26 shows the flow weightings assigned to machines (shared) 1, 4 and 7 since they are branching nodes relating two or more paths (the weights have been arrived at iteratively; see detailed explanation section 5.2.2.4). Not all of them need be treated as indivisible. Machines 1 and 4 are common to paths 1, 2 and 3. Only one machine of type 1 is required (two available) and two of type 4 are required and available. Hence, there is need to keep paths 1, 2 and 3 adjacent to each other while allowing permutations amongst them. Since machine 7 is common to two paths, and three of its type are required (five are available), no weighting was assigned to it. Thus, this would allow the second stage solution to place path 1 between paths 2 and 6, without forcing the dividing of machine types 1 and 4. This is the approach of constrained machine duplication to minimise intercell flow distances adopted in the proposed dynamic cell formation method.

	P1	*P2*	*P3*	*P4*	*P5*	*P6*
P1						
P2	400					
P3	100	100				
P4	0	0	0			
P5	100	100	100	0		
P6	400	400	100	0	100	

Table 5. 27 - Intercell flows (shared machines)

	P1	*P2*	*P3*	*P4*	*P5*	*P6*
P1						
P2	410					
P3	101	101				
P4	0	0	1			
P5	104	110	101	1		
P6	400	406	100	0	100	

Table 5. 28 - Combined intercell flows for stage II

5.5.2.2. Step 8

The intercell flow weights due to the shared machines which must be added to the values in *Table 5.25* are shown in *Table 5.27*. The combination of *Table 5.25* and *Table 5.27* must be the input to the travel chart for the model second stage *(Table 5.28)*. The upper bound obtained with the *SDPI* method was $

6498 (for the material flow unit cost of $1). As shown previously, in section 5.1, this was obtained with a pairwise interchange heuristic (Appendix C) for solving an *OLA* in six nodes. The linear arrangement of the six paths obtained was: 6, 2, 1, 5, 3, 4; compared to the initial arrangement shown previously as 1, 2, 3, 4, 5, 6.

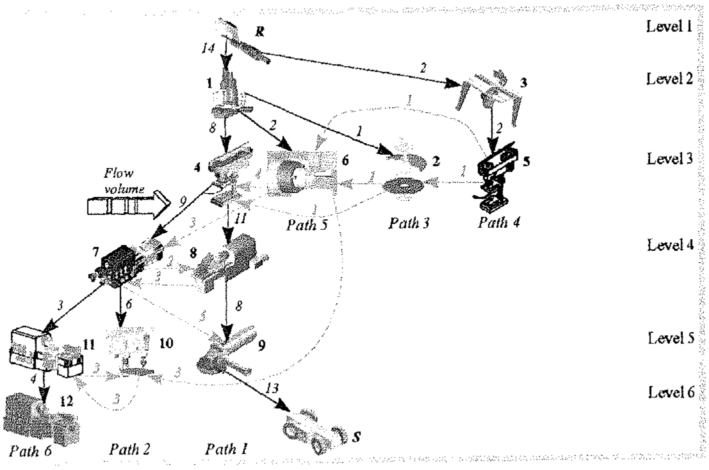


Figure 5. 39 - Crisscrossing arcs after MWDRST stage II

The modified *MWDRST* output from the second stage of the methodology is shown in *Figure 5.39*. The small reduction in the number of arcs spanning nonadjacent paths in the *MWDRST* leaves arcs (2,4), (5, 6), (6, 7) and (6, 10) creating the intercell flows. A possible approach to conducting a machine analysis to eliminate these flows, after a preliminary layout has been developed, is discussed in the next section.

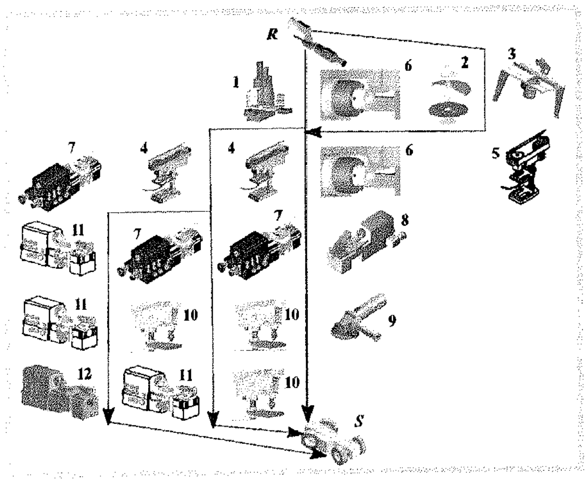


Figure 5. 40 - DCM flowline layout after stage II of MWDRST

5.5.2.3. Step 9

Figure 5.40 shows a possible layout developed for the individual flowlines, by providing aisles with intercell flows, which are allowed amongst adjacent flowlines. The approach used for the rudimentary example in this Chapter 5 was used initially. Thus, additional machines of each type were inserted into empty locations adjacent to the previously located machines by inspection of the crisscrossing arcs. The operational sequence of a part can now be matched to machines, either within a flowline, or in an adjacent flowline. Hence, without considering part family formation using operational sequences, the resultant flowlines are defined in layout form by the final *MWDRST*. Thus all machines have been retained in functional layouts and the intercell adjacencies have been used as a guideline for locating identical machines.

5.5.2.4. Step 10

Arcs (2, 4), (5, 6), (6, 7) and (6, 10) constitute the minimal set of arcs which create intercell flows between nonadjacent cells. Only those parts whose operational sequences contain these arcs need to be considered for capacity calculations, process planning or value analysis, in order to eliminate the intercell machine sharing problems created. The approach that was used breaks the operational sequences in the predecessor and successor strings (graph theory and network analysis applied to *DCM* flowline layout analysis), as shown in *Table 5.29.*

Arcs	***Part***	***Batch quantity***	***Predecessors***	***Successors***
2,4	3	1	1,2	4,7,8,9
5,6	9	1	3,5	6,4,8,9
6,7	18	3	6	7,10
6,10	5	2	1,6	10,7,9
6,10	6	1	6	10,7,8,9

Table 5. 29 - Data for planning machine sharing

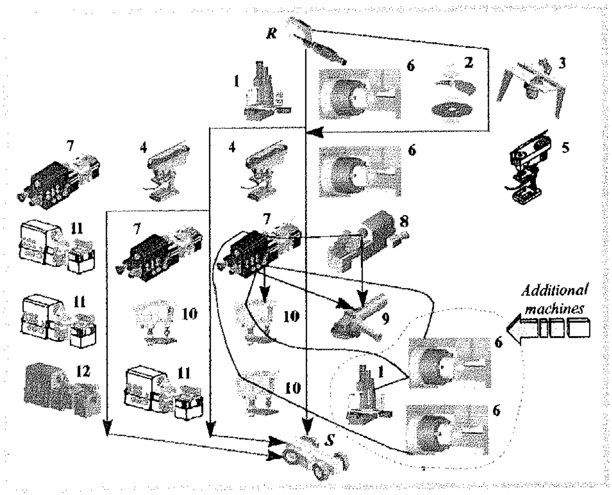


Figure 5. 41 - Resulting DCM Layout after duplication of machines is applied

With reference to *Table 5.29,* duplication of machines 3, 5, 8 and 9 is infeasible (since they do not have any additional machine of its type). Arcs (2, 4) and (5, 6) correspond to exception operations. Neither their predecessor nor their successor machines can be duplicated. Thus, these arcs will correspond to the exception operations causing intercell flows for which the alternate routes need to be identified. In the case of arcs (6, 7) and (6, 10), machines 1 and 6, from *Figure 5.41* can be duplicated to eliminate the intercell flows due to parts 5, 6 and 18, since both machine types 1 and 6 are additional. This figure shows how the duplication of machines 1 and 6 now allows the operational sequences of these three parts to be accommodated amongst machines, which are included in the adjacent flowline. It may be noted that in the successor set of machines, machine 8 does not have any additional machine of its type.

Also, duplication of machines 7, 9 and 10 would require, before economical layout adjustments, for a larger number of machines and fail to eliminate the intercell flow due to arcs (7, 8) and (8, 9). Hence, the proposed methods would also consider the detailed part data at a later stage, when only those parts creating intercell flows and machine sharing problems need to be considered.

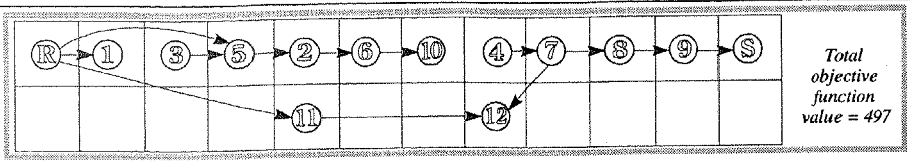


Figure 5. 42 - Original machines as a branched flowline [Vakharia 1990]

5.5.2.5. Step 11

Figure 5.42 - *Figure 5.50* shows a variety of layout evaluations that were conducted. *Figure 5.42* shows the original set of three unidirectional flowlines assimilated into parallel flowlines with no machine duplication. This layout was developed since it was observed that cell 1 contained a disproportionately larger number of machines compared to cells 2 and 3. Furthermore, cell 3 processed only two parts and did not contain any unique machines, for example, all its machines were common to the other two cells or involved in intercell flows.

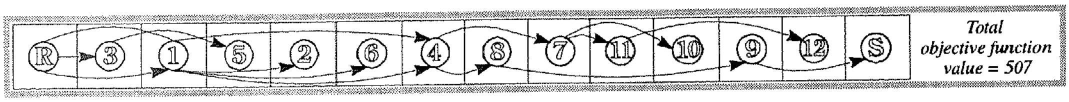


Figure 5. 43 - Bidirectional flowline layout

Figure 5.43 shows the *QAP* solution (linear) for a pure flowline for all twelve machines with no intracell machine duplication. Thus, the flow penalty is only marginally worse than the previous layout. The arcs in the *MWDRST* have been shown in this layout to increase in their lengths. Further, it fails to capture the feature, or the *MWDRST* required to partition the digraph for the travel chart in directed paths sharing one or more machines.

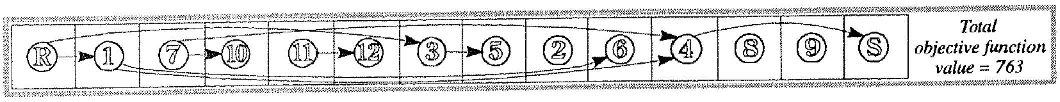


Figure 5. 44 - Unidirectional flowline layout

5.5.2.6. Improved Procedures

Figure 5.44 - 5.47 demonstrates the conversion of a unidirectional flowline obtained, using a new *TSP* improved *SA* algorithm (programmed in *MATLAB* and presented in appendix C), into U, S and serpentine layouts to reduce intracell travel distances without affecting flow directions. Dummy costs were used to convert the travel chart into a complete digraph, with zero cost arcs connecting *R* and *S* nodes. This repositioning requires these two nodes to be at the ends of the route chart. This *SA* improved algorithm for the *TSP* problem is employed here, to test the ability of the algorithm, to learn about how it works, and was an improved building 2-opt (presented within Appendix C) procedure and local search to bridge the *SA* weaknesses. However, more importantly, although when it was employed it gave the same result, it produced much better computational times and the ability to work with bigger sizes of input data then commonly *TSP* employed algorithms. This knowledge was needed as it was very important for the next two stages of this methodology, where the *DCM* layout displacement will be solved automatically.

Figure 5.48 is the layout implemented for the post second stages of the *MWDRST*. The significantly low value of the objective function (defined in Chapter 4 as total distance moved by all parts) clearly demonstrates that, assuming crisscrossing flows amongst flowlines, the proposed method also suggests the outer shape that the shop must have, to lower material handling costs for intercell flows. Thus an aisle between each line (row) of machines can be assumed without affecting the assumption of rectilinear flow travel distances. This is an improvement on the *CRAFT*-type design programs, which only work with pairs of machines, assumes a rectangular perimeter for a layout, and ignores travel distances along aisles, when flows involve non adjacent pairs of machines.

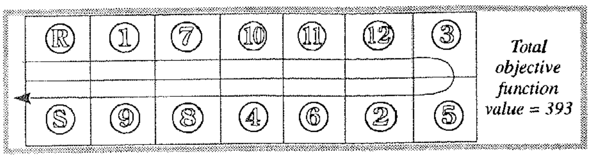


Figure 5. 45 - U Unidirectional flowline layout

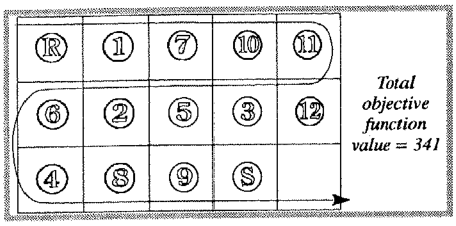


Figure 5. 46 - S layout for unidirectional flowline

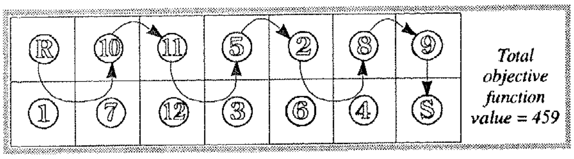


Figure 5. 47 - Serpentine layout for unidirectional flowline

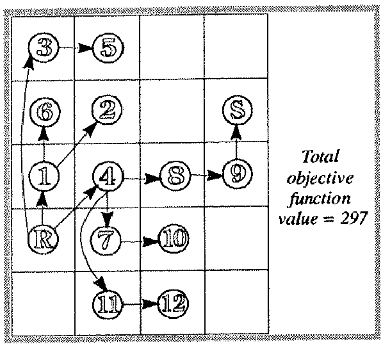


Figure 5. 48 - MWDRST final layout

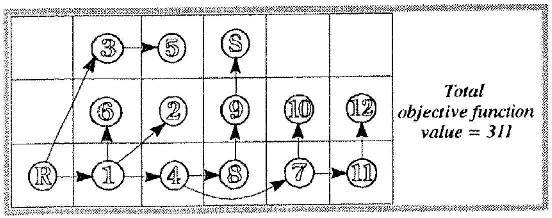


Figure 5. 49 - L shaped layout flowlines

The layout (*Figure 5.48*), suggested by the *MWDRST,* shows several vacant nodes in the rectilinear grid. *Figure 5.49* shows the flexibility of the *MWDRST* to conform to restrictions on the perimeter of the shop floor. Without losing the original flow or adjacency structures, the longer flowlines have been given an L shape with minimum effect on the objective function, which was the minimisation of the material flow distance.

It may be noted that the *SA* algorithm was utilised for solving the *TSP* problem of 100 locations, which is more complicated, although for this example *SA* gave the same bound solution as the branch and bound algorithm. Thus the output results are presented in the Appendix C and *MATLAB* codes in the Appendix D. In addition to the experimental study *EA* (Evolutionary Algorithm) is employed to a *TSP* problem, with the results presented within Appendix C (including theoretical study) and the *MATLAB* program codes in Appendix D. For both combinatorial optimisation algorithms, the same bound is given, with quicker times then the heuristic algorithms. However, although at this stage *SA* performed better for the smaller problem detailed in Appendix C, a further study of *EA* is needed.

5.5.3. Comments

The unidirectional flowline alone provides an unfavourable layout configuration for machine sharing. However, it has the unique property that it can conform to the overall shape of the machine working envelope, with respect to any other machine, along a circular aisle. *Figure 5.50* shows a combination of a *TSP* based flowline and a centrally located common facilities section. The flowlines of *Figure 5.44* have been configured around a central group of shared machines (machines 1, 4, 6, 7 and 10, extracted from the overall sequences of machines). Although this configuration fails to improve upon the *MWDRST* based layouts, it is better than the U, S and serpentine configuration tested previously. Hence, the structures of the *MWDRST* and *TSP* have been shown as flexible tools for flowline based layout design.

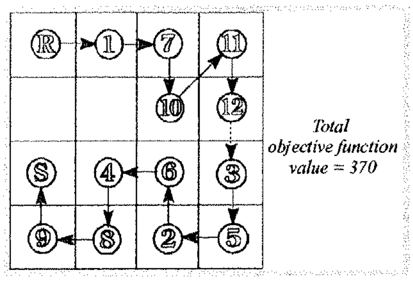


Figure 5. 50 - Facilities with flowline layout

5.6. Comparison of *MWDRST* with Graph-Based Layout Design

This example helps to compare the proposed *DCM* layout design method with a general purpose package for functional and cellular layouts, *PLANTAPT* [Carrie 1977]. This author implemented spanning tree and maximal planar graph algorithms for layout designs. The package is capable of performing multiple analysis for cell formation and layout design, such as machine part grouping, part family formation based on similarity of operational sequences, intracell flowline design and layout planning. Unlike traditional machine tool metalcutting applications, this package deals with processes such as flame cutting, presswork, shearing, drilling, planning and welding. Some of these processes, such as flame cutting, welding, cleaning and fitting, cannot be located in a cell, apart from machines such as presses. Hence, this example describes flow based cell formation in the operational sequences of the parts, which are located external to their cell.

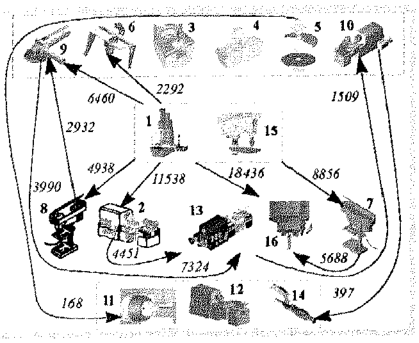


Figure 5. 51 - Original layout for guillotine cell with volumes of material flows [Carrie 1977]

The original layout obtained using *PLANTAP* is presented in *Figure 5. 51*, where the major flow arcs are shown prominently. Due to the incompatibility of machines 9, 6, 3, 4, 5 and 10 (burning, welding, cleaning and fitting), they have been located away from the other machines. Similarly, since machines

11,12 and 14 appear to perform assembly operations, they have been located adjacent to machines 1 to 15, 13 to 16, 7, 8, and 2. These constraints, based on machine grouping performance, are best considered using the flow based cell formation method. *Table 5.30* lists the machine types and a description of the basic process performed by each.

Machine type	***Machine name***	***Operations number***
1, 15	Guillotine	776
2	Rail bender	182
3	Floor weld	1
4	Floor clean	4
5	Floor fit up	1
6	Profile burn	43
7	6' Radial drill	137
8	9' Radial drill	113
9	Hand burn	244
10	Floor clean	14
11	Degrease	10
12	Dip paint	7
13, 16	Brake	492
14	Booth	7

Table 5. 30 - Machine loads and availability data

TO Machine

FROM Machine	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
1		1153				2292	8856	4938	6460				1843	
2							1725	930	1582				4551	
3														
4														
5											168			
6		216					150						1504	
7				73	168	75			470	246			5688	
8			1			297			2932				2137	
9	90	883		146			540			45			7324	
10											192			397
11												227		40
12														
13		503					519	75	3990	1509	75			
14														

Table 5. 31 - Travel chart for GT layout [Carrie 1977]

5.6.1. Stage I: Steps 1 - 6

Table 5. 31 shows the travel chart for the material flow between the different machines that were used as input to the first stage. Since the different stages in the analysis have been described in detail in the previous two examples, only the essential results for this example are presented.

Figure 5. 52 shows the *MWDRST* after the first stage. The guillotines, which perform the first operation on most sheet metal parts, act as the root node for the four paths. An interesting feature is that these paths have no other intermediate shared machines (*MWDRST* has no branching nodes). Hence, in the second stage, which has been simplified to a simple four-node permutation problem, additional weightings need not be given to any nodes to force paths containing them to be adjacent to each other.

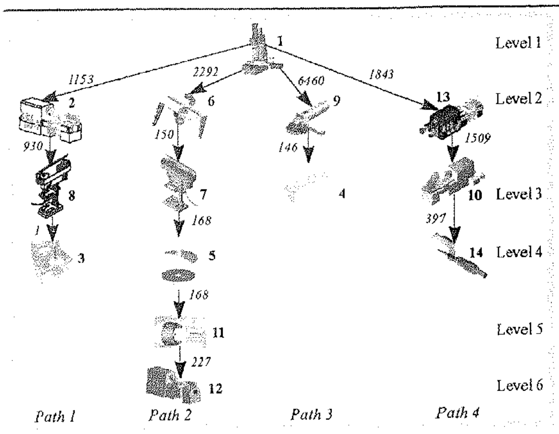


Figure 5. 52 - MWDRST arcs after stage I

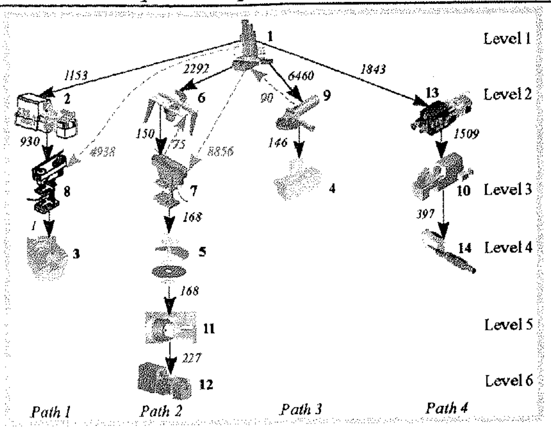


Figure 5. 53 - Forward and backward arcs in MWDRST

The incompatible machines and assembly machines have been temporarily included in the paths. Their locations were adjusted after the second stage solution was obtained. *Figure 5.53* shows the few forward and backward arcs identified, whilst *Figure 5.54* shows the *MWDRST* with the crisscrossing arcs prior to the second stage.

	P1	**P2**	**P3**	**P4**
P1				
P2	2238			
P3	5397	1083		
P4	11181	8264	11539	

Table 5. 32 - Intercell flows

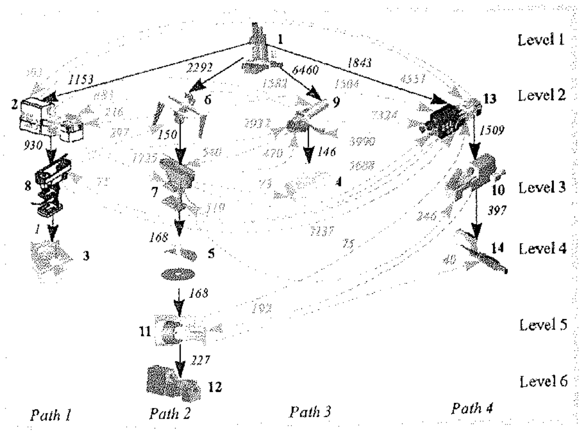


Figure 5. 54 - Crisscrossing arcs before stage II

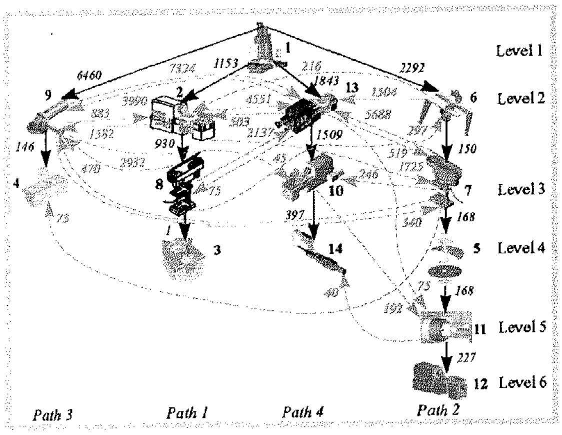


Figure 5. 55 - Crisscrossing arcs after stage II

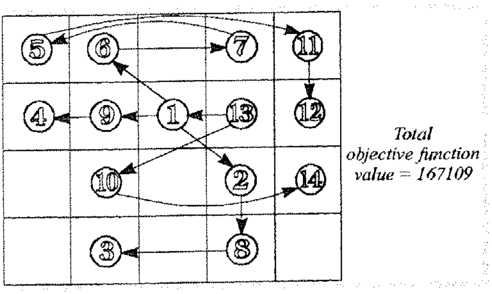


Figure 5. 56 - Original layout [Carrie 1977]

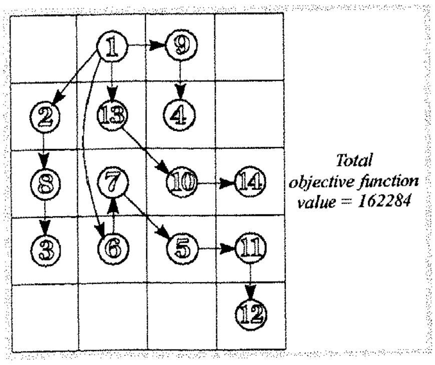


Figure 5. 57 - Final layout using the proposed method

5.6.2. Stage II: Steps 7 - 11

Table 5.32 shows the intercell flow input to a program for a four node *QAP - SDPI* heuristic, and the solution for the second stage can be expected to be optimal. Optimally is guaranteed, because the *SDPI* algorithm gives a optimal solution for a two dimensional *OLA*, which is a more complicated problem then the one dimensional *OLA* problem here. The new configuration for the *MWDRST* is shown in *Figure 5.55*.

Figure 5.56 presents the rectilinear grid presentation for the original layout and *Figure 5.57* shows the grid obtained using the final *MWDRST*. The locations of the incompatible machines and assembly in this layout were actually obtained from the final *MWDRST*. The improvement in the objective function (presented in Chapter 4), the total distance moved by all parts, and the costings by the proposed method is quite small. However, it is able to correctly group and locate compatible machines, incompatible machines, and unidirectional assembly machines, which are not possible with the considerable backtracking in the original layout. This is achieved mainly by the L - shaped layout for the incompatible processes section, which allows the machine tools to be reached from two sides.

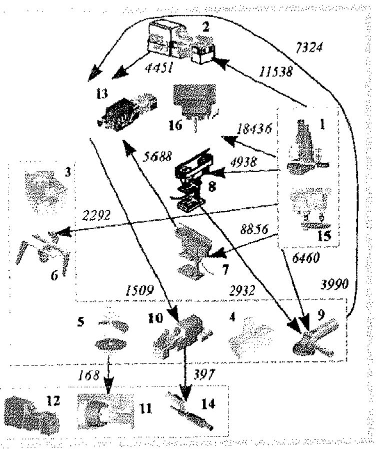


Figure 5. 58 - DCM layout developed from proposed method

Figure 5.58 shows a possible approximate layout for the dynamic cell from Carrie's paper, including aisles to separate the machine tools from the incompatible processes section. Hence, the machine sequences and path adjacencies of the *MWDRST* are a viable alternative for existing layout design algorithms. Although the final *DCM* layout (approximate) configuration is currently developed manually, the process is guided by analytically suggested machine groups and cell adjacencies.

5.7. Conclusion

In this chapter a number of illustrative studies are presented. These studies compare the proposed first two stages of the proposed methodology with two other graph adjacency methods. Three other different *CM* design methods are also detailed. An improved procedure, were the *SDPI* algorithm is employed, is also applied with respect to these three methods. Hence, results from the proposed method, in combination with other techniques and algorithms (which are coded in *MATLAB*), are compared with results from packages such as *STORM*. Thus, this work can quickly evaluate a finite number of *DCM* layout alternatives which conform to the machine grouping, flowline layout and intercell machine adjacencies given by the second stage of the *MWDRST*. This helps to integrate the effects of reducing the lengths of individual material flowlines by increasing the width of the shop (converting a rectangle to a square with the same area).

These results show that the proposed method correctly models the immediate adjacencies of machines and the groups to which they could be assigned, if cells were desired. Thus, flow distances and machine locations can be determined subsequently. Hence, the evaluation of alternative *DCM* flowline layouts, using the simple objective (total distance) function, coded in the *MATLAB* package, was adequate for these analyses. In addition, it is shown that combinations of a *TSP* based flowline and centrally located common facilities section fails to improve upon the *MWDRST* based layouts. However, the structures of the *MWDRST* and the *TSP* have been shown as flexible tools for flowline based layout design.

Stages	*Steps*	*Main Tools*	*Description of Steps*
I	1	*Matrix form*	*Input* 1. Batch quantity for each part *Qk* 2. Operation sequences for each part *Sk* 3. Number of machines of each type available *Ni*
	2	*MATLAB matrix form*	Compute the distance between every pair of machines *i* and *j* and construct the distance matrix
	3	*MATLAB*	Generate initial digraph *D(v,a)*
	4	*MATLAB Chu-Tarjan algorithm*	Solve first stage to generate *T* Maximum Weighted Directed Rooted Spanning Tree *(MWDRST)*
	5	*Tree draw*	Generate *path (1)... path(p)* in the *MWDRST*
	6	*Tree draw*	Identify arcs in forward, backward and crisscrossing paths of the digraph *D*
II	7	*MATLAB matrix form*	With crisscrossing paths and *path(1)...path(p),* generate combined interpath flow matrix using total flow for machines *i* and *j*, and weighting for the shared machines
	8	*MATLAB QAP*	Solve second stage to pivot paths – to rearrange *MWDRST*
	9	*MATLAB matrix*	Plan layout based on material flowlines from second stage and path adjacencies
	10	*Tree draw*	Identify arcs in crisscrossing to plan for machine duplication in nonadjacent cells
	11	*MATLAB SDPI*	Design and analysis of approximate *DCMS's* flowline layout
III	12	*MATLAB*	Economical capacity oriented machine duplication
IV	13	*MATLAB SA*	Economical optimisation of *DCMS's* layout using *SA* algorithm minimising total material handling and duplication cost's
	14	*MATLAB SA*	Shop floor layout displacement

Table 5. 33 - Improved research procedure after experimental study completed

Combinatorial optimisation algorithms are also employed to solve *NP*-problems as part of the improved procedure. These are programmed in the main mathematical programming tool *MATLAB* and the codes presented in Appendix D. These optimisation algorithms, *QAP*, *SA*, *TSP*, *2-opt* and *EA*, have been tested with good results (improved computational time, robustness). Although a short review is included in this chapter, a more detailed explanation about combinatorial optimisation algorithms used in this research is presented in Appendix C. The main conclusions from *SA* and *EA* is the better performance of *SA* compared with *EA* for cases up to 5000 locations (very good for industrial scenarios). However, the *EA* area of research needs further work, especially in applying tree directionality into algorithms (coding and encoding with "Prufer" number-enumeration of the tree).

The main goal of the first two stages in this proposed research was to capture the directionality embedded in the operation sequences of a variety of parts produced within a desired facility layout. This goal was achieved by incorporation of directionality into the design skeletons using the Maximum Weight Directed Rooted Spanning Tree. Adequate theoretical elaboration has been presented with comparative experimental studies.

The algorithm presented in this chapter groups the production equipment of the manufacturing facility into dynamic cells, and forms a set of part families, which are processed within these cells. It does not, however, address the relative placement of the machines within cells or relative placement of the cells on the shop floor. Both these issues are critical in terms of the total material flow within the shop and are addressed in the next Chapter 6. The proposed approach is based on the method of simulated annealing, which also will be presented in Chapter 6.

CHAPTER 6

Optimising Dynamic Cell Facility (Shop Floor) Layout – Stages III & IV

6.1. Introduction

This Chapter describes stage III of the proposed research methodology and considers the economical machine duplication model with an illustrative example. Also addressed is the proposed research methodology for stage IV, which is the facility layout problem for a dynamic cell shop consisting of both cell and independent machines. The major issue here is the placement of manufacturing resources (cells and machines) within the available area of the facility. A simulated annealing *(SA)* based algorithm is developed, improved and utilised to obtain a near optimal solution for the layout problem, which is *NP* complete. The general problem is defined along with the solution approach and an illustrative example.

6.2. Stage III Economical Machine Duplication

The stage II approach, which breaks the operational sequences in the predecessor and successor strings (graph theory and network analysis applied to *DCM* flowline layout analysis), was presented in Chapter 5, *Table 5.29.* In order to eliminate the intercell machine sharing problems created, only those parts whose operational sequences contain these arcs (predecessor and successor) need to be considered for capacity calculations, process planning or value analysis.

This section describes the approach for economical duplication of the material flowline layout, which is an extension of the research methodology presented within stages I and II. Here, in stage III, additional system/machine variables are included for finding the required number of the flowline machines. Such variables are:

- Setup time, duration of the production period, machine capacity, utilisation of each machine type, flow volume, production volume, operation time, non-operation time, average time between failures and servicing time of the machine type, and availability of the machine type.

In order to duplicate machines in several flowlines, where machines cannot be freely duplicated, the mathematical model must compute the total capacity per machine of each type for the entire production period. This is presented as follows:

$$A_j = \Delta U_j C \qquad \textit{Equation 6. 1}$$

where:

i – part index $(i = 1,2,3,\ldots,n)$,

j – machine index $(j = 1,2,3,\ldots,m)$,

A_j – total available capacity per machine of type j,

C – duration of production period, and

ΔU_j – utilisation factor of machine type j, and defined as follows:

$$\Delta U_j = \frac{\sum_{i=1}^{n} PT_{ij}}{PA_j} = \frac{PPV_i\left(\frac{ST_{ij}}{B_i} + TOT_{ij}\right)}{MA_j PDL_j} = \frac{PPV_i\left(\frac{ST_{ij}}{B_i} + OT_{ij} + NOT_{ij}\right)}{\left(\frac{ATBB_j + TNS_j}{ATBB_j}\right)PDL_j} \qquad \textit{Equation 6. 2}$$

where:

PT_{ij} – total production time. This is time to produce a batch flow volume during the production period (in volume units), of the part i on the machine j (time/period time), and defined by *Equation 6.3,* ie.

$$PT_{ij} = PPV_i\left(\frac{ST_{ij}}{B_i} + TOT_{ij}\right) = PPV_i\left(\frac{ST_{ij}}{B_i} + OT_{ij} + NOT_{ij}\right) \qquad \textit{Equation 6. 3}$$

where:

PPV_i – production volume of part i for production period (unit/period time),

ST_{ij} – setup time for the part i on the machine type j (time/batch),

B_i – flow volume for the batch, for the part i (unit/batch),

TOT_{ij} – total processing time of the part i on machine j (time/unit),

OT_{ij} – operation time for processing part i on the machine type j (time/unit),

NOT_{ij} – non-operation time for processing part i on machine type j (time/unit), ie. that non-value added time at machine j, such as machine loading and unloading, and

PA_j – availability of the machine type j for the production period (time/period time) defined by *Equation 6.4,* ie.

$$PA_j = MA_j PDL_j = \left(\frac{ATBB_j + TNS_j}{ATBB_j}\right)PDL_j \qquad \textit{Equation 6. 4}$$

where:

MA_j – availability (ratio) of the machine type j (time/production period time as a %),

PDL_j – length of the production period for the machine j (time),

$ATBB_j$ – average time between failures of the machine type j (time), and

TNS_j – average time needed for servicing machine type j (time).

Thus, the numbers of machines required in a flowline can be calculated as follows:

$$N_{jk} = \left\lceil \sum_i \frac{T_{ijk}}{A_j} \right\rceil \qquad \text{Equation 6. 5}$$

where:

N_{jk} – number of machines of type j required in flowline k, and

T_{ijk} – capacity requirement for operation i on machine type j in flowline k.

Thus *Equation 6.5* gives the economical required integer number of machines for each flowline in the *DCM* layout design. This duplication is necessary because the machine type processing time exceeds the flowline machine's availability in the dynamic cells (very common in the design of manufacturing systems). Thus, if the machine is a bottleneck machine then it can be duplicated in the flowlines (paths), where the corresponding part volumes are required. This procedure is now utilised in an example taken from Chapter 5, which is now detailed in the next section. Finally, an interactive chart for the design of the *DCM* layout (stages three and four of the proposed research methodology) is presented in *Figure 6.1.*

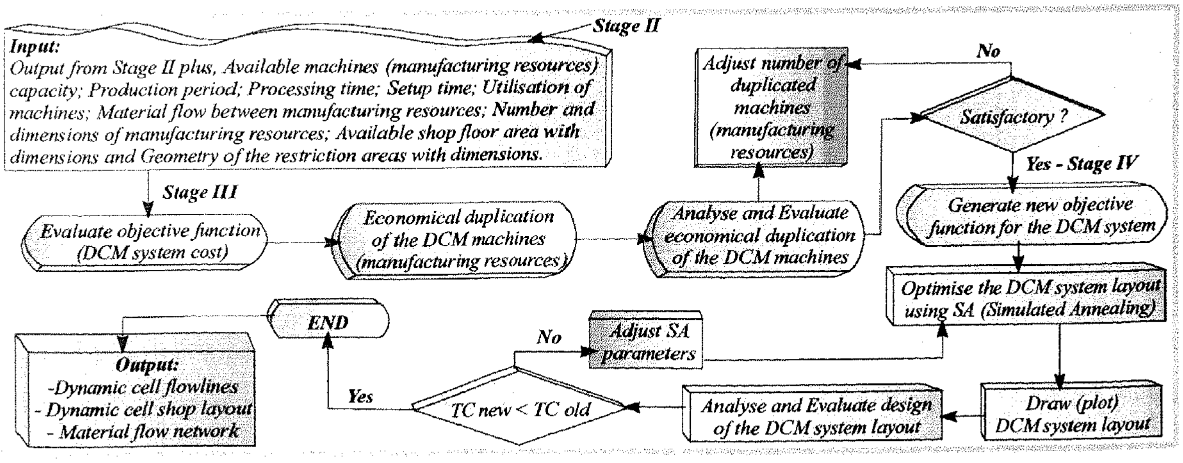

Stage II
Input:
Output from Stage II plus, Available machines (manufacturing resources) capacity; Production period; Processing time; Setup time; Utilisation of machines; Material flow between manufacturing resources; Number and dimensions of manufacturing resources; Available shop floor area with dimensions and Geometry of the restriction areas with dimensions.
Stage III
Evaluate objective function (DCM system cost)
Economical duplication of the DCM machines (manufacturing resources)
Analyse and Evaluate economical duplication of the DCM machines
Adjust number of duplicated machines (manufacturing resources)
No
Satisfactory ?
Yes - Stage IV
Generate new objective function for the DCM system
Optimise the DCM system layout using SA (Simulated Annealing)
Draw (plot) DCM system layout
Analyse and Evaluate design of the DCM system layout
TC new < TC old
No
Adjust SA parameters
Yes
END
Output:
-Dynamic cell flowlines
- Dynamic cell shop layout
- Material flow network

Figure 6. 1 - Interactive stages III & IV of the DCM layout framework based on SA

6.2.1. Economical Duplication - Example

In this section the example for consideration is derived from a paper (Vakharia 1990), which was used also for comparison of the *MWDRST* in the proposed stage II research methodology in Chapter 5.4. Detailed analysis of the input data was presented in the section 5.4.1 with the first two stages of the solution of the proposed research methodology. *Table 6.1* presents the minimum input data requirement for stage III, the economical duplication of machines, and the final result *(Figure 6.2)* from stage II of the proposed research methodology, together with the From – To machine material flow matrix (*Table 6.2*).

Part No.	Sequence	Total batch time (min's) per machine	No. of Batches	Batch quantity
1	1,4,8,9	96, 36, 36, 72	1	2
2	1,4,7,4,8,7	36, 120, 20, 120, 24, 20	1	3
3	1,2,4,7,8,9	96, 48, 36, 120, 36, 72	1	1
4	1,4,7,9	96, 36, 120, 72	1	3
5	1,6,10,7,9	96, 72, 200, 120, 72	1	2
6	6,10,7,8,9	36, 120, 60, 24, 36	1	1
7	6,4,8,9	72, 36, 48, 48	1	2
8	3,5,2,6,4,8,9	144, 120, 48, 72, 36, 48, 48	1	1
9	3,5,6,4,8,9	144, 120, 72, 36, 48, 48	1	1
10	4,7,4,8	120, 20, 120, 24	1	2
11	6	72	1	3
12	11,7,12	192, 150, 80	1	1
13	11,12	192, 60	1	1
14	11,7,10	288, 180, 360	1	3
15	1,7,11,10,11,12	15, 70, 54, 45, 54, 30	1	1
16	1,7,11,10,11,12	15, 70, 54, 45, 54, 30	1	2
17	11,7,12	192, 150, 80	1	1
18	6,7,10	108, 180, 360	1	3
19	12	60	1	2

Table 6. 1 - Operations sequence and batches data [source Vakharia 1990]

TO Machine

FROM Machine	1	2	3	4	5	6	7	8	9	10	11	12	S	R
1		1		8		2	3							
2				1		1								
3					2									
4							9	11						
5		1				1								
6				4			3			3			3	
7				5				2	5	6	3	2	3	
8							3		8				2	
9													13	
10							3				3		6	
11							5			3		4		
12													8	
S														
R	14		2	2		9					6	2		

Table 6. 2 - Stage III initial material flow machine matrix

For this illustrative example the first step is to duplicate machines in several material flowline paths utilising *Equation 6.5*. Previously, to find the number of machines of each type in each module, it was assumed that each machine type must be an integer number, shared between material flowlines (shown in *Figure 6.2*) and assigned to only one material flowline path. Analysing the available data from Vakharia [1990] it is clear that all required information (such as setup time and machine availability) is not available for the short duration of the production period (one shift). However, from the author's industrial experience, it is common to assume an average utilisation of 80%. With these assumption, ie. $\Delta U=80\%$ and $C=8$ hours, the available capacity $A=384$ minutes, and these are the parameters utilised in *Equation 6.4*.

Thus the economical numbers of each type of machine required in the different flowlines were calculated and are shown in *Table 6.3*. This table also shows that in certain cases the integer requirements of machines summed over all the flowlines exceeds the available numbers of machines of each type. This is an easy problem to solve if extra machines of these types can be requisitioned for the flowlines (or if some of the flowlines can be partially replaced by multifunction machines).

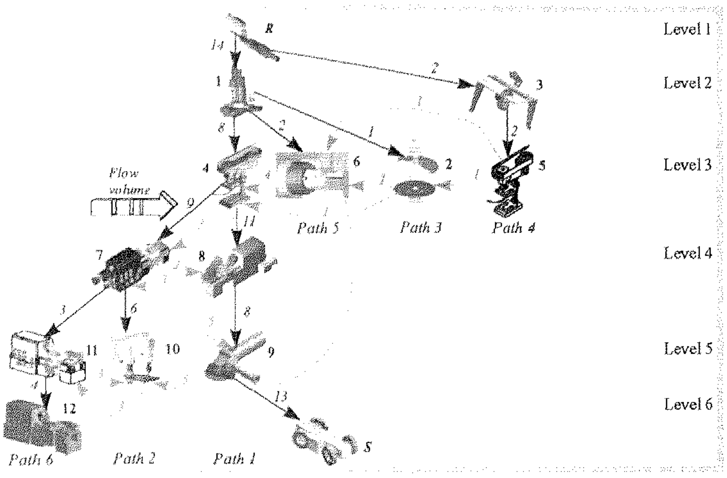


Figure 6. 2 - Material flow lines of DCM after stage II

	Machine requirements in the flowline paths						*Machines*	
Machine	***Path 1***	***Path 2***	***Path 3***	***Path 4***	***Path 5***	***Path 6***	***Available***	***Stage III Requirements***
1	0	1.17	0	0	0	0	2	2
2	0	0	0.25	0	0	0	1	1
3	0	0	0	0.75	0	0	1	1
4		1.8125	0	0	0	0	2	2
5	0	0	0	0.625	0	0	1	1
6	0	0	0	0	1.3125	0	4	2
7	3.7235	0	0	0	0	0	5	4
8	0	0.75	0	0	0	0	1	1
9	0	1.2187	0	0	0	0	1	2
10	2.9427	0	0	0	0	0	7	3
11	0	0	0	0	0	2.8125	3	3
12	0	0	0	0	0	0.89	1	1

Table 6. 3 - Machine requirements for flowline paths

However, after completing the machine requirements for the flowlines, and in order to eliminate the intercell machine sharing problems, a further part analysis was required. Here the operational sequences contain arcs, and these were considered for capacity calculations, process planning or value analysis. As stated previously the approach that was used breaks the operational sequences in the predecessor and successor strings (graph theory and network analysis applied to *DCM* flowline layout analysis), as presented in Chapter 5.

Because of having to change the assignment of the flowlines resulting from introducing a duplication of the machines, evaluation of the material flowlines network is then required. A new From – To machine chart (matrix) is then developed, and presented in *Table 6.4*. This travel chart *(Table 6.4)* is the main output of stage III, and consequently the input for the stage IV, the economical optimisation of the layout design. Here the objective function is to minimise the total system cost which is made up of the material handling cost and the duplication cost (presented in the next section).

Table 6.4 presents the material flow between machines in the dynamic cells, where the same machine type has the same initial number (ie. *machine 7 type = 7, M7, N7, O7*). Comparing the average utilisation *(Table 6.3)* of the proposed dynamic cell system, derived from the research methodology, with the utilisation of the cellular system from the Vakharia paper [1990] *(Table 5.21)*, shows improvement in the average utilisation. From *Table 6.3* the average utilisation of the machines in the dynamic cell is 77% (system utilisation) and is higher than the average utilisation of 51 % from the classical cellular system of Vakharia *(Table 5.21)*.

TO Machine

FROM Machine

	M1	*1*	*2*	*3*	*M4*	*4*	*5*	*M6*	*6*	*7*	*M7*	*N7*	*O7*	*8*	*M9*	*9*	*M10*	*N10*	*10*	*M11*	*N11*	*11*	*12*	*R*	*S*
M1					6	2			2		1		2												
1			1																						
2					1				1																
3							2																		
M4										6	3														
4														11											
5			1						1																
M6						2						3							1						3
6						2													2						
7						5								2											
M7															3			3		1					
N7																	3						1		
O7															2					1	1		1		3
8													3			8									2
M9																									5
9																									8
M10																					1				3
N10										1			2								1				3
10											3										1				
M11												1					1		1						
N11													1					1					3		
11																							1		
12																									8
R	13	1		2	2			9												3	1	2	2		
S																									

Table 6. 4 - Travel chart after economical duplication of machines is applied

Analysing the results from the utilisation comparison shows a visible improvement of 50% of the dynamic cell system organisation compared to the classical cell. In addition, it is seen that in the proposed research methodology for the design of the *DCM*, not all the available machines from the classical cell were utilised, which should give the potential for further increased utilisation.

Thus, the presented utilisation results justify the first three stages of the proposed research methodology. Additional case studies not recorded in this thesis also showed the same results. It is also recognised however, that graph theory and network analysis are still valuable tools for presentation and analysis of the material flows, especially for the directed tree's theory. However, after the first three stages of the *DCM* design, the next stage to consider is the economical optimisation of the *DCM* layout and machine placement on the available shop floor. Here the *SA* algorithm is utilised and is presented in the next section.

6.3. Stage IV Shopfloor Machine Placement - Layout Optimisation

The algorithm presented in Chapters 4 and 5, groups the production equipment of the manufacturing facility into dynamic cells and forms a set of part families, which are processed within these dynamic cells. It does not, however, address the relative placement of the machines within cells, or the relative placement of the cells and machines on the shop floor. Both these issues are critical in terms of the total material flow within the shop and are addressed in this section. To formulate the shop floor layout design problem, first of all a geometrical model of the shop floor and of the manufacturing resources is developed. Subsequently, decision variables are introduced to model a discrete choice in the construction of the flow network and the location of the resources on the shop floor. Finally, the assumptions guiding the design are stated and an integer programming formulation of the layout problem is developed.

Unfortunately, the layout design problems described in Chapter 4 and 5 do not exhibit optimal substructure; each step of layout design has been shown to be *NP* – complete. Furthermore, when a greedy algorithm search is applied, the strategy usually stagnates at a local minimum. In many cases there is a large disparity between the local minimum and the global minimum cost. That is why a search strategy, which avoids local minimum and finds the global minimum is needed, and simulated annealing is such a search strategy.

In the following sections the formulation for the discrete block layout problem, which is adequate for layout and material flow analysis, is now developed with shape constraints. The improved development now introduces definitions for restriction areas, swap, translation and rotation functions. Numerical experience with the optimal solution of the dynamic cellular layout formulation is reported, together with the results of a simulated annealing based heuristic (background of the *SA* is presented in the Appendix C) for the dynamic cell discrete block layout formulation. The proposed approach, based on the method of simulated annealing, is outlined in the next section.

6.3.1. Definition of the Dynamic Cell Discrete Layout Model

It has been noted before (section 2.4.2 background of the *SA* facility layout), that the facility layout problem in *CMS* has not captured researchers' attention as much as cell formation in the past two decades. A poorly designed layout will result in poor productivity, increased work in progress, disordered material handling, and so on. Only a few researches have dealt with this subject. The problem of shop floor layout can be considered analogous to that of chip placement on a microprocessor circuit board, since both involve the placement of manufacturing resources in a given two dimensional discrete space. This analogy has been exploited in the proposed research in order to generate efficient alternatives for the layout of a manufacturing shop. The machine and cell placement problem is similar, although they are addressed at different levels in the hierarchy of the manufacturing facility. Thus, they can be solved using an approach which is now described. Throughout the discussion, the term manufacturing resource (entities) represents either a machine within a cell, an entire cell within a shop, or an independent machine, depending on the context in which it is being used.

The layout problem entails the physical placement of manufacturing resources on the shop floor. The general model framework for the manufacturing shop, upon which this approach is based, considers an orthogonal unit imposed on its area as shown in *Figure 6.3.* This discrete model and solution algorithm approach identifies both the manufacturing resources to be located, and the shop floor enclosure in which they have to be placed, into a number of unit blocks. The blocks of a single resource are located closely on the shop floor and allow the size and location of the resources to vary continuously within the shop floor area. This approach must define in advance the acceptable geometrical shapes for the resources.

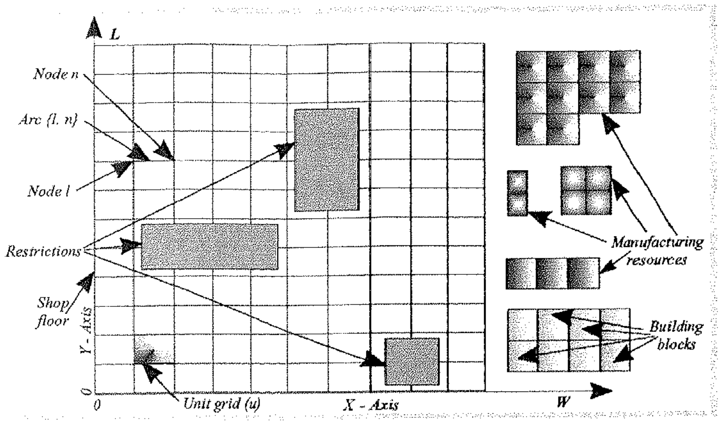


Figure 6. 3 - Representation of the shop and the manufacturing resources

6.3.1.1. Geometrical Attributes Representation

A square grid is imposed on the area available for the placement of the resources. The unit length of the grid is defined such that it is larger than the width of a typical aisle of the material flow handling system, and it is small enough to adequately capture the geometry of the manufacturing shop, the restricted areas, and the manufacturing resources.

The manufacturing resources which are going to be placed on the shop floor form the set *MR*. Each resource is decomposed into unit square building blocks, as shown in *Figure 6.4.* Although each building block is treated as a distinct entity, strong relationships are established between adjacent blocks of the same resources in order to retain the size of the resources in the final layout solution. For each block *k*, *[adj(k)],* the set of blocks that belong to the same resource is designated and are adjacent to (have a common edge with) *k* . The set of building blocks is denoted by *B*. The manufacturing resources may occupy one or more nodes of the grid depending on the resource size. Restricted areas are excluded from this grid as shown in the same *Figure 6.4* (no nodes are assigned to these areas).

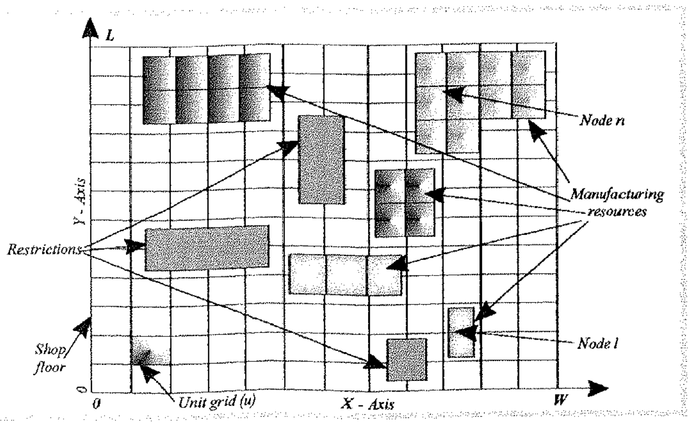


Figure 6. 4 - Representation of a feasible manufacturing resources shop layout

Each intersection of the grid represents a node of the underlying graph *G*. Graph nodes are candidate positions for the building blocks of the resources, and graph arcs are candidates of the material flow paths (aisles). The set of nodes that constitute graph *G* is denoted by G_N. The centroids of the building blocks of set *B* are located on grid points, as shown in *Figure 6.3.* It should be noted that the intersections of the shop grid, which are inside restricted areas occupied by manufacturing resources in certain layout configuration (ie. nodes *l* and *n* in *Figure 6.3*), are not considered as nodes available for material flow. Furthermore, the set of the graph's undirected arcs which connect the nodes available for material flow is denoted by *A*.

This discrete representation of the manufacturing shop and the manufacturing resources facilitates the introduction of decision variables to model the design problem. Furthermore, this representation also allows for a simply way to overcome area overlapping conflicts and complex flow path modelling.

6.3.1.2. Assumptions

Given the geometrical representations in the previous section 6.3.1.1, the development of the mathematical model is based on the following assumptions (made to simplify the analysis):

- The manufacturing resources (machines or cells) are represented by equidimensional square blocks to be placed in a finite, two-dimensional discrete space.
- The material flow paths are parallel to the building's walls (which is typical in manufacturing shops) and include only arcs of the grid imposed on the shop floor.
- The inter resources material flow rates per time period are constant and known. They are calculated from the production routing's (sequence of operations) of the products to be manufactured and the product demand for the planning period.
- The available shop area is predefined and is adequate to fit all manufacturing resources.
- The manufacturing resources of the dynamic cellular manufacturing system are enclosed by rectangular work areas.
- Restrictions on the shop floor are enclosed by rectangular areas.
- The distances D_{ij} between two manufacturing resources is defined as the rectilinear distance between their geometric centres.

The material flow between each resource pair $(i, j) \in MR^2$ is denoted by F_{ij}. The value of F_{ij} represents the volume of interactions between *i* and *j*, and is calculated from the part routing's demands over the planning period and the batch sizes. The levels of material flow are assembled in a square matrix of dimension equal to $|MR|$; this matrix is known as the material flow matrix. A cost C_{ij} is associated with each complete move $(i, j) \in C$, and then C_{ij} is defined as the travel cost from resource *i* to *j*, respectively.

6.3.1.3. Definition, Decision Variables and Parameters for DCM *Layout Design*

One of the main reasons for the limited success in the application of layout algorithms to real world facilities design projects has been the tendency of the computer-generated layouts to produce departments with unacceptable shapes. However, the proposed discrete modelling and design algorithms approach, is based on the fundamental notion that resources consists of a number of equal-sized squares, which are called unit squares. The area in which the resources are to be located consists of a number of equal-sized unit locations. The layout problem then consists of assigning the unit squares to the unit locations, whilst observing a number of layout constraints and objectives. Several decision variables are introduced which concern important attributes of the shop design to model the discrete choices. In the next section the formulations for the discrete block *DCM* layout problem with shape constraints are introduced and elaborated.

A) Parameters and Variables in the Layout Design

The problem of selecting the best unit square size is challenging in itself. If resources are represented by a large number of small unit squares, then the algorithms have substantial flexibility in determining the shape and location of these resources. This tends to reduce the distance rating for the objective function (the resources). On the other hand, a large number of unit squares per resource can generate complex resource shapes that are not acceptable in practice. In addition, the use of small unit squares decreases the probability that two resources will be located on the same number of unit squares. If resources have to have equal areas before they can be considered for a pairwise exchange, then using small unit squares decreases the number of possible exchanges. The parameters and variables definitions, which are used in this mathematical modelling, are illustrated in the block layout *Figure 6.5*.

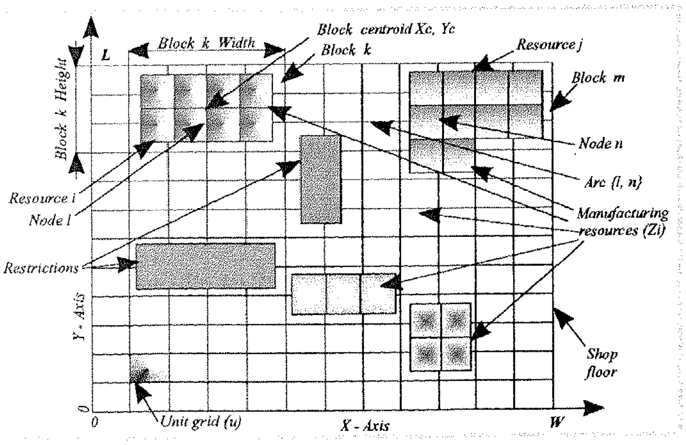


Figure 6. 5 - Block layout variables and parameters

The first of the decision variables models the assignment of blocks of resources to the nodes of *G*. A feasible assignment of the building blocks $k \in B$ to grid nodes $l \in N$ provides a feasible shop layout, in which the resources retain their size and shape. To model the assignment of block $k \in B$ to node $l \in G_N$, the binary variable is defined as follows:

$$h_{kl} = \begin{cases} 1 & \text{if building block } k \in B \text{ is located at node } l \in G_N \\ 0 & \text{otherwise} \end{cases} \qquad \textit{Equation 6. 6}$$

The values of these variables in the final solution provides the locations of the resource on the available area of the shop floor in the optimal (or near optimal) shop layout. The definition is closely related to the binary variables employed in quadratic assignment formulations [Willhelm 1987].

The second set of decision variables determines whether an arc *{l, n}* ∈ *A* is active or not in the flow network. The topology of this network depends on the arc costs, as well as on the arc capacities. The binary variable associated with each candidate undirected arc *{l, n}* ∈ *A*, is defined as follows:

$$y_{ln} = \begin{cases} 1 & \text{if arc } \{l, n\} \in A \text{ is active in the flow network,} \\ 0 & \text{otherwise.} \end{cases} \qquad \textit{Equation 6. 7}$$

Therefore, the values of arc related decision variables in the final solution define the optimal (or near optimal) material flow network. The cost F_{ij}, (material flow between resources *i* and *j*) related to each edge *{l, n}* ∈ *A* depends on the material flow handling cost. Finally, D_{ij} is the rectilinear distance between grid points *l* and *n* (the summation of the absolute values of the differences between the horizontal and vertical coordinates of the two point's *l* and *n*). This metric is known, and depends only on the structure of the grid imposed on the shop floor.

B) Layout Design Definitions

The first decision in the discrete assignment formulation is the determination of the unit square size. Based on the size of the unit square, the shop floor is divided into *R* rows by *NC* columns of unit locations, and each resource is divided into a number of unit squares. All areas are then normalised by dividing them by the unit square size, and all distances are normalised by dividing them by the length of the side of the unit square. The length and width of a manufacturing resource are defined as:

$$w_i = xr_i - xj_i \quad \textit{and} \quad l_i = yt_i - yb_i \quad . \qquad \textit{Equation 6. 8}$$

where:

xj_i - leftmost *x* coordinate of the smallest rectangle enclosing resource *i*,

xr_i - rightmost *x* coordinate of the smallest rectangle enclosing resource *i*,

yt_i - topmost *y* coordinate of the smallest rectangle enclosing resource *i*,

yb_i - bottommost *y* coordinate of the smallest rectangle enclosing resource *i*,

u - length of the side of the unit square,

w_i - width along the *x* axis of the smallest rectangle enclosing resource *i*, and

l_i - length along the *y* axis of the smallest rectangle enclosing resource *i*.

As previously referred to, one of the main reasons for the limited success in the application of layout algorithms to real world facilities design projects has been the tendency of the computer-generated layouts to contain departments with unacceptable shapes. Thus, a new formulation is proposed which attempts to avoid these unacceptable shapes by introducing additional shape constraints. Here the shape ratio of a resource is defined as the maximum of the length to width, or width to length ratios, of the smallest complete rectangle enclosing the resource, that is the shape ratio of resource *i* is defined by the following equation:

maximum value of the shape ratio of resource i, $S_i = max\left\{\frac{l_i}{w}, \frac{w_i}{l_i}\right\}$ *Equation 6. 9*

Basic definitions - Number of unit locations on the shop floor, indexed by *l*, is defined by *Equation 6.10* as the product of the number of rows along the y axis on the shop floor and the number of columns along the x axis on the shop floor. The number of unit squares for manufacturing resource i is defined by *Equation 6.11* and the total number of unit squares for all resources, indexed by *k*, is defined by *Equation 6.12*. All these definitions are as follows:

$$N = R \times NC \qquad \text{Equation 6. 10}$$

$$b_i = \mid I_i \mid \qquad \text{Equation 6. 11}$$

$$M = \sum_{i=1}^{Z} b_i \qquad \text{Equation 6. 12}$$

where:

R - number of rows along the y axis on the shop floor ,

NC - number of columns along the x axis on the shop floor ,

b_i - number of unit squares for resource i ,

I_i - set of unit squares belonging to resource i ,

M - total number of unit squares for all resources, indexed by k ,

Z - number of manufacturing resources, indexed by i , and

N - number of unit locations in the shop floor, indexed by l .

Feasibility Constraints - Based on the input parameters, the feasibility constraints (resource area and number of unit squares for the resource) can be immediately tested as follows:

- a_i - area of resource i $\sum_{i=1}^{Z} a_i \leq A$ (area of the shop floor), and
- b_i - number of unit squares for resource i $\sum_{i=1}^{Z} b_i \leq R \times NC$.

6.3.2. Mathematical Formulation of the Dynamic Cell Layout Approach

The main objective in this stage IV of the proposed research methodology, is to design a conceptual block layout for a number of manufacturing resources with unequal areas. All manufacturing resources have to fit inside the confines of a rectangular shop floor, and the manufacturing resources cannot overlap. In addition, the manufacturing resources must satisfy a shape ratio constraint. Furthermore, the manufacturing resources have material flow affinities. The objective function is to minimise the affinity-weighted centroid-to-centroid rectilinear material flow cost distance rating. The placement problem consists of determining the relative positions of the Z manufacturing resources in the set *{m_1, m_2,..., m_z}*, which might be either the set of machines belonging to a cell, or the set of manufacturing cells within the shop or machines. The criterion involved in the analysis is the minimisation of the total distance travelled by the parts between manufacturing resources. After this framework is given, the objective function *(TC)* of the layout problem is stated as follows:

$$Minimise \qquad TC = \sum_{i=1}^{Z-1} \sum_{j=i+1}^{Z} F_{ij} D_{ij} C_{ij} \qquad j \neq i; \ \{k,l\} \in A \qquad \textit{Equation 6. 13}$$

where:

TC – is the total material flow distance cost resulting from part transfers between resources within a specified production period plan (time horizon),

Z – is the number of manufacturing resources,

F_{ij} – is the material flow (volume flow units) traffic between manufacturing resources Z_i and Z_j,

C_{ij} – is the per unit traveling cost, and

D_{ij} – is the distance from manufacturing resource i to manufacturing resource j, and is computed from the manufacturing resource i to the manufacturing resource j in a manner that avoids passing through the other manufacturing resources and shop floor restrictions. Here the distance is measured rectilinearly between manufacturing resources centroids according to *Equation 6.14*. The coordinates of the centroid of each department are linked to the assignment constraints with the following constraints- *Equation 6.15 and 6.16*. These equations are as follows:

$$D_{ij} = | x_i - x_j | + | y_i - y_j | \qquad i, j = 1, \dots, Z; \ i \neq j \qquad \textit{Equation 6. 14}$$

$$x_i = \frac{\sum_{k \in I_i} \sum_{l=1}^{N} nc_l x_{kl}}{b_i} \qquad i = 1, \dots, Z \qquad \textit{Equation 6. 15}$$

$$y_i = \frac{\sum_{k \in I_i} \sum_{l=1}^{N} r_l x_{kl}}{b_i} \qquad i = 1, \dots, Z \qquad \textit{Equation 6. 16}$$

where:

nc_l - column index along the x axis of unit location k,

r_l - row index along the y axis of unit location k,

x_i - x coordinate of the centroid of resource i, *and*

y_i - y coordinate of the centroid of resource i.

Here (x_i, x_j) and (y_i, y_j) are the coordinates corresponding to the geometric centres of the manufacturing resources Z_i and Z_j respectively. Since the coordinate space considered is discrete and finite, each point in the space is assigned a unique position number in order to simplify the analysis. The problem thus consists of determining the position number corresponding to each of the Z manufacturing resources in order to minimise the objective function of *Equation 6.13*.

As was stated previously, it is assumed that each position on the grid can completely accommodate a manufacturing resource (manufacturing resources do not overlap when they are assigned to adjacent positions). Thus, the minimisation problem is subject to one simple set of the constraints, which states that each position can be occupied by not more than one manufacturing resource. The problem then is subject to the constraints detailed in the following sections:

6.3.2.1. Shop Floor Boundaries

Each machine must be fully located inside the associated boundary. For rectangular shop floors boundaries the shop floor area *(A)* is equal to the product of the shop floors width and length and defined by *Equation 6.17* as follows:

$$A = L \times W \qquad \textit{Equation 6. 17}$$

where:

W - width along the x axis of the shop floor, and

L - length along the y axis of the shop floor.

6.3.2.2. Area Overlap Constraints

This set of constraints ensures that one and only one resource block *(Figure 6.3)* is assigned to each node, *Equation 6.18*, and that each resource block is assigned to a grid node, *Equation 6.19*.

$$\sum_{i=1}^{Z}\sum_{k=1}^{M} h_{kl}^{i} \le 1 \qquad l = 1,\dots,N; \qquad i = 1,\dots,Z; \qquad \forall l \in G_N \qquad \textit{Equation 6. 18}$$

$$\sum_{i=1}^{Z}\sum_{l=1}^{N} h_{kl}^{i} = 1 \qquad k = 1,\dots,M; \qquad i = 1,\dots,Z; \qquad \forall k \in B \qquad \textit{Equation 6. 19}$$

$$\sum_{i=1}^{Z} h_{kl}^{i} \in \{0,1\} \qquad k = 1,\dots,M; \qquad l = 1,\dots,N; \qquad i = 1,\dots,Z \qquad \textit{Equation 6. 20}$$

$$\sum_{l \in G_N}\sum_{n \in G_N} h_{kl} h_{mn} d_{ij} = 1 \qquad \forall k,m \in B : k \in adj(m) \qquad \textit{Equation 6. 21}$$

$$h_{kl} + y_{mn} \le 1 \qquad \forall k \in B, \qquad \forall \{l,n\} \in A$$

$$max\left(\frac{(x_k - (x_l + w_l)) \ ((x_k + w_k) - x_l)}{(y_k - (y_l + l_l)) \ ((y_k + l_k) - x_l)} \right) \ge 0 \qquad \forall k \ne l \qquad \textit{Equation 6. 22}$$

$$h_{kl},\ y_{mn} \in \{0,\ 1\} \quad \forall (i,\ j) \in MR, \ \ \forall \{l,\ n\} \in A, \ \ \forall k \in B \qquad \textit{Equation 6. 23}$$

where:

$$h_{kl}^{i} = \begin{cases} 1 & \text{if block } k \text{ of resource } i \text{ is assigned to location (node) } l \in G_N \\ 0 & \text{otherwise} \end{cases} \qquad \textit{Equation 6. 24}$$

In *Equations 6.18* to *6.24*, G_N is the number of nodes of the grid, Z is the number of resources, and M is the number of building blocks required to satisfy the area requirements of manufacturing resource i. The assignment constraints expressed by *Equation 6.18 and 6.19*, are at most one resource block and should be assigned to every grid node. Adjacency constraints *(Equation 6.21)* force adjacent blocks of each resource to occupy adjacent grid points between the locations occupied by these blocks, and to be equal to the grid length *(=1 unit)*. Finally, expressions *Equation 6.20 and 6.23* ensures that the decision variables assume binary values.

6.3.2.3. Non Overlapping Condition

The two blocks h_k and h_l are not overlapping provided their respective x and y co-ordinates are themselves not overlapping. In other words, for two resources not to overlap, they must be separated either in the x dimension or in the y dimension, or both, defined by *Equation 6.22.* Two resources are separated in the x dimension if either resource i is to right of resource j, or resource j is to the right of resource i, and furthermore only one of these two conditions can be satisfied at any one time *(Equation 6.15, Equation 6.16 and Equation 6.21).*

6.3.2.4. Restrictions of the Location

Certain restrictions might exist for the physical location of a machine or cells. For example, consider a situation where a machine is already installed and its relocation would be economically unjustified. Subsequently however it was decided that the machine should retain its position in the new layout configuration. Thus if an area is desired for this machine, the grid blocks (of adequate size) will be restricted in the layout optimisation design. Location and rectangular size define these area restrictions, within *Equation 6.19* and *Equation 6.24.*

As previously noted, the layout problem is *NP* – complete, and thus a heuristic method has been developed to find a near optimal solution in a reasonable amount of time. The method employs simulated annealing (*SA*) in order to avoid local optima, and to provide several alternate solutions from the same initial configuration.

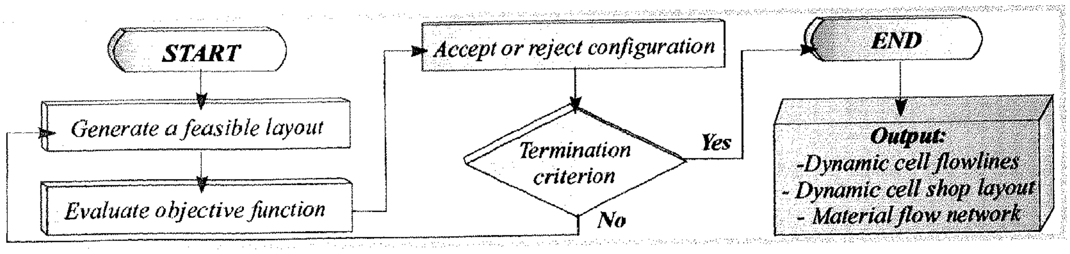

START
Generate a feasible layout
Evaluate objective function
Accept or reject configuration
Termination criterion
Yes
No
END
Output:
-Dynamic cell flowlines
- Dynamic cell shop layout
- Material flow network

Figure 6. 6 - Flow chart of the simulated annealing based dynamic cell layout methodology

6.3.3. Proposed Solution Using Improved Simulated Annealing

Figure 6.6 illustrates the basic steps of the Metropolis [1953] simulated annealing based method which is enhanced by an improved *SA* approach *(Figure 6.13)* specifically developed for the solution of *DCM* layouts. The proposed *SA* based layout method minimises the objective function of *Equation 6.13*, which models the assignment of the manufacturing resources to the nodes of the grid (decision variables h_{kl}, $k \in B$, $l \in G_N$). The algorithm described in this section provides feasible values to these variables, evaluates the resulting feasible layouts, and repeats this process until a near optimal shop layout is reached. This algorithm comprises two basic loops, a global loop that decreases the temperature according to the cooling schedule and an internal loop with each temperature. The essential components of the proposed method are:

- a systematic generator of alternative feasible shop layouts, and
- an annealing schedule that drives the system configuration to near optimal solutions (from larger to small temperatures).

Constraints *(Equation 6.18 – 6.23)* are mathematical relationships that express the assignment of each resource building block to one grid node. The initial distance between all pairs of manufacturing resources is determined within stages II and III of the proposed research methodology (utilising the *MWDRST* algorithm from the Chapter 4). Also, it is noted from the Appendix C sections that four elements are essential for the Metropolis algorithm [1953] described by Press [1990]. These elements are as follows:

- description of possible system configurations,
- generator of random changes in the system configuration,
- an objective function *TC* (analogous to energy), the minimisation of which is the goal of the procedure, and
- a control parameter *T* (analogous to initial temperature) and annealing schedule, which dictates how *T* is decreased (how many random configuration changes in each downward step in *T* is taken and how large is that step). The initial value of *T* and the determination of the annealing schedule, which may require physical insight and/or trial and error experiments.

The Metropolis algorithm searchers for an optimum solution (minimum energy) by modifying the system configuration. In *SA*, the system will change from a configuration with energy E_1 with probability $p = e^{-\Delta E/kT}$ for $E_2 > E_1$, or $p = 1$ for $E_1 \geq E_2$, where k is a constant which relates temperature to energy (Boltzman's constant) and T is the temperature of the system. If $E_1 \geq E_2$, the change is always accepted. Otherwise there is non-zero probability of accepting the uphill solution. In this implementation of simulated annealing, each resource is considered as a single entity, and based on its location on the shop floor. Thus, the assignment of building blocks to grid nodes is determined.

Problem specific	***Generic***
Solution space (Configuration)	Initial temperature
Configuration changes (Neighbouring)	Annealing (Cooling)schedule
Objective function (Configuration cost)	Length of the search (Epoch)
Initial feasible configuration (Placement)	Stopping criterion (Frozen system)

Table 6. 5 - Parameters used in simulated annealing

It may be noted that the choices the designer of a simulated annealing algorithm has to make can be classified into two classes, namely problem specific and generic. The terms that must be specified are shown in *Table 6.5*. Simulated annealing can be modelled as an algorithm which, given a neighbourhood structure, constantly attempts to transform the current configuration into one of its neighbours. As mentioned before this is an approach for solving combinatorial optimisation problems, rather than a specific algorithm. This is a direct result of the fact that the algorithm designer must specify many of the terms which appear in the following sections, ie. inputs.

6.3.3.1. Inputs

The proposed solution for determining the shop floor layout requires the following input information:

- The set of manufacturing resources, and thereby for each resource.
- The size of its rectangular envelope and its moveability status, and data, are provided.
- The input parameters required for the *SA* algorithm.
- The set of area restrictions, defined by location and rectangular size.
- The geometry of the area that is available for the placement of manufacturing resources. (Also, note that the dimensions of the shop, the manufacturing resources, and restricted areas, should be given in the same units.)
- The minimum width of material handling corridors.
- The set of parts (demands, average pallet size, average batch size and part description).
- The traffic between manufacturing resources in terms of the material flow. This information can be provided either directly, or indirectly, by specifying the unique process plans of all the parts produced in the system, their production volume over a certain production period, and the number of units of each part that can be accommodated on a standard pallet.
- In the case of manufacturing resources, the relative sizes and dimensions of the manufacturing resources are also required in terms of the number of square building blocks required for representing each manufacturing resource.

As discussed before, a square grid covers the shop area available for the placement of manufacturing resources, excluding the restrictions. The resolution of this grid is so defined that the basic system entities (resources, material handling corridors and details of the shop area) can be described adequately. It is also noted that in order to accommodate material handling corridors between resources, both length and width of each manufacturing resource are incremented by the corridor width. The definition of all parameters, which is needed for an adequate problem set up, within the basic elements of the Metropolis algorithm, is described in the following sections.

6.3.3.2. Problem Set Up

A) Solution Space (Configuration)

The solution space or configuration, is the first problem specific term that appears in *Table 6.5* for the *SA* based *DCM* layout. The solution space (configuration) is given by assignment of the manufacturing resources to the candidate locations. Here *Z*, the number of manufacturing resources under consideration, have to be placed on a $(R \times C)$ grid, where each point is assigned a unique identification number. It is noted that only *Z* of the $R \times C$ positions on the grid would be actually occupied throughout the solution process. The reason for considering such a large solution space is to accommodate even the extreme case of a linear facility placement.

B) Description of System Configuration Changes (Neighbourhood)

The second problem specific term from *Table 6.5* is the configuration changes (neighbourhood of a configuration), which consist of those configurations that result from the interchange of the manufacturing resource locations. Here, for a given initial (feasible) layout configuration, new system layout configurations are generated by applying the following operations:

- Translate - A move of a manufacturing resource from its current position to another, previously unoccupied position.
- Swap - A swap of the positions of any two manufacturing resources.
- Rotate – Rotates manufacturing resources by 0^0, 90^0, 180^0 or 270^0.

All these actions involve the exchange of the contents of two positions, at least one of which contains a manufacturing resource. Exchanges involving more than two positions can also be accommodated, although this increases the complexity of the method and does not offer substantial advantages. A random number generator is utilised at each iteration to select the type of operation to perform swap, translate or rotate.

Swap

Two resources are selected at random, and their positions are interchanged. The algorithm overcomes the problems imposed by differences in the size of the resources as follows:

- Given two resources selected for swapping, the algorithm identifies the larger one (ie. the one comprising more building blocks), and swaps it with a set of resources adjacent (and including) to the second one. The overall size of the resources in this set is almost equal to the size of the larger one.

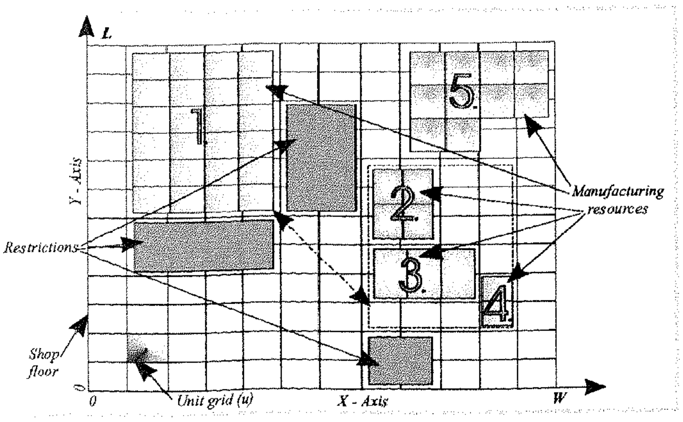


Figure 6. 7 - Swap operation of manufacturing resources

Figure 6.7 illustrates this layout transformation. Resources 1 and 2 have been selected for a swap operation. However, resource 1 includes many more building blocks than resource 2. Thus, the algorithm identifies the resources (2, 3 and 4) with resource 1. This identification is accomplished by checking all resources surrounding 2 and selecting those that, when added to 2, form a set of building blocks that is similar to 1 (ie. resource 5 in *Figure 6.7* is such a candidate that is disregarded since when it is added to resource 2 it forms a bigger set of building blocks that resource 1).

Rotate

One resource is selected at random and it is rotated counterclockwise around it's centre of gravity by 90^0, 180^0 or 270^0. The rotation angle is selected at random. However, as in the case of translation, all three angles of rotation are examined for feasibility regarding the resulting configuration. *Figure 6.8* illustrates the possible resource rotations.

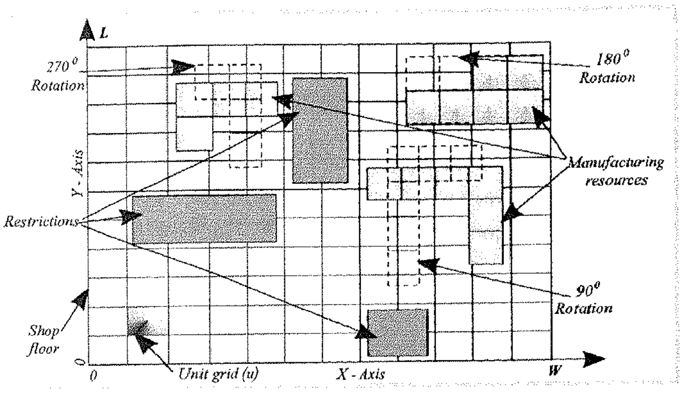


Figure 6. 8 - Rotation of manufacturing resources

Translate

One resource is selected at random and is translated by one grid unit in one of four directions:

Up, down, left or right.

The direction is selected at random. However, if it is not feasible to perform the transformation along the selected direction, all remaining directions are examined and any feasible one is selected. *Figure 6.9* illustrates one possible translations per resource. It is important to emphasize that several of the attempted configuration changes may not be feasible because of resource overlapping constraints (*Equation 6.18 - 6.23*).

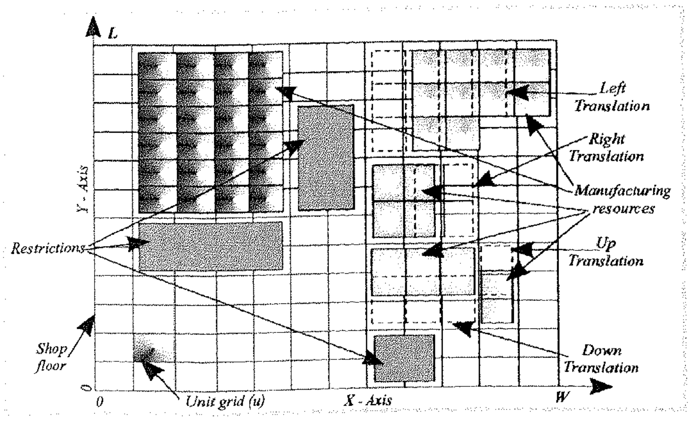


Figure 6. 9 - Translation of manufacturing resources

Since *SA* is most effective when small configuration changes are performed, the probability of choosing between these operations is not uniform. Swap alters the energy of the system the most, and thus, it has the lowest probability of being selected *(0.3)* (only *30%* of the overall configuration changes consists of swaps of resources). The other two remaining operations, translate and rotate have a smaller (minimal changes) impact on the system's energy, and thus are selected with a probability of *0.1* each. These probabilities of the configuration changes will be justified in this Chapter: Experimental Study Section 6.3.7. (Analysis of Parameters).

C) Objective Function, Total (Configuration) Cost TC (Analogous to Energy E)

The cost of a configuration, or objective function, is the total material flow handling cost, which is the third problem specific parameter of the simulated annealing algorithm. In this application of simulated annealing, the objective function to be minimised has been defined by *Equation 6.13.* The following parameters are related to the objective function, described as follows:

- TC_c is the value of the objective function for an accepted configuration. TC_c is related to the energy function in the *SA* process.
- TC_n is the objective function of a configuration before the decision of acceptance or rejection of this configuration has been made.
- TC_0 is the value of the objective function for the best configuration generated by the algorithm.

Here, minimisation of the objective function (configuration cost) is done by utilising simulated annealing. If the cost ever reaches zero, a dynamic cell layout design has been found and the iteration can be ended.

D) Initial (Feasible) Configuration

The initial feasible configuration is the fourth problem specific parameter of the simulated annealing algorithm, which assigns manufacturing entities to locations. The procedure, which is used to construct a initial feasible assignment, is described in detail in the section *step 3* of this chapter. There are two entities which should be placed on the grid: manufacturing resources and restrictions. All manufacturing resources and restrictions are randomly placed within the solution space (grid) of the shop area grid nodes, without violating the related constraints of the *DCM* layout mathematical model, described earlier in this chapter *(Equation 6.18 – 6.23).*

This initial placement configuration is done with the help of a random number generator, which assigns each manufacturing resource to an empty position. Since restrictions are areas in which no resources may be placed, the grid is redefined not to include the nodes corresponding to these restrictions. Immovable resources are first placed onto the grid at user specified locations, and are immovable throughout the layout analysis. The remainder of the resources are placed randomly onto the available area.

E) Control Parameter, T (Analogous To Initial Temperature)

In the physical analogy the initial temperature should be large enough to heat up the solid until all particles are randomly arranged in the liquid phase. This means that at the beginning of the annealing process, the transitions are able to reach all of the configurations. By this property, the algorithm can end by producing a solution that does not strongly depend upon the initial configuration. The initial temperature *T*, is a first generic parameter *(Table 6.6)* which controls the highest point of the transition probability. A smaller *T* value leads to a smoother search process. Again at the limit point, $T = 0$, the process becomes deterministic.

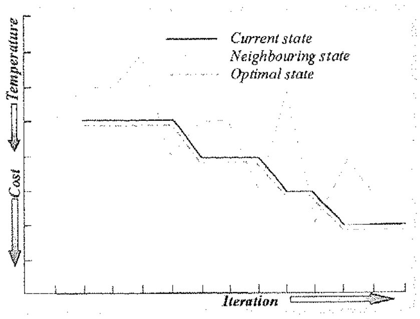


Figure 6. 10 - Effect of low initial temperature T

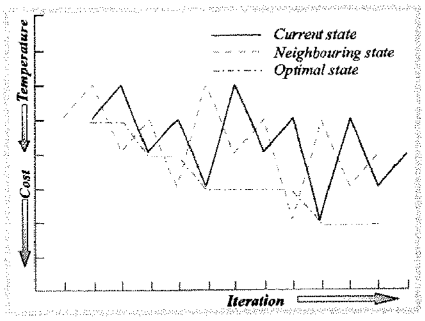


Figure 6. 11 - Effect of high initial temperature T

Computational experience shows that *T* is an important parameter and should be selected carefully. Thus, if a very low *T* is selected, the annealing schedule will quickly reach the freezing point and, consequently, the stochastic search process becomes deterministic. Once it becomes deterministic, the current state will never transit to a higher cost *(Figure 6.10)* and thus there will be no chance to escape a local minimum. On the other hand, if *T* is too high, the chance of transition from a current solution to a worse solution will increase considerably. Thus the current state will simply follow the neighbouring state pattern *(Figure 6.11)* and hence the search process deteriorates into an unguided random process, although this process may not last too long. Therefore, a good initial temperature should be selected in such a way that the current state will have a better chance of transition to a lower cost *(Figure 6.12)*.

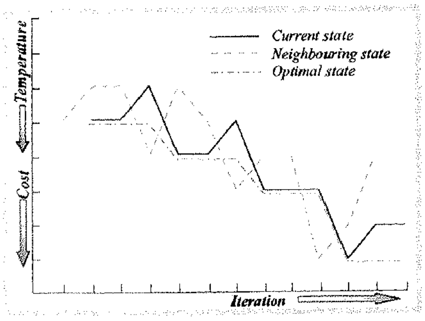


Figure 6. 12 - A good initial temperature T

Following the principle of simulated annealing, a proper initial temperature is the one that yields a probability of close to *1* for the first iteration. Thus, a more robust guideline for selecting the initial temperature *T* is now suggested. The initial temperature *T* can be determined by means of a cost increasing transition, which would be accepted in the beginning of the annealing process with probability P_0. The mean cost increasing $\mu\Delta$, of the cost-increasing transitions, is then computed. In the calculation, *T* is defined as follows:

$$T = \frac{\mu\Delta TC}{ln(P_0)}$$ *Equation 6. 25*

where:

T – initial temperature,

$\mu\Delta TC$ – the mean cost increasing, of the cost increasing transition,

P_0 – acceptance probability of the annealing process in the beginning *($P_0 = 0.8$)*, and

ΔTC – the number of transitions attempted for calculating ΔTC, as a fraction of the total number of the neighbouring configurations, where the total number (size) of the neighbouring configuration is of the order of Z^2.

It may be noted that the user specifies the control parameter *T (Equation 6.25)*. A high value of *T* will result in an initial random search of the configuration space, followed by a more systematic search, as *T* assumes lower values. A good initial value for *T* is given by the maximum possible change in the objective function [Aarts 1989] and, in addition, a proposed acceptance probability *($P_0 = 0.8$)* at the beginning of the annealing process. This will result in the initial acceptance of some random changes. However solutions of lower energy will be primarily accepted. The change of temperature is dictated by the annealing schedule, which is also specified by the user. The initial temperature should be sufficiently large so that virtually all configuration changes are accepted by the criterion presented in the next section.

F) Annealing (Cooling) Schedule

The annealing (cooling) schedule, the second generic parameter of the *SA* algorithm, consists of defining the initial (melting) temperature *T* and the equation which governs the change in the temperature of the system at each step of the procedure. This annealing schedule is a constant by which the temperature is reduced at each temperature iteration. Since the temperature reduces exponentially with the annealing schedule, its value drops quickly. The temperature change schedule, in its simplest form, can be a constant positive multiplier with values less than unity. For a certain value of temperature, the temperature is reduced when the number of transitions reach the upper bound of the Markov (presented in more detailed within Appendix C) chain length. The control parameter, ie. the reduction ratio of temperature, usually is chosen for small temperature changes. The Markov chain more easily leads to an equilibrium state if the temperature change is small. Hence, in this algorithm the decrement rule is defined as follows:

$$T_i = T_{FC} \times T_{i-1} \qquad \textit{Equation 6. 26}$$

$$T_{FC} = \frac{T}{1 + \log i} \qquad \textit{Equation 6. 27}$$

where:

T_i - is the temperature for the new iteration,

T_{i-1} - is the temperature at the current iteration,

T_{FC} - is the annealing schedule temperature decrement factor defined as a logarithmic function, and

i - is the number of the annealing iterations.

The control parameter T_{FC} is chosen small, but close to *1*. Typical values for T_{FC} lie *between 0.85 – 0.95*. An annealing schedule value of *0.9* allows for moderate changes in the objective function under which *SA* works best [Aarts 1989]. This value of *0.9* is used in this *SA* algorithm program as the default setup.

G) Epoch – The Length of the Search (Markov Chain)

The simulated annealing process transfers the current configuration to its neighbours with a certain probability, this modelling is equivalent to a Markov chain (also known as Epoch – the length of the search). Thus, it is considered that each value of temperature is modelled as a Markov chain. In fact, the number of accepted transitions are smaller with lower values of the temperature. The length of each Markov chain, referred to here as the epoch length *L*, is the third generic parameter used in *SA* from *Table 6.5*. The epoch length is taken to be equal to a percentage of the total neighbourhood size (Z^2), and should be as large as possible. It should not be significantly dependent on the temperature, but must take into account the fact that, at low temperatures, the number of moves actually accepted is rather small to find an upper bound for the epoch length. Otherwise, unnecessarily long Markov chains would exist in the final stages of the annealing schedules.

The length of the search (Epoch – the number of trials to be performed with the same temperature value), together with the annealing scheduling, is employed to decrease the temperature after each cooling stage is completed. Prior to the description of the algorithm, some notation relevant to simulated annealing is presented as follows:

- Let N_l be the total number of temperature changes allowed by the *SA* procedure, and n_i the number of those iterations at any stage of the algorithm.
- Let N_T be the number of configuration changes allowed within each temperature (internal loop), and n_t the cumulative number of these iterations at any stage of an internal loop (n_t is initialised to *0* when the temperature is decreased, and increases by one for each configuration tested within this temperature).
- Let N_s be the temperature of accepted configuration changes allowed within each temperature *T*, and n_s the number of accepted configurations at any stage of internal iterations (n_s is initialised to *0* when the temperature is decreased, and increases by *1* for each configuration accepted within this temperature).

The length of the epoch is determined by the above three parameters with the capital subscript letter, and in this application of the *SA* the epoch length is controlled by (two parameters) the maximal allowable number of times the new solution is accepted and rejected, per epoch. These values are computed for a constant value temperature, and if one of these numbers, or both, become equal to their maximal allowable value, the equilibrium state is established and the temperature is reduced in order to start a new epoch. The default values of the above parameters utilised in this algorithm are as follows:

$N_l = 100$ - the total number of temperature changes allowed by the *SA* procedure,

$N_T = 100 \times Z$ - the number of configuration changes allowed within each temperature,

$N_s = 10 \times Z$ - the temperature of accepted configuration changes allowed within each temperature *T*, and

$L = 20 \times Z/(T + 2)$ - the length of the epoch (length of the search) *Equation 6. 28*

Here *Z* is the total number of the manufacturing resources, *T* is the initial temperature of the *SA*, and the epoch length *(L)* is defined as a geometric function of *Z* and *T* as shown in *Equation 6.28.* It should be noted that all these previously-mentioned parameters can also be re-specified by the program user.

H) Stopping Criterion

The fourth generic parameter used in the simulated annealing algorithm, which must be addressed, is the stopping criterion. The annealing process is terminated when the system is frozen, ie. the value of the cost function of the solution does not improve after a certain number of consecutive Markov chains. In addition, and obviously, the execution of the program-algorithm can be discontinued if the expected improvement in the configuration cost is rather small. In this proposed version of *SA*, the annealing process is terminated if either of the following two conditions are satisfied:

1. The process will stop if the number of accepted transitions is less than a given fraction of the total number of attempted transitions (or below a certain point).
2. The annealing process will also stop if the current best configuration remains unchanged for a number of temperature reduction steps.

Aarts and Korst [1989 and 1996] have proved that the upper bound of the total number of temperature reduction steps (ie. the number of Markov chains) is proportional to the solution space that denotes the finite set of all possible solutions. In the layout problem solved here, the solution space is equivalent to the factorial of *Z* (number of manufacturing resources). Most of the elements in the solution space, however, are infeasible solutions because there are too many constraints, so a number of reduction steps are used as the upper bound for the number of Markov chains. In addition, it should be noted that the program default stopping criterion is when a given final temperature of $T_f = T / 800$ is reached, and the user can then redefine the final temperature.

I) Configuration Change Acceptance Criterion

The Metropolis criterion [Metropolis 1953] was selected to govern the acceptance or rejection of configuration changes. It considers the following cases:

- If the configuration change results in a net reduction in the objective function of *Equation 6.13*, then it is accepted.
- If the configuration change increases the objective function, then it is accepted with probability of $e^{-\Delta TC/T}$, where ΔTC represents the change in the value of the objective function and *T* is the temperature of the system at the time the configuration change is attempted. Thus, the configuration change is accepted if a randomly generated number between *0* and *1* is less than

the value $e^{-\Delta TC/T}$.

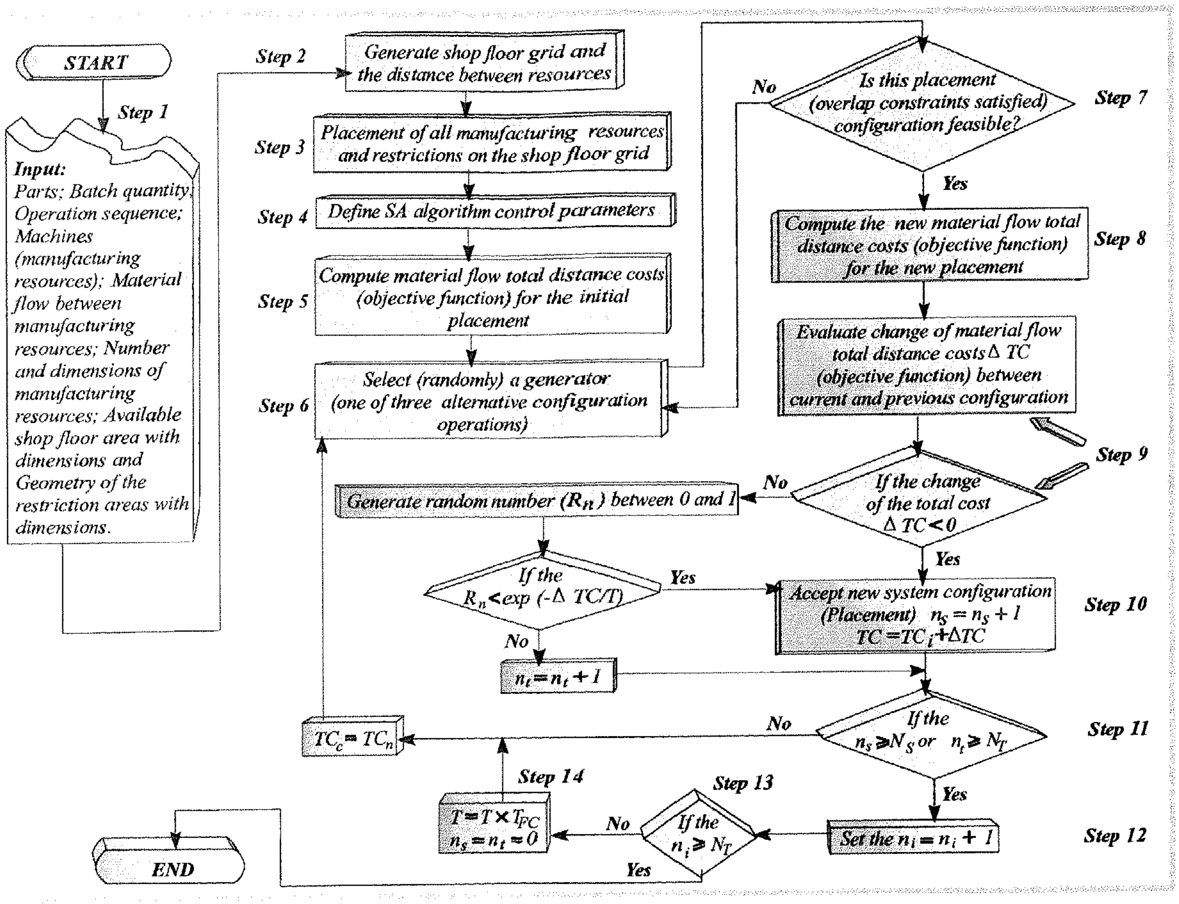

START
Step 1
Input:
Parts; Batch quantity; Operation sequence; Machines (manufacturing resources); Material flow between manufacturing resources; Number and dimensions of manufacturing resources; Available shop floor area with dimensions and Geometry of the restriction areas with dimensions.
Step 2
Generate shop floor grid and the distance between resources
Step 3
Placement of all manufacturing resources and restrictions on the shop floor grid
Step 4
Define SA algorithm control parameters
Step 5
Compute material flow total distance costs (objective function) for the initial placement
Step 6
Select (randomly) a generator (one of three alternative configuration operations)
No
Is this placement (overlap constraints satisfied) configuration feasible?
Step 7
Yes
Compute the new material flow total distance costs (objective function) for the new placement
Step 8
Evaluate change of material flow total distance costs Δ TC (objective function) between current and previous configuration
Step 9
No
If the change of the total cost Δ TC<0
Generate random number (R_n) between 0 and 1
Yes
If the R_n<exp (-Δ TC/T)
Yes
No
Accept new system configuration (Placement) $n_S = n_S + 1$ $TC = TC_i + \Delta TC$
Step 10
$n_t = n_t + 1$
No
If the $n_s \geq N_S$ or $n_t \geq N_T$
Step 11
$TC_c = TC_n$
Step 14
Step 13
Yes
$T = T \times T_{FC}$ $n_s = n_t = 0$
No
If the $n_i \geq N_T$
Set the $n_i = n_i + 1$
Step 12
END
Yes

Figure 6. 13 - Flow chart for the SA based layout of DCM shop floor

6.3.4. The *SA* Layout Algorithm

Once the parameters of the method are completely defined, the method of simulated annealing can be applied to generate efficient layout alternatives for the manufacturing system. The flow chart of the proposed algorithm is shown in *Figure 6.13.* A detailed description of its various steps is given in the next section. Solutions to the Nugent et al and Bazaraa problems are shown in *Figure 6.15.*

Step 1 - Data Processing

The number of manufacturing resources to be allocated and the material flow between them are specified, either directly or indirectly as mentioned in section 6.3.3.2.

Step 2 - Shop Floor Grid Definition

A ($R \times C$) grid is generated for the placement of the Z manufacturing resources, and the distance between all pairs of resource positions is determined using the rectilinear distance criterion.

Step 3 - Initial Configuration

This *Step 3* is actually the fourth problem specific parameter of the *SA* algorithm. The following procedure is used to construct an initial feasible assignment of resources to grid nodes. One manufacturing resource is randomly placed within the solution space (grid) of the shop area grid nodes, in such a manner that the block interrelation constraints *(Equation 6.23)* are satisfied. The procedure continues with a new manufacturing resource, until either all resources are placed on the shop floor, or one or more of them cannot be placed on the grid without violating the related constraints *(Equation 6.18 – 6.23)*. This is done with the help of a random number generator, which assigns each manufacturing resource to an empty position. The user overrides this default option to specify the initial placement of the manufacturing resources. There are two entities which should be placed on the grid: manufacturing resources and restrictions.

Since restrictions are areas in which no resources may be placed, the grid is redefined not to include the nodes corresponding to the restrictions. Immovable resources are first placed into the grid at user specified locations, and are immovable throughout the layout analysis. The remainder of the resources are placed randomly into the available area. If the placement of the resources is feasible, the algorithm proceeds to *Step 2*, otherwise this configuration is disregarded and *Step 1* is repeated, starting from a different resource.

Step 4 - SA *Control Parameters*

This *Step 4* defines the *SA* annealing control parameters, which are listed in *Table 6.6*. The annealing schedule is set up in this step. The initial temperature is determined by identifying the lowest temperature at which at least 80% of a certain number of random configuration changes would be accepted. The user can also directly specify the initial temperature. The following parameters, such as: the temperature reduction factor T_{FC} *(<1,0)*, the number of configuration changes N_I to be attempted at each temperature, the number of successful configuration changes allowed N_S at each step, and the number of temperature steps in the annealing procedure N_T, are also specified.

Parameters	***Definition***
T	Initial temperature
T_{FC}	Annealing schedule factor *(0.0 – 1.0)*
N_I	Total number of temperature changes
N_T	Total number of iterations at each temperature
N_S	Number of successes at each temperature before continuing to the next temperature

Table 6. 6 - Definition of the SA variable - control parameters

Step 5 - Evaluating the Objective Function

This step evaluates the objective function of the shop design problem for a given assignment of blocks to locations that satisfies: the area overlapping constraints, respects the geometry of the shop, and the restricted areas *(Equation. 6.18 – 6.23)*. The objective function, the total material flow handling distance *TC* between the manufacturing resources, is calculated for the initial placement using *Equation 6.13*. The distance D_{ij}, between all pairs of manufacturing resources *i*, is calculated as before for stages II and III (Chapter 4 and 5) utilising the *MWDRST* algorithm, this also determines the shortest path between these entities. This path may not pass through other entities and restrictions. The material flow traffic F_{ij} is computed from the material flow handling traffic between manufacturing resources *i* and *j*. The initial placement is stored as the best layout and the corresponding value of *TC* is stored as the minimum material flow distance.

If any of the constraints *(Equation 6.18 – 6.23)* are violated, the configuration is rejected, and the objective function value is set equal to ∞ . In this case, the algorithm returns to *Step 3* to determine a new start configuration. Otherwise, TC_c is given the value of the objective function for the current layout, setting $TC_0 \leftarrow TC_c$, and the various *SA* control parameters initialised (ie. $n_t = 0$, $n_i = 0$ and $n_s = 0$).

Step 6 - Generating a New Layout

Given a feasible shop design layout, the random number generator (transformations) is utilised to select the type of configuration change (swap, translate or rotate) to generate a new layout. Thus:

> Two positions from the solution space, at least one which contains a manufacturing resource, are randomly selected using the random number generator. The attempted configuration changes consist of swapping, translating or rotating the contents of these two positions (as explained using the defined values within the Problem Set Up section 6.3.3.2) ie. the probability of a swap is *0.3,* whilst the probabilities of a translate or a rotate are each *0.10.*

Step 7 - Checking for Overlapping

If the solution does not satisfy the area overlap constraint *(Equation 6.18 – 6.23),* then a new system is configured, *(step 6),* otherwise continue to *step 8.*

Step 8 - Evaluating the New System Objective Function

For the new system configuration, the new objective function value TC_c is computed as per *step 5* for the shop layout design problem according to:

- If any of the constraints of the mathematical model (section 6.3.4) are violated, then the objective function value TC_n is set to ∞.
- If no constraints of the mathematical model (section 6.3.4) are violated, then TC_c is calculated (the value of the objective function) for the current layout.

Step 9 - Accepting or Rejecting a Configuration

The usual Metropolis criterion is utilised for accepting or rejecting configurations. The change in total cost (objective function) ΔTC (as the change in the material flow handling distance between the current and previous configurations) is evaluated first, as follows:

$$\Delta TC = TC_c - TC_n \qquad \textit{Equation 6. 29}$$

Here TC_n is the value of the objective function for the previous accepted configuration and TC_c is the value of the objective function for the layout evaluated in *step 8.* In order to determine whether the new configuration is acceptable, a random number R_n $(R_n \in [0,1])$ is selected and compared to the acceptance (probability) criterion $(e^{-\Delta TC/T})$.

Thus, there exists two possibilities for ΔTC:

1. If the improved configuration results in $\Delta TC \leq 0$ (net reduction in the objective function

Equation 6.6) the configuration is always accepted, since the acceptance criterion $e^{-\Delta TC/T} > 1$. Thus, because the objective function is improved, and two parameters updates are performed:

- $TC_c \leftarrow TC_n$,
- If $TC_n < TC_0$, than $TC_0 \leftarrow TC_n$.

Note that TC_0 is the value of the objective function for the best (up to this point) configuration.

2. If the configuration change of energy TC_c is greater than TC_n, resulting in $\Delta TC > 0$ (net increase in the objective function) the configuration is accepted with probability of:

$$\eta = e^{-\Delta TC/T} \qquad \textit{Equation 6. 30}$$

where ΔTC represents the change in the value of the objective function, e is the exponential factor, and T is the temperature of the system at the time the configuration change is attempted. The acceptance or rejection is accomplished by comparing a randomly generated number $(R_n \in [0,1])$ with the value of $\eta = e^{-\Delta TC/T}$:

- If $\eta > R_n$, then the new configuration is accepted, and $TC_c \leftarrow TC_n$.
- If $\eta \leq R_n$, then the new configuration is rejected.

At this point the number of overall iterations at the current temperature T is increased by one $(n_t \leftarrow n_t + 1)$. If the new configuration is accepted, the number of successful changes in temperature T is also increased by one $(n_s \leftarrow n_s + 1)$. Thus, the configuration change is accepted if a randomly generated number $(R_n \in [0,1])$ between 0 and 1 is less than the value $\eta = e^{-\Delta TC/T}$. The probability of acceptance is progressively lowered as the temperature decreases. If the solution is not accepted, than the counter number of iterations is incremented at a temperature and continued to *Step 11* - otherwise it is continued to the next step.

If the configuration change attempt is successful, the composition of the two candidate positions is reconfigurated and the value of the objective function is updated. It should be noted that as a result of the configuration change the objective function might increase, decrease, or remain stationary. At lower temperatures, however, the probability of an increase in the objective function is significantly lower. If the material flow handling distance for the resulting layout is less that of the best layout stored in memory, the former replaces the best layout, and the corresponding material flow handling distance replaces the minimum distance (stored in memory).

Step 10 – Incrementing the Counters

If the solution is accepted, then $TC_c = TC_n$ is set and the counters incremented such that:

- n_t = number of iterations at a temperature, and
- n_S = the number of successes at a temperature.

Step 11 – Checking Internal Iterations

If the number of internal iterations at the current temperature has not exceeded the maximum number of iterations at the preset upper bound (ie. $n_t < N_T$ and $n_s < N_S$) temperatures, or the number of successfull (or accepted) iterations at a temperature has not exceeded the maximum number of successfull iterations at that temperature, then the iterations are continued to the next system configuration. At this stage a new internal iteration is executed for the current temperature by returning to *Step 6*. Otherwise, the procedure is continued to the next temperature, *Step 12*.

Step 12 – Updating SA *Parameters*

The number of temperature iterations is then incremented and a set of a number of iterations at the new temperature is begun, with updating of the following parameters:

- Internal iteration counters are then reset (ie. $n_t \leftarrow 0$).
- Temperature iterations are then increased by *1* (ie. $n_i \leftarrow n_i + 1$).

Step 13 – Checking Temperature Iterations

If the number of temperature iterations exceeds the maximum number of temperature (changes) iterations ($n_i \geq N_I$), then the execution of the program (algorithm) is halted. Otherwise ($n_i < N_I$), when the number of allowed temperature changes has not been reached, the algorithm is continued to *Step 14*.

Step 14 – Completing Predefined Temperature Steps

The annealing temperature is decreased by the temperature reduction factor T_{FC}. This provides a lower probability at which an uphill solution is accepted. *Steps 3* through to *14* are repeated until the predefined number of temperature steps is completed (or equals the number of steps). The annealing is also stopped if the number of successful configuration changes at any temperature, equals zero. The value of the objective function for the final layout is compared with that of the layout stored in memory, and the better one is saved. Thus, the objective function value for the best configuration is TC_0. This configuration comprises the layout of the manufacturing resources on the shop floor. Thus, a complete shop floor design is derived at the termination of the simulated annealing based algorithm.

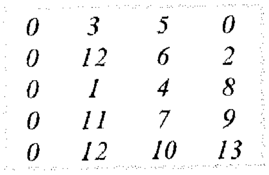
The output information, obtained from the application of the layout algorithm to the dynamic cell manufacturing system, is as follows:

- Optimal or near optimal layout of the manufacturing resources (facility).
- Flow paths between all pairs of manufacturing resources for the optimal layout solution.
- Final objective value, which quantifies the total material flow costs distances within the manufacturing shop (system).

6.3.5. Numerical Implementation

The shop layout displacement of the dynamic cellular manufacturing system, based on the simulated annealing algorithm was implemented using *MATLAB*, the main programming tool in this research. All the mathematical computing was done using *MATLAB*. This included coding and building of the algorithms, which provided an environment to develop, run and modify the algorithm models. The *MATLAB* program, which is a powerful matrix generator, provides all the features required to generate a model in a efficient, intuitive manner. Output from the *DCM* shop displacement *MATLAB* program is given in two styles:

- matrix output (example presented in the *Figure 6.14),* and
- graphical interface presentation as shown in the *Figures 6.15 - 18.*



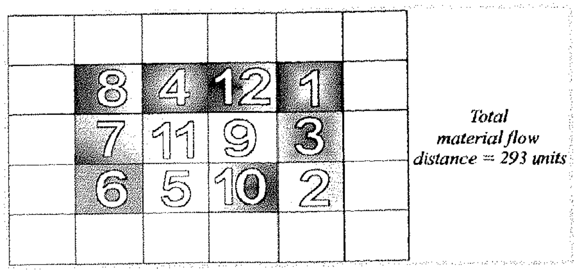
```
0    3    5    0
0   12    6    2
0    1    4    8
0   11    7    9
0   12   10   13
```

Figure 6. 14 - Example of the matrix style output from the DCM placement program

The computer implementation was tested using illustrative examples. Problems are presented in the next sections, with detail explanation of the proposed *SA* procedure. However, formal computational experiments were carried out to validate the performance of simulated annealing, and this is given in section 6.3.7.

6.3.6. Examples – with Comparisons

The example proposed by Rosenblatt [1989 and 1992] is used in this restricted example with a comparison. This example was originally proposed by Hillier [1963 and 1966] but is much better known as the Nugent [1968] examples, and is used in recent literature as the benchmark comparison example for a twelve resource location problem, ie. for the case of a manufacturing cell with twelve machines. The manufacturing resources are considered to be equidimensional and the rectilinear distance criterion is used.



	8	4	12	1	
	7	11	9	3	
	6	5	10	2	

Total material flow distance = 293 units

Figure 6. 15 - Final layout for the example

6.3.6.1. Equal Dimensions of the Resources

The material flow values between manufacturing resources are shown in *Table 6.7* and the *SA* algorithm default control parameters are used (section 6.3.4). The starting temperature *T* and the reduction factor are defined as *30* and *0.9* respectively. The program randomly generated the initial layout on the *6 x 5* grid. The final layout is shown in *Figure 6.15* and corresponds to a total material flow distance of *293* units.

It is noted that the layout presented by Rosenblantt corresponds to a material flow distance of *297* units.

Consequently, the proposed *SA* base layout has yielded a marginally better result, and these results provide some justification for the algorithm used and are indicators of its potential. In addition, in order to compare various results and to provide some means of quantifying the efficiency, a global efficiency statement is proposed as follows:

$$Global\ efficiency = \left(\frac{FD}{PFD}\right) \qquad \textit{Equation 6. 31}$$

Manufacturing resources

Manufacturing resources	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
1	0											
2	5	0										
3	2	3	0									
4	4	0	0	0								
5	1	2	0	5	0							
6	0	2	0	2	10	0						
7	0	2	0	2	0	5	0					
8	6	0	5	10	0	1	10	0				
9	2	4	5	0	0	1	5	0	0			
10	1	5	2	0	5	5	2	0	0	0		
11	1	0	2	5	1	4	3	5	10	5	0	
12	1	0	2	5	1	0	3	0	10	0	2	0

Table 6. 7 - Material flow between resources

Here *FD* is the total material flow handling distance, assuming that all material transfers involve unit distance, and *PFD* is the material flow handling distance for the proposed research method. The higher the value of this criterion, the better the solution. It may be noted that the optimal value of unity can never be obtained for a practical case involving more than two manufacturing resources (as the number of resources increases, the value of this criterion tends to decrease). For this example, the global efficiency is *59.4%*, compared to *58.6%* for the Rosenblantt solution, which is a small improvement in the efficiency of *0.8%*.

6.3.6.2. Unequal Dimensions of the Resources

The example presented in the previous section is based on the assumption that all sizes of the resources are of equal dimensions, and can be placed at any of the positions in the solution space. This, however, is not true in most practical cases. In order to accommodate the approximate size of various resources, each entity is considered to be composed of an integer number of square building blocks.

Dimension	*No. of building blocks*		*1*	*2*	*3*	*4*	*5*	*6*	*7*
1 x 3	3	*1*	0						
1 x 2	2	*2*	9	0					
1 x 2	2	*3*	6	4	0				
1 x 2	2	*4*	0	4	0	0			
1 x 1	1	*5*	0	4	0	4	0		
1 x 1	1	*6*	0	4	4	0	4	0	
1 x 1	1	*7*	3	0	0	4	0	0	0

Table 6. 8 - Resource sizes and flow for the unequal size example

To demonstrate the effectiveness of the proposed research approach in the case of unequal resources sizes, the theoretical example of *Figure 6.14* is again considered. The system contains seven resources as shown in *Table 6.8*. The material flow values between the resources is shown in *Table 6.8*. This data and a random initial placement of the blocks are used as inputs to the layout procedure.

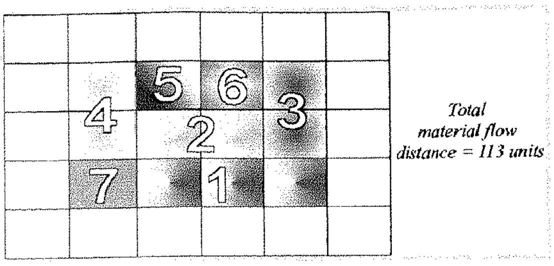


Figure 6. 16 - Unequal sizes layout examples

The material flow between identical blocks is assumed as *2* × the maximum traffic between non-identical blocks. An initial temperature and an annealing schedule from the set up problem section 6.3.3.2 are used. The layout obtained corresponds exactly to the optimal layout of *Figure 6.16.* It can be seen that the blocks representing resources *1* to *4*, which are the ones represented by more than one building block, have been placed together in the final layout.

FROM Resource \ TO Resource	M	1	2	3	M	4	5	M	6	7	M	N7	O7	8	M	9	M	N1	10	M	N1	11	12
M1					6	2			2		1		2										
1			1																				
2					1				1														
3							2																
M4										6	3												
4														11									
5			1						1														
M6						2						3							1				
6						2													2				
7						5								2									
M7															3			3		1			
N7																	3						1
O7															2					1	1		1
8													3			8							
M9																							
9																							
M10																					1		
N10										1			2								1		
10											3										1		
M11												1					1		1				
N11													1					1					3
11																							1
12																							

Table 6. 9 - Travel chart after economical duplication of machines is applied, as input for SA

6.3.6.3. Vakharia Example

In this section the example for consideration is derived from a paper (Vakharia 1990). This example was used also for comparison of the *MWDRST* in the proposed stages II and III of the research methodology presented in Chapter 5.4 and Chapter 6.2. Detailed analysis of the input data is presented in section 5.4.1, together with the first two stages of the solution of the proposed research methodology. *Table 6.1* presents the minimum input data requirement for stage III (the economical duplication of machines), and the final result *(Figure 6.1)* from stage II of the proposed research methodology, together with the From –

To machine material flow matrix (*Table 6.2*). The economical numbers of each type of machine required in the different flowlines were calculated and are shown in *Table 6.3.*

However, after completing the machine requirements for the flowlines, further analysis of the parts whose operational sequences contained arcs were considered for capacity calculations, and process planning or value analysis, in order to eliminate the intercell machine sharing problems created. Evaluation of the material flowline networks is utilised because of having to change the assignment of the flowlines resulting from introducing a duplication of the machines. A new From – To machine chart (matrix) is then developed, and presented in *Table 6.9.* This travel chart *(Table 6.9)* is the main output of stage III, and consequently the input for this stage IV, the economical optimisation of the layout design. The same type resources are named, using a combination of a letter and a number, ie. *M7* means resource number M, resource type *7.* Here the objective function is to minimise the total system cost, which is made up of the material handling cost and duplication cost.

Equal Resource Dimensions

For this example a cellular manufacturing system with twenty-four manufacturing resources is considered (after stage III-economical duplication). The manufacturing resources are considered to be equidimensional and the same rectilinear distance criterion is used as in the previous example. The material flow between resources is that again shown in *Table 6.9.* Each resource is assigned a dimension size of one unit grid, and a material travelling cost of *$1* per unit.

The *SA* algorithm control parameters described and defined in the section 6.3.3.2 were also used. Thus, the program generated the initial layout on the given shop floor size of the 6 × 6 grid. The final layout is shown in *Figure 6.17,* with the total material handling distance of *142* units. In addition, *Figure 6.18* presents the change of the annealing temperature during the *SA* algorithm search of the total material handling cost.

It may be noted that the layout presented in Chapter 5 *(Figure 5.48, Section 5.5.2.6)* corresponds to a material flow distance of *297* units. However this approximate layout was design manually and for fourteen manufacturing resources and a different shop floor sizes. The proposed procedure shows a computanional time of 78 seconds when running a *CPU PENTIUM 90;* an improvement of *53%.*

To compare the approximate layout from Chapter 5 the same example is utilised with the same shop floor size and *5 × 4* grid (without the duplicated resources from stage III). The material flow data input is used from *Table 5.22* (Chapter 5.4) and a uniform cost of *$1* per unit lengths for all flows. Results are presented in *Figure 6.19* and the temperature diagraph in *Figure 6.20.* The material flow distance after the *SA* algorithm is utilised corresponds to *232* units, which is an improvement of *22%* with respect to the *297* units distance of the approximate *DCM* layout *(Figure 5.48, Section 5.5.2.6).*

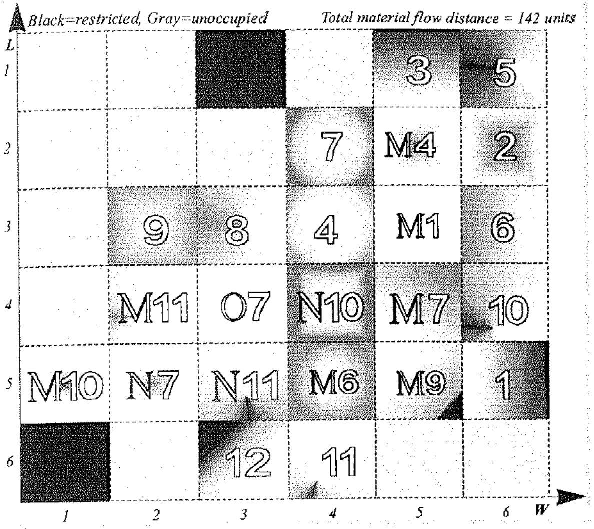


Figure 6. 17 - Final SA based DCM layout

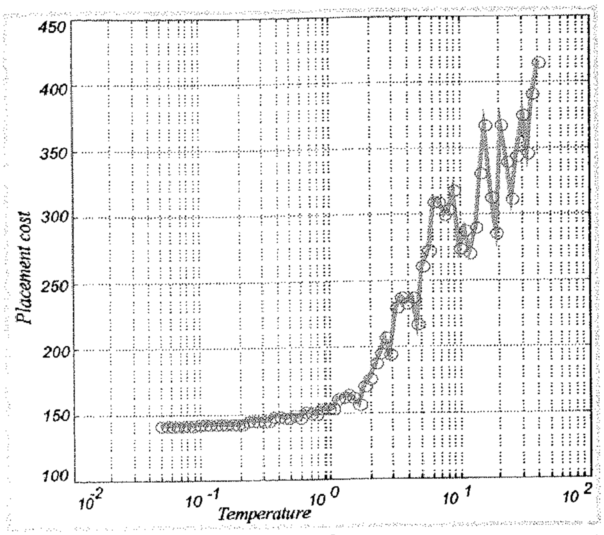


Figure 6. 18 - SA search temperature versus cost

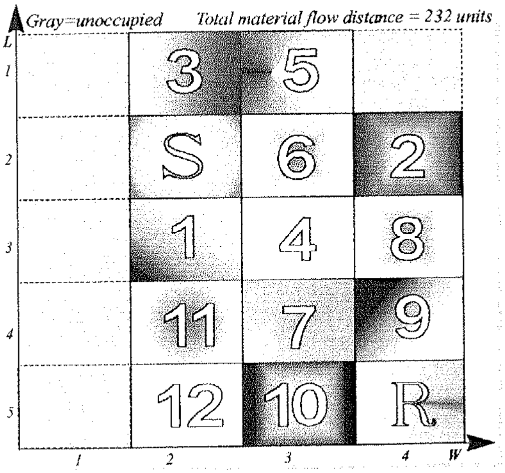


Figure 6. 19 - Final layout for the basic model

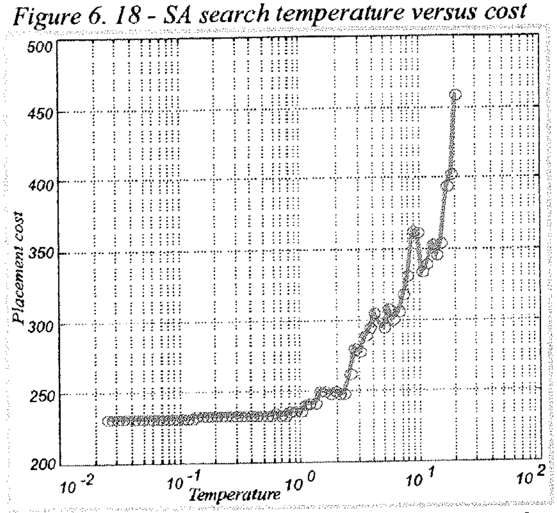


Figure 6. 20 - Corresponding temperature graph

Unequal Resource Dimensions

The example presented in the previous section is based on the assumption that all sizes of the resources are of equal dimensions, and can be placed at any of the positions in the solution space. However, this is not true in most practical cases. In order to accommodate the approximate size of unequal dimensioned resources, each entity is considered to be composed of an integer number of square building blocks (grid units).

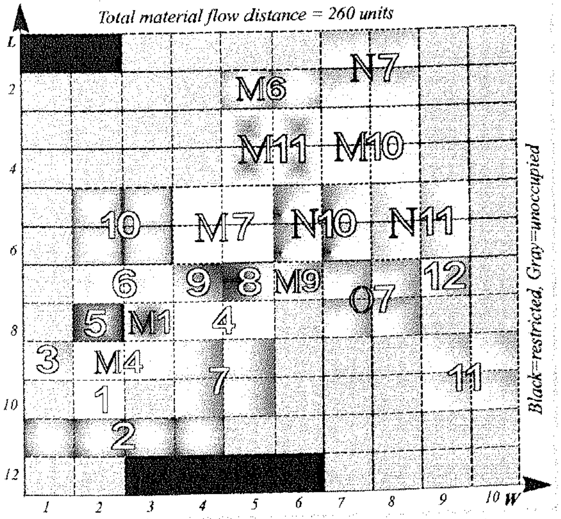


Figure 6. 21 - Final unequal DCM layout

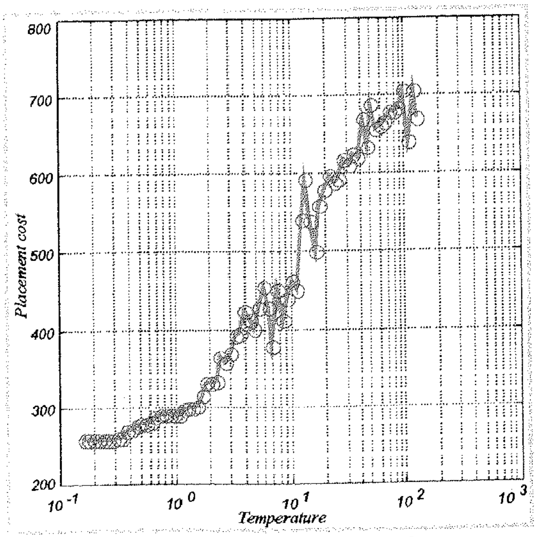


Figure 6. 22 - Corresponding SA search temperature

Dimension	*No. of building blocks*	***Resource type***	***Cost***
1 x 1	1	***1***	1
1 x 4	4	***2***	1
1 x 1	1	***3***	1
1 x 2	2	***4***	1
1 x 1	1	***5***	1
1 x 2	2	***6***	1
2 x 2	4	***7***	1
1 x 1	1	***8***	1
1 x 1	1	***9***	1
2 x 2	4	***10***	1
2 x 2	4	***11***	1
1 x 1	1	***12***	1
1 x 2	2	***Restriction A***	1
1 x 4	4	***Restriction B***	1

Table 6. 10 - Resource sizes and flow for the unequal size example

Here the initial From - To chart from *Table 6.9,* together with the resource and restrictions data from *Table 6.10* were the input for the *SA DCM* layout algorithm. A random initial placement of the resources is then applied to the shop size of a *12 × 10* grid. The algorithm resulted in a material flow cost distance of *260* units, with a *CPU* time of *65* seconds. The optimal *DCM* layout is presented in ***Figure 6.21*** with the corresponding temperature graph in *Figure 6.22.*

6.3.7. Analysis of Parameters

To find the appropriate parameters for the annealing procedures necessitated the testing and comparing these parameters with the parameters provided by Kouvelis et al. [1992] who changed one parameter at a time. Parameter values provided by Kouvelis et al. are as follows: *P_o =0.4, T_{FC} = 0.99 and L = 0.95.* The problem size utilised in the parameter analysis was *Z = 12* and is applied to the Nugent [1968] problems. Ten replications with different random searches were run for each combination of parameters, with the value of the parameter that had the best average solution quality being chosen. The parameter analysis results are shown in *Tables 6. 11 - 6.14.* The standard (default) parameters set used for the simulated annealing procedures are shown in *Table 6.6.*

Parameter P_o	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***
AVG cost	289	289	289	290	292
AVG time (sec)	10.5	11	10	9.5	10

Table 6. 11 - SA parameter P_0 (probability of the annealing process)

Parameter T_{FC}	***0.99***	***0.95***	***0.9***	***0.85***	***0.8***
AVG cost	291	289	289	289	290
AVG time (sec)	10	11	10	10	12

Table 6. 12 - SA parameter T_{FC} (temperature decrease factor)

Parameter L	***1***	***0.95***	***0.9***	***0.88***	***0.8***
AVG cost	289	289	289	289	289
AVG time (sec)	11	10	11	11.5	10.5

Table 6. 13 - SA parameter L (length of the search)

With respect to *Tables 6.11 – 6.14* and from papers Kouvelis [1992], Willhelm [1990] and Heragu [1993], it has been suggested that *SA* is very sensitive to the annealing parameters. However, these parameters do not effect the quality of the solutions. Thus, there is a requirement for the multiply running of the program for each problem to find the optimum value of the objective function (total travelling cost).

Parameter T_F	***0.01***	***0.1***	***0.2***	***0.3***	***0.4***
AVG cost	289	289	289	289	289
AVG time (sec)	12.5	12	11	10	9

Table 6. 14 - SA parameter T_F (stopping temperature)

The next step in this comparative analysis is to analyse the change of the configuration (perturbation) and the generation factor (swap, translate and rotate). Designing an effective change configuration mechanism of the random solution is one of the essential components in applying the *SA* algorithm, and is relevant to the proposed research. Focusing on the configuration change of the random permutations, it is now required to investigate the generation scheme property. The four classical Nugent [1968] problems are used for this phase of the experiment. The sizes of the problem are eight, twelve, fifteen and twenty facilities for the *QAP* problem. Objective functions (total distance) with optimum minimum values of *107, 289, 575* and *1285* respectively are selected for these problems. The results of the experimental evaluation for solving these four problems by the *SA* algorithm for these three operations of the configuration changes are presented in *Table 6.15.*

Problem size	***Change of the configuration***	***Ratio***	***Mean***	***Variance***	***Number of the iteration***
Z = 8	*Swap*	84%	0.4	0.79	1500
	Translate	84%	0.42	0.88	
	Rotate	80%	0.78	2.49	
Z = 12	*Swap*	35%	1.19	0.89	2940
	Translate	19%	3.39	3.51	
	Rotate	5%	5.26	2.41	
Z = 15	*Swap*	30%	1.21	0.77	4580
	Translate	15%	3.78	1.44	
	Rotate	5%	4.36	1.85	
Z = 20	*Swap*	24%	1.78	0.71	5940
	Translate	5%	7.14	1.19	
	Rotate	5%	6.96	0.65	

Table 6. 15 - The results of solving QAP by the SA with the three changes of the configuration scheme

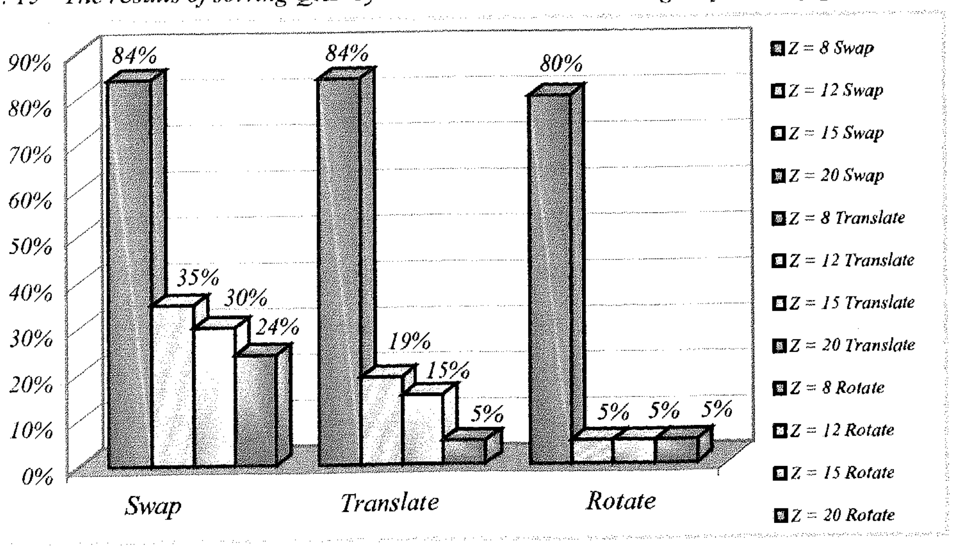


Figure 6. 23 - Comparisons of the configuration changes with regard to the optimal ratio of the QAP

These results *(Table 6.15 and Figure 6.23)* show that *SA* can produce very efficient solutions to combinatorial optimisation problems, and that the best change of configuration (perturbation) scheme is swapping (interchange) of the facilities, or locations, for the *QAP* or facility layout problem. The second important configuration scheme (operations) is translation, and the third and last is rotation. These findings validated the proposed research *SA* scheme (operations) from section 6.3.3.2. for the change of configuration. In addition, this comparative analysis shows that the parameters of the *SA* algorithm, which are proposed in this research, are adequate in addressing the problem of designing a *DCM* layout. It should be noted however that if the proposed *SA* algorithm is going to be used for other combinatorial optimisation (configuration) problems, adequate parameter analyses should be undertaken.

6.3.8. Computational Experience

The above simulated annealing procedures used the cooling schedule and parameters in accordance with *Section 6.3.4.* and *Table 6.6*. This procedure was programmed in *MATLAB* language and run on a *PC* with a *90 MHz* Pentium *CPU*. The test problems were randomly generated with feasible resource dimensions. Each test problem was run twenty times with different random seeds.

	Problem size				
Method	***8***	***12***	***15***	***20***	***30***
Nugent [1968]	107	293	580	1313	3124
Heragu *SA* [1992]		291	577	1295.6	3107.8
Heragu *HSA* [1992]		289	576.5	1286.8	3069.2
Willhelm *SA* [1987]		291	578.2	1308	3099.8
Connolly [1990]		293.5	583.3	1305.4	3094.1
Connolly II [1990]			577.8	1296.3	3096
Proposed *SA* method	107	289	575.5	1286.6	3069.3

Table 6. 16 - Comparison between mean costs generated by the proposed and other SA methods

For the equal resource sizes the test data from Nugent [1968] was used. These problems relate to sizes of twelve to thirty equal shape-size resources and they have been addressed by many researchers to demonstrate the effectiveness of different algorithms, including simulated annealing. These methods use different annealing parameters or different changes of the configuration operations (factors) solutions, and some researchers such as Jajodia [1992], Bazargan–Lari [1996, 1998], Wang [1998] and Kouvelis [1993] did not report averages over several runs, and thus were not included in this *Table 6.16*.

In this section, examples for unequal resources size are compared and the two examples of twelve and fourteen facilities test data from Bazaraa [1975] are used as test data. The material flow and dimensions of the resources is included in the data set. The material handling cost from one resource to another resource is assumed (as per in the literature) to be constant ($ 1/ unit distance). *Table 6.17* shows the result obtained by Bazaraa [1975] and the proposed *SA* method for the twelve and fourteen resources test problems.

Method	*12 Size problem*	*14 Size problem*
Bazaraa [1975]	14079	8170.5
Proposed SA method	12111	7040

Table 6. 17 - Comparison with Bazaraa's total cost model

This computational comparison section justifies the superiority of *SA* based algorithms with respect to facility layout methods, which has also been conclusively reported by many researchers. In their work mainly sets of classical or cell layout problems were solved by several methods, in addition to *SA*. In all cases, *SA* performed better than all the other methods. The proposed research method presented here is the most advanced version to date of *SA* based algorithms, since it considers:

- the size of manufacturing resources,
- restricted areas, and
- material flow paths that cannot pass through manufacturing resources and restrictions.

6.4. Conclusions

This chapter has proposed and developed a research methodology for economical duplication of the manufacturing resources in a *DCM* system, and a simulated annealing based method for generating shop layouts. The first section of this chapter describes the stage III approach for economical resource duplication of the material flowline layout, which is an extension of the research methodology presented within stages I and II (Chapter 4 and 5). In the proposed mathematical model additional system/machine variables are included for finding the required number of the flowline machines. These variables are: setup time, duration of the production period, machine capacity, utilisation of each machine type, flow volume, production volume, operation time, non operation time, average time between failures, and the servicing time and availability of the machine type.

In addition, the proposed dynamic cell system which considers the economical duplication model for the research methodology is utilised and compared with an illustrative example from Vakharia [1990]. Comparing the average utilisation of the resources shows promising results; in the dynamic cell it is *77%* (system utilisation) and it is also higher than the average utilisation *51%* from Vakharia [1990] (classical cellular system). Further analysis clearly shows a visible improvement of *50%* in the utilisation of the dynamic cell system organisation compared to the classical cell. In addition, it is noted that in the proposed research methodology for the design of the *DC*, not all the available machines of the classical cell were utilised, which should give higher utilisation. However, it is suggested that the presented utilisation results justify the first three stages of the proposed research methodology. It is also recognised, however, that graph theory and network analysis are still valuable tools for presentation and analysis of the material flows, especially for the directed tree's theory.

The second part of this chapter presents the proposed research methodology for stage IV, the facility layout problem for a dynamic cell. The major issue here is the placement of manufacturing resources (cells and machines) within the available area of the shop floor. A simulated annealing *(SA)* based algorithm is developed, improved and utilised to obtain a near optimal solution for the layout problem, which is *NP* complete. The evaluation of the objective function at each step of the proposed research methodology is accomplished by applying the *SA* algorithm. Since the shop layout problem is an assignment problem, local improvement solution approaches are prone to converge to a local minimum, especially in the cases of shop design in which the objective function comprises several conflicting factors. The simulated annealing method has been previously shown to converge to global near optimal solutions for the *NP* problems. This is the main reason it has been utilised in this proposed research methodology for stage IV. A mathematical model has also been developed to examine resource layout, and an analysis is presented with various practical aspects. These constraints are the restricted areas, irregularity shapes of the resources, shop floor shapes, and equal and unequal dimensions of the resources. It should be noted that most of the past research work concentrates on layout facilities of equal sizes and shapes, and do not consider the material handling cost. However, the proposed research methodology advances current work by considering unequal shapes, as well as material handling costs and the restrictions within a shop floor size. Finally, an interactive chart for the design of the *DCM* system layout (stages I, II, III & IV of the proposed research methodology) is presented in *Figure 6.24*.

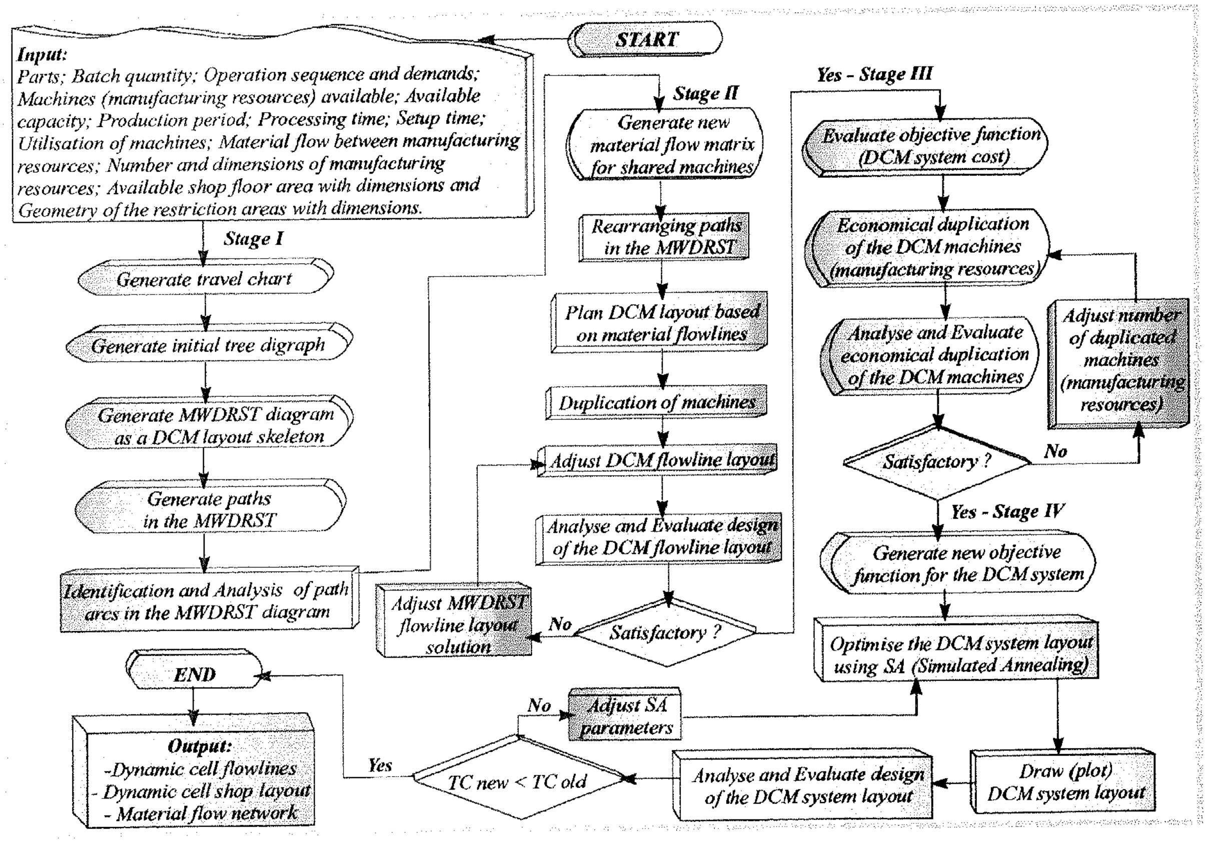

START
Input:
Parts; Batch quantity; Operation sequence and demands; Machines (manufacturing resources) available; Available capacity; Production period; Processing time; Setup time; Utilisation of machines; Material flow between manufacturing resources; Number and dimensions of manufacturing resources; Available shop floor area with dimensions and Geometry of the restriction areas with dimensions.
Stage I
Generate travel chart
Generate initial tree digraph
Generate MWDRST diagram as a DCM layout skeleton
Generate paths in the MWDRST
Identification and Analysis of path arcs in the MWDRST diagram
Stage II
Generate new material flow matrix for shared machines
Rearranging paths in the MWDRST
Plan DCM layout based on material flowlines
Duplication of machines
Adjust DCM flowline layout
Analyse and Evaluate design of the DCM flowline layout
Satisfactory ?
No
Adjust MWDRST flowline layout solution
Yes - Stage III
Evaluate objective function (DCM system cost)
Economical duplication of the DCM machines (manufacturing resources)
Analyse and Evaluate economical duplication of the DCM machines
Adjust number of duplicated machines (manufacturing resources)
Satisfactory ?
No
Yes - Stage IV
Generate new objective function for the DCM system
Optimise the DCM system layout using SA (Simulated Annealing)
Draw (plot) DCM system layout
Analyse and Evaluate design of the DCM system layout
TC new < TC old
No
Adjust SA parameters
Yes
END
Output:
-Dynamic cell flowlines
- Dynamic cell shop layout
- Material flow network

Figure 6. 24 - An interactive chart for the design of the DCM system layout (stages I, II, III & IV)

In summary, to formulate the shop floor layout design problem, first of all a geometrical model of the shop floor and of the manufacturing resources is developed. Subsequently, decision variables are introduced to model discrete choices concerning the construction of the flow network and the location of the resources on the shop floor. Finally, the assumptions guiding the design are stated and an integer programming formulation of the layout problem is developed. In addition, the discrete block layout problem with shape constraints is developed and presented along with the solution approach, an illustrative example, and comparisons with other algorithm methods from literature case studies.

In the worked examples, parametric analyses of the *SA* algorithm is presented and utilised from known literature problems. This analysis shows an improvement in the responsiveness and effectiveness of the proposed *SA* layout procedure. Also reported are the mean values of the objective function (material flow cost), which also is an improvement with respect to current methodology, because researchers rarely include in their reports parametric analysis results. The *SA* algorithm was coded in *MATLAB* (attached in Appendix D), and has matrix and graphical interface outputs, which is again a further step forward in this research area. Comparison examples are utilised for both equal and unequal resource dimensions, with the average value of the material flow cost (objective function) comparable with other methodologies. It should be appreciated however, that running times of the programs (*CPU* time) are not comparable because different algorithms, which are presented to date in the literature, were utilised on different platforms, and with various computer languages or programs.

CHAPTER 7

Summary and Conclusions

7.1. Introduction

This thesis examines the problem of designing the manufacturing shop system in an innovative and comprehensive manner. The research motivation stemmed from the significant savings that could be realised from an effective manufacturing system design in terms of investment (layout design) and operational costs (material flow handling system). The shop design problem is particularly complex and here new methods for both the dynamic cell formation and the layout problems are presented. Both methods address a host of practical issues, some of which have either been ignored or inadequately treated by past methodologies. In addition, a critical review of state-of-the-art design and layout of cellular manufacturing systems which details the various judgemental criteria utilised is discussed. Subsequent analysis introduces the methodologies for the integrations of the manufacturing cells into dynamic cell configurations.

7.2. Summary and Research Critique

This section contains a summary and research critique of the dynamic cellular manufacturing design methodology presented in this thesis. The summary section highlights problems in the existing approaches to cell formation with a focus on the methodology for dynamic cell facility design. In the research critique section new contributions with major features to the proposed methodology are described.

7.2.1. Summary

It is noted that existing approaches to the cell formation problem concentrate on simultaneous part family formation and machine sharing, and leads to cell underutilisation. These approaches do not use flow data, and they cannot solve together the machine grouping, machine sharing, intracell layout, intercell layout and handling subproblems. To address these cellular manufacturing design problems clearly part family formation and machine capacity calculations alone cannot accomplish effective cell design. Thus, this thesis proposes that decisions concerning machine groupings need to be made after formulating intracell and intercell layout and handling solutions, to simplify the machine sharing problem. This work addresses, for the first time, an analytical approach to the integrated problems of designing the dynamic cellular manufacturing (*DCM*) system layout concurrently with its material flow (handling) requirements, in such a manner that minimises the material handling within the system. For these reasons the proposed strategy encourages the simultaneous design of a dynamic layout to identify the machine groups, economical machine duplication, and intracell and intercell layouts.

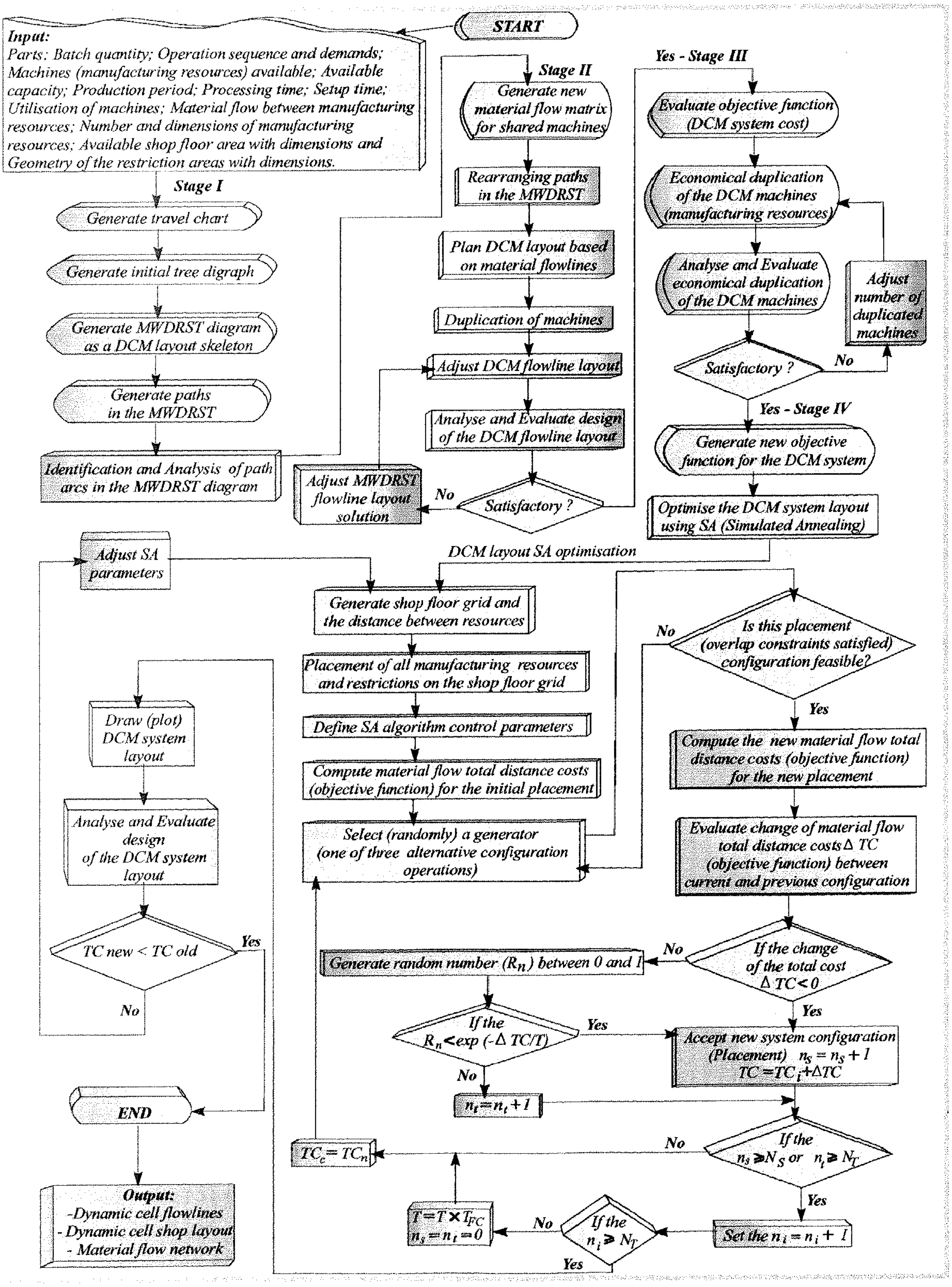

START
Input:
Parts: Batch quantity; Operation sequence and demands; Machines (manufacturing resources) available; Available capacity; Production period; Processing time; Setup time; Utilisation of machines; Material flow between manufacturing resources; Number and dimensions of manufacturing resources; Available shop floor area with dimensions and Geometry of the restriction areas with dimensions.
Stage I
Generate travel chart
Generate initial tree digraph
Generate MWDRST diagram as a DCM layout skeleton
Generate paths in the MWDRST
Identification and Analysis of path arcs in the MWDRST diagram
Stage II
Generate new material flow matrix for shared machines
Rearranging paths in the MWDRST
Plan DCM layout based on material flowlines
Duplication of machines
Adjust DCM flowline layout
Analyse and Evaluate design of the DCM flowline layout
Satisfactory ?
No
Adjust MWDRST flowline layout solution
Yes - Stage III
Evaluate objective function (DCM system cost)
Economical duplication of the DCM machines (manufacturing resources)
Analyse and Evaluate economical duplication of the DCM machines
Adjust number of duplicated machines
Satisfactory ?
No
Yes - Stage IV
Generate new objective function for the DCM system
Optimise the DCM system layout using SA (Simulated Annealing)
DCM layout SA optimisation
Adjust SA parameters
Generate shop floor grid and the distance between resources
Placement of all manufacturing resources and restrictions on the shop floor grid
Define SA algorithm control parameters
Compute material flow total distance costs (objective function) for the initial placement
Select (randomly) a generator (one of three alternative configuration operations)
No
Is this placement (overlap constraints satisfied) configuration feasible?
Yes
Compute the new material flow total distance costs (objective function) for the new placement
Evaluate change of material flow total distance costs Δ TC (objective function) between current and previous configuration
If the change of the total cost Δ TC<0
No
Yes
Generate random number (R_n) between 0 and 1
If the R_n<exp (-Δ TC/T)
Yes
No
Accept new system configuration (Placement) $n_s = n_s + 1$ $TC = TC_i + \Delta TC$
$n_t = n_t + 1$
If the $n_s \geq N_S$ or $n_t \geq N_T$
No
Yes
$TC_c = TC_n$
$T = T \times T_{FC}$ $n_s = n_t = 0$
No
If the $n_i \geq N_T$
Yes
Set the $n_i = n_i + 1$
Draw (plot) DCM system layout
Analyse and Evaluate design of the DCM system layout
TC new &lt; TC old
Yes
No
END
Output:
-Dynamic cell flowlines
- Dynamic cell shop layout
- Material flow network

Figure 7. 1 An interactive chart for the design of the DCM system layout (stages I, II, III & IV)

The presented methodology for dynamic cell facility design *(Figure 7.1)* may be used for the redesign of existing manufacturing shops, and the design of planned facilities with known production period product demands and defined process routing's. The methodology compromises four stages:

1. formation of dynamic manufacturing cells,
2. evaluation and determination of intercell and intracell layouts,
3. economical duplication of machines-resources in the dynamic cells, and
4. determination of the layout of the dynamic cell resources.

The dynamic cell formation method minimises the intercell material flow within the shop, and the allocation of identical machines to various cells, in a way consistent with these objectives, whilst respecting the capacity of each machine. Given a set of production routines for the parts manufactured in the system, the method also selects specific machines on which these parts are to be processed. Although the primary application of the system would be to rearrange an existing job shop into dynamic manufacturing cells, it is equally applicable to the planning of resources with a forecasted production plan and known manufacturing processes. The major features of the proposed dynamic cell formation system can be summarised as follows:

- identical machines are assigned to cells based on both material flow traffic and capacity considerations,
- the sequence of operations is taken into account whilst evaluating the material flow traffic between resources (machines and/or cells),
- multiple operations on the same machine type are taken into account and are individually treated,
- the method minimises the total number of intercell movements of pallets, as opposed to the movements of individual parts.
- both setup time and run times, along with the average batch sizes, are used to compute the capacity requirements for each part, and
- a machine duplication algorithm provides simulations for determining the optimum number of machines of each type to minimise the material flow traffic, whilst ensuring sufficient capacity availability.

At stage I (Chapter 4; section 4.6: A dynamic cell shop design method) a comprehensive mathematical model is formulated which captures the interrelated decisions associated with the placement of the resources on the shop floor, the design of the material flow network, and the sequence of material flow operations. The objective here accounts for the fixed costs related to the flow network and for the variable (operational) costs of material flow handling. Thus the proposed mathematical models are based on network analysis and graph structures for flowline decomposition of machine groupings, and flowline layout design of dynamic cellular manufacturing systems. Therefore, at the first stage, an algorithm is developed for the flowline solution of the material flow network, which is based on the *MWDRST* algorithm-heuristic for the directed network design. To minimise travel distances for forward and backward material flow arcs, the derived model minimises total travel distances and machine duplications. In the second stage (Chapter 4; section 4.7) the developed algorithm solves the crisscrossing flow network arcs, utilising a *SDPI* heuristic algorithm for the *QAP* assignment problem. Consequently, stages I and II of this research generates machine groups, identifies a flowline layout for each group, indicates which flowlines must be placed adjacent to each other to minimise intercell distances, and an approximate configuration of the aisles. Thus by capturing the directionality embedded in the operational sequences of a variety of parts produced, the associated facility layout area can be optimised. It is concluded that the classification of flow arcs used is effective for assessing whether handling and layout, or machine sharing, is necessary to minimise intracell or intercell travel distances.

As a development from stages I and II, Chapter 6 stages III (section 6.2) and IV (section 6.3) outlines the associated economics of machine duplication and layouts of a dynamic cellular manufacturing facility. Furthermore, by employing simulated annealing algorithms the design (placement) of the shop layout for these dynamic cells can be optimised. In this chapter the method of economical machine duplication is also discussed. Thus, the economical duplication model for the third stage is employed for the machine sharing problem, utilising additional shop floor constraints such as: setup time, utilisation machine availability and production planning period. Another unique feature of the proposed methodology is the development of a strong link between the layout and machine duplication processes. In the last stage (IV) a simulated annealing scheme is adopted for the generation of feasible layouts and the subsequent convergence to a near optimal dynamic cell shop layout design. The simulated annealing scheme starts from a randomly generated shop layout - the complementary material flow handling system is designed at stages I and II - and proceeds by generating alternative adjacent layouts, until certain convergence criteria are satisfied. Thus the machine duplication and layout design processes are put in an iterative loop procedure to investigate all machines which are nominated for duplication.

Once the sizes and shapes of the dynamic cells and the manufacturing resources are determined, the shop layout is designed to minimise the traffic and distance travelled by the parts within the shop. The resources could consist of both cells and independent machines, or just cells for instance. The simulated annealing algorithm is utilised to iteratively swap, translate, or rotate resources until a near optimal solution is obtained. Operations are re-assigned to functionally identical machines according to material handling considerations where it is appropriate. The major contributions of this stage IV include:

- consideration of the resources (dynamic cells and machines) in a unified manner,
- consideration of physical constraints, such as actual shop size and shape, as well as restrictions, and
- reassignment of operations to appropriate machines.

7.2.2. Research Critique

One of the most important contributions of this methodology is its practicality. Major features include:

1. Formation of the dynamic cells, whilst respecting machine capacity.
2. Evaluation of the dynamic cells with respect to savings in material handling and robustness in demand changes (which is appropriate for agile manufacturing implementation).
3. Economical machine duplication with respect to resource capacity and manufacturing systems constraints.
4. Design of a dynamic cell shop layout with respect to the resources - particularly appropriate for the majority of practical manufacturing layout configurations.
5. Inclusion of resource dimensions (or areas) and shapes in the resource size constraints, giving greater significance to a manufacturing facility containing entities with a wide range of sizes and shapes.
6. Consideration of the actual shopfloor dimensions, which with walls and other obstacles may impede regular material flow traffic.
7. Formulation of a integer programming model for the dynamic cell shop design problem, which integrates the layout of the manufacturing resources and the material flow network (Hitherto published work has only considered such decisions separately).
8. Development of an effective simulated annealing optimisation based method to systematically generate and evaluate shop layouts by invoking the material flow network design algorithm. (This method converges to a near optimal dynamic cell shop design, which includes the layout design of the resources on the shop floor and the material flow network.)

7.3. Research Overview

The proposed dynamic cellular manufacturing layout method (Chapters 3 to 6) overcomes the major drawback of other methods that have attempted to address this problem, which have been the dependence of the final solution on the initial shop configuration. Thus the proposed solution obtained typically corresponds to a global optimum or a near optimal solution for the given problem. Additionally, several alternative layouts of nearly the same quality are obtained, the most suitable being selected for implementation. This is possible since the proposed method is a probabilistic one, and consequently different solutions can be obtained starting from the same initial solutions. This is not possible in deterministic heuristics like *CRAFT*, *CORELAP*, etc., wherein for a given set of initial conditions, the same final solution is obtained. Different initial layouts have to be supplied to these methods to generate alternative solutions. This becomes very difficult and tedious as the physical magnitude of the problem increases.

An attempt has also been made to account for rectangular resource sizes, which clearly is of practical significance. This feature, subjected to rigorous testing, has the potential for application to shopfloor scenarios wherein the previous assumption of equidimensionality for the resources might lead to unrealistic solutions. Use of this proposed feature in selected problems from literature has resulted in some success in obtaining the optimal layout for resources with unequal area requirements. Overall, integrating the approaches developed for the components (cell formation; layout simulation; machine sharing & economical duplication and *MWDRST* intracell and intercell flowline design skeleton) of the dynamic manufacturing problem produces encouraging solutions. The method proposed in this thesis is one of only a few to address all the issues related to dynamic shop design in an 'integrated' manner.

7.4. Conclusions

The following conclusions were obtained from the development and application of the proposed dynamic cell facility design methodology:

- A pure cellular arrangement is not practical for many advanced industrial environments.
- The dynamic cell facility design problem is conventionally decomposed as a sequence of the subproblems, ie. dynamic cell formation, dynamic cell layout and shop layout.
- These subproblems may be solved more than once to arrive at an optimal solution.
- It is necessary to consider a common objective for all stages of the facility design problem.
- Practical issues, such as setup time and production mix, should be considered in the final solution to have practical significance,
- The objective of minimising material handling within the production shop is critical. This is validated by implementation of examples from the literature.
- The proposed methodology addresses critical practical issues and provides solutions that result in improved shop performance.
- Workload criteria such as machine utilisation and availability of machines are incorporated in the dynamic cell algorithm for the assignment of parts to machines.

7.5. Recommendations for Future Work

Issues that will enhance the applicability of the proposed dynamic cell facility design methodology are discussed in the following sections.

7.5.1. Cell Formation Stage

Issues that could be addressed by a grouping algorithm which considers the assignment of parts to machines are:

1. Incorporation of economic considerations, such as cost estimates associated with machine relocation, and
2. Incorporation of workload considerations, such as machine utilisation and load balancing.

7.5.2. Intercell Layout Stage

An important issue here is the design of a shop traffic corridor system which prevents material congestion and is cost effective. In this current study, inter - resource corridors are defined from the *MWDRST* between the corresponding resources. However, this may potentially create a very large number of corridors, which is clearly a cost ineffective solution. In addition, the flow along these corridors may be unbalanced, resulting in highly congested and/or rarely used flow paths. To address these issues emphasis should be given to determining intercell avenues where the majority of material flow should occur.

7.5.3. Intercell and Intracell Material Handling Systems

Clearly a well designed dynamic cell or shop with an inefficient material handling system will perform poorly. If certain material handling systems are already in place, the rearranged shop layout should be created with these systems in mind. Alternatively, a methodology that identifies opportunities to use a new material handling system is desirable. Obviously, the cost of implementation these material-handling systems should also be considered.

7.5.4. Integration of *GT* Coding

The integration of *GT* coding with the proposed methodology should standardise products, as well as production methods, leading to an effective transformation of the manufacturing system to independent cells. Thus, planning and scheduling should be simplified.

7.5.5. Physical Criteria as Inputs to the System

Finally, the proposed methods could be enhanced by including part dimensions, weight and volume as inputs to the system. This feature, coupled to a graphics interface for simulation purposes, would aid decision making regarding intercell and intracell material handling system requirements.

7.5.6. Integration of an Expert System

The enhanced method (section 7.4.5) could be integrated into an expert system to address the entire decision making process during the layout design of the manufacturing resources. Such a prototype expert system would comprise methods for machine selection, machine grouping, shop floor layout, shop control and system simulation.

Although this thesis proposes a robust design methodology for a dynamic cell shop, the above enhancements are important in developing optimal or near optimal solutions and will impact both the facility redesign and new facility synthesis problems.

References

1. Aarts E. and Korst J., *(1996),* Simulated annealing and Boltzman machines. *John Wiley.*

2. Aho A., *(1974),* The design and analysis of computer algorithms. *John Wiley.*

3. Aho A., *(1983),* Data structure and algorithms. *Prentice Hall.*

4. Ahuja R.K., Magnanti T.L. and Orlin J. B., *(1993),* Network flows. *Prentice Hall.*

5. Alfa A.S. and Heragu S.S., *(1992),* A hybrid simulated annealing based algorithm for the layout problem. *European Journal of Operational Research, Vol. 57, pp. 190-223.*

6. Alfa A.S., Chen M. and Heragu S.S., *(1992),* Integrating the grouping and layout problems in cellular manufacturing systems. *Comput. Ind. Eng., No. 23, pp. 55-58.*

7. Aneke N.A.G. and Carrie A.S., *(1984),* A comprehensive flowline classification scheme. *Inter. Jour. of Prod. Research, Vol. 22, No. 2, pp. 281-297.*

8. Aneke N.A.G. and Carrie A.S., *(1986),* A design technique for the layout of multi-product flowlines. *Inter. Jour. of Prod. Research, Vol. 24, No. 3, pp. 471-481.*

9. Ang C.L. and Willey P.C., *(1984),* A comparative study of the performance of pure and hybrid GT manufacturing systems. *Int. Jour. Prod. Res., Vol. 22, pp. 193-233.*

10. Askin R.G. and Zhou M., *(1998),* Formation of independent flowline cells based on operation requirements and machine capabilities. *IIE Transactions, No. 30, pp. 319-329.*

11. Askin R.G. and Ciarallo F., *(1996),* A material flow based evaluation of layouts for agile manufacturing. *In Progress in Material Handling Research, Ed. Graves R.J., pp. 71-90.*

12. Apple J.M., *(1963),* Plant layout and materials handling. *Ronald Press, New York.*

13. Apple J.M., *(1977),* Plant layout, Ronald Press, *New York, John Wiley.*

14. Armour G.E. and Buffa E.S., *(1963),* A heuristic algorithm and simulation approach to relative location of facilities. *Management Science, No. 9, pp. 294-309.*

15. Banerjee P. and Jones M., *(Jan 1992),* Parallel simulated annealing algorithms for cell placement on hypercube multiprocessors. *IEEE Transactions, Vol.1, No.1, pp. 91-106.*

16. Bazaraa M.S. and Jarvis J.J., *(1990),* Linear programming and network flows. *John Wiley.*

17. Bazaraa M.S. and Kirca O., *(1983)* A branch-and-bound heuristic for solving the quadratic assignment problem. *Naval Research Logistics, Vol. 30, No. 2, pp. 287-304.*

18. Bazaraa M.S. and Sherali H.D., *(1982)* On the use of exact and heuristic cutting plane methods for QAP. *Journal of Operations Management, Vol. 33, pp. 991-1003.*

19. Bazaraa M.S., *(1975),* Computerised layout design: a branch and bound approach. *AIIE Transactions, Vol. 7, No. 4, pp. 432-437.*

20. Bazargan-Lari M. and Kaebernick H., *(1997),* An approach to the machine layout problem in a cellular manufacturing environment. *Prod. Plan. and Cont., Vol. 8, No. 1, pp. 41-55.*

21. Bazargan-Lari M. and Kaebernick H., *(1996),* Intra-cell and inter-cell layout designs for cellular manufacturing. *Inter. Jour. of Indus. Engin., No.3, pp. 139-150.*

22. Bazargan-Lari M., *(1998),* Layout design in cellular manufacturing. *European Journal of Operational Research, No. 112, pp. 258-272.*

23. Ballakur M. and Steudel, H.J., *(1987),* A within cell utilisation based heuristic for designing cellular manufacturing systems. *Int. Jour. Prod. Res., Vol.25, No5, pp. 639-648.*

24. Bentley J.L., *(1990a),* Experiments on travelling salesman heuristics. *In Proc. 1st Ann. ACM- SIAM Symp. on Discrete Algorithms, SIAM, Philadelphia, PA, pp. 91-99.*

25. Bentley J.L, *(1992),* Fast algorithms for geometric travelling salesman problems. *ORSA Jour. Comput. No. 4, pp. 387-411.*

26. Bentley J.L and Johnson D., *(1996),* Experimental analysis for the Held-Karp travelling salesman bound. *Proc. 7th ACM SIAM Symp., Philadelphia,.*

27. Black J.T., *(1995),* The design of the factory with a future. *Prentice Hall.*

28. Bland R.G. and Shallcross D.F., *(1989),* Large travelling salesman problems arising from experiments in X-ray crystallography. *Operations Res. Lett., No. 8, pp. 125-128.*

29. Bock F.C., *(1971),* An algorithm to construct a minimum directed spanning tree in a directed network. *Developments in operations research, ed. B. Avitzak, pp. 29 – 44.*

30. Boese K.D., Kahng A. B. and Muddu S., *(1994),* A new adaptive multistart technique for combinatorial global optimisations. *Operations Res. Lett., No. 16, pp. 101-113.*

31. Bonomi E. and Lutton J.L., *(1984),* The N-city travelling salesman problem: statistical mechanics and the Metropolis algorithm. *SIAM Rev., No. 26, pp. 551-568.*

32. Borovits I. and Ein Dor P., *(1977),* Cost utilisation: a measure of system performance. *Comunications ACM, Vol 20, No 3, pp. 185-191.*

33. Bozer Y.A., Meller R.D. and Erlebacher S. J., *(1994),* An improvement type layout algorithm for multiple floor facilities. *Management Science, Vol. 40.*

34. Bozer Y.A. and Meller R.D., *(1996),* A new simulated annealing algorithm for the facility layout problem. *Inter. Jour. Prod. Res., Vol. 34, No. 6, pp. 1675-1692.*

35. Bozer Y.A. and Meller, R.D., *(1997),* A reexamination of the distance-based facility layout problem. *IIE Transactions, Vol. 29, No. 7, pp. 549-560.*

36. Buffa E.S., Armour G.C. and Vollman, T.E., *(1964),* Allocation facilities with CRAFT. *Harward Business Review, Vol. 42, No. 2, pp. 136-158.*

37. Buffa E.S., *(1995),* Sequence analysis for functional layouts. *The Journal of Industrial Engineering, Vol. 12-13, No. 25.*

38. Burbidge J.L., *(1979),* Group Technology in the Engineering Industry. *Prentice Hall.*

39. Burbidge J.L., *(1990),* Data Base for Production Management. *Prentice Hall.*

40. Burbidge J.L., *(1971),* Production Flow Analysis. *Produc. Engineer, No.50, pp. 139-152.*

41. Burbidge J.L., *(1963),* Production Flow Analysis. *Produc. Engineer, No. 42, pp. 742-756.*

42. Burbidge J.L., *(1977),* A manual method of Production Flow Analysis. *Production Engineer, No.10, pp. 34-38.*

43. Burbidge J.L., *(1995),* Back to production management. *Manufac. Engineering, pp. 66-71*

44. Burkard R.E., *(1984),* Quadratic assignment problems. *European Journal of Operational Reseasrch, Vol. 15, No. 3, pp. 283-289.*

45. Busacker R.G., *(1965),* Finite Graphics and Networks: An Introduction with Applications. *McGraw-Hill Book CO., New York.*

46. Carrie A.S., *(1974),* Numerical taxonomy applied to group technology and plant layout. *Int. Jour. Prod. Reser., Vol. 11, No. 4, pp. 399-416.*

47. Carrie A.S., *(1977),* The layout of multiproduct lines. *Int. Jo. Pro. Res., No.6, pp.541-557.*

48. Carrie A.S., *(1975),* Graph theory applied to computer aided plant layout. *Proc. 15th International Conference on Machine Tool Design and Research, Macmillan, London.*

49. Carrie A.S. and Mannion J., *(1976),* Layout design and simulation of group cells. *Proc. 16th Inter. Conf. on Machine Tool Design and Research, Macmillan, London, pp. 99-105.*

50. Carrie A.S., Moore J., Roczniak M. and Seppanen J.J., *(1978),* Graph theory and computer aided facilities design. *Omega, Vol. 6, No4, pp. 353-364.*

51. Casotto A. and Romeo F., *(1997),* A parallel simulated annealing algorithm for the placement of macrocells. *IEEE Trans. Comp. Aided Des., Vol.CAD6, No.5, pp. 838-847.*

52. Cerny V., *(1985),* A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Jour. Optimisation Theory and Appl., No. 45, pp. 41-51.*

53. Chandra B. and Tovey C., *(1994),* New results on the old k-opt algorithm for the TSP. *Proceedings 5th ACM-SIAM Symp. on Discrete Algorithms, Philadelphia, pp. 150-159.*

54. Chakravatry A.K. and Naik B., *(1992),* Strategic acquisition of new manufacturing technology. *Int. Jour. Prod. Reser., Vol. 30, No.7, pp. 1575-1601.*

55. Chen W.H. and Srivastava B., *(1994),* Simulated annealing procedures for forming machine cells in GT. *European Journal of Operational Research, No. 75, pp. 100-111.*

56. Christofides N., *(1976),* Worst-case analysis of a new heuristic for the travelling salesman problem. *Report No. 388, GSIA, Carnegie-Mellon University, Pittsburgh, PA.*

57. Chu Y. J. and Liu T. H., *(1965),* On the shortest arborescence of directed graph. *Scientia Sinica, Vol. XIV, No 10, pp. 1396 - 1440.*

58. Croes G., *(1958),* A method for solving TSP. *Oper. Res. 6, pp. 791-812.*

59. Cummings, G.F., *(1980),* Simulation model to compare group technology and functional layout. *Summer Computer Simulation Conference, pp. 626-630.*

60. Darema F. and Kirkpatrick S., *(May 1987),* Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development, Vol 31, pp. 391-402.*

61. Das S., *(1993),* A Facility Layout Method for Flexible Manufacturing Systems. *International Journal of Production Research, Vol. 31, No. 2, pp. 279-297.*

62. De Witte, *(1980),* The use of similarity coefficients in production analysis. *Int. Jour. Prod. Reser., Vol 18, No. 4, pp. 503-514.*

63. Edmonds J., *(1967),* Optimum branching . *Jour. Res. Nat. Bur. Stand., 71B, pp 233 – 240.*

64. Emmons H., *(1992),* STORM Version3: Quantitative modelling for decision support. *Englewood Cliffs, Prentice Hall.*

65. Flood M. M., *(1956),* The travelling salesman problem. *Operations Res. No. 4, pp. 61-75.*

66. Flynn B.B. and Jacobs, F.R., *(1986),* A simulation comparison of GT with traditional job manufacturing. *Int. Jour. Prod. Reser., Vol 24, No. 5, pp. 1171-1192.*

67. Foulds L.R. and Gibbons, P.B., *(1985a),* Facilities layout adjacency determination: An experimental comparison. *Operations Research, Vol. 33, No. 5, pp. 1091-1106.*

68. Foulds L.R. and Giffin, J.W., *(1985b),* A graph-theoretics heuristic for minimising total transport cost in facilities layout. *Inter. Jour. Prod. Res., Vol. 23, No. 6, pp. 1247-1257.*

69. Foulds L.R., *(1983),* Techniques for facility layout: deciding which pairs of activities should be adjacent. *Management Science, Vol. 29, No. 12, pp. 1414-1426.*

70. Foulds L.R. and Robinson D.F., *(1983),* Graph theoretics heuristic for the plant layout problem. *Inter. Jour. Prod. Res., Vol 16, No 1, pp. 27-37.*

71. Foulds L.R. and Robinson D.F., *(1978)* A strategy for solving the plant layout problem. *Operations Research Quarterly, Vol. 27, No. 41, pp. 845-855.*

72. Foulds L.R., *(1994),* Graph theory applications. *Spring-Verlag, New York.*

73. Foulds L.R., Hamacher H. W. and Wilson J. M., *(1998),* Integer programming approach to facilities layout models with forbidden areas. *Operations res. proc., Springer-Verlag*

74. Foulds L.R. and Wilson J.M., *(1995),* Integer programming approach to facilities layout models with forbidden areas. *Res. report 1995-4, University of Waikato, New Zealand.*

75. Francis R.L. and White J.A., *(1974),* Facilities layout and location: an analytical approach. *Eaglewood, Cliffs, N.J., Prentice-Hall.*

76. Francis R.L. McGinnis Jr., Leon F.Jr. and White John A., *(1992),* Facility layout and location: an analytical approach. *Prentice Hall, 2nd edition.*

77. Gabow H.N. and Tarjan R.E., *(Mach 1991),* Faster scaling algorithms for general graph-matching problems. *Jour. Assoc. Comput.,No. 38 pp. 815-853.*

78. Gabow H., *(1973),* Implementations of algorithms for maximum matching on nonbipartite graphs. *Ph.D Dissertation, Dep. of Computer Science, Stanford University.*

79. Gallaghar C.C. and Knight W.A., *(1973),* Group Technology. *London Butterworth.*

80. Garcia-Diaz and Lee H., *(1995),* A network flow approach to solve clustering problems in group technology. *Inter. Jour. Prod. Res., Vol. 31, No.3, pp. 603-612.*

81. Garey M.R. and Johnson D.S., *(1989),* Computers and intractability: a guide to the theory of NP-Completeness. *New York, Freeman W. H.*

82. Gavett J.W. and Plyter N.V., *(1966),* The optimal assignment of facilities to locations by branch and bound. *Operations Research, Vol. 14, pp. 210-232.*

83. Gibbons A., *(1985),* Algorithmic graph theory. *Cambridge University Press.*

84. Giffin J.W., Foulda L. and Cameron D., *(1986),* Drawing a block plan from a REL chart with graph theory and micro-computer. *Comp. Ind. Eng., Vol. 10, No. 2, pp 109-116.*

85. Gillmore P.C. and Gomory R.E., *(1962),* Sequencing a one state-variable machine: a solvable case of the travelling salesman problem. *Operations Res., Vol. 10, pp. 305-313.*

86. Glover F., *(1990),* Future paths for integer programming and links to artificial intelligence. *Computers and Ops. Res., No. 13, pp. 533-549.*

87. Golden B.L. and Skiscim C.C., *(1986),* Using simulated annealing to solve routing and location problems. *Naval Research Logistics Quarterly, Vol. 33, pp. 261- 279.*

88. Goldberg D.E., *(1989),* Genetic algorithms in search, optimisation, and machine learning. *Addison-Wesley, Reading, MA.*

89. Golden B.L. and Stewart W.R., *(1985),* Empirical analysis of heuristics, the travelling salesman problem. *John Wiley and Sons Ltd., New York, pp. 207-214.*

90. Goldratt E.M., *(1990),* The Haystack Syndrome. *Prentice Hall.*

91. Gomory R., *(1961),* Multi-terminal network flows. *SIAM Jour., Vol.9, No.4, pp. 551-570.*

92. Gould R., *(1988),* Graph Theory. *The Benjamin/Cummings Publishing Company, Inc.*

93. Gupta R.M. and Tompkins J.A., *(1982),* An examination of the dynamic behaviour of part families in group technology. *Int. Jour. Prod. Res., Vol. 20, No 1, pp. 73-86.*

94. Hakimi S. L., *(1964),* Optimum locations of switching centers and the absolute centers and medians of a graph. *Oper. Res., Vol 12, pp. 450-465.*

95. Ham I. and Yoshida T., *(1985),* Group technology: application to product management.

96. Harary F., *(1973),* Graphical enumeration. *John Wiley.*

97. Harary F., *(1967),* A seminar on graph theory. *John Wiley.*

98. Harhalakis G., Minis I. and Nagi R., *(1996),* A practical method for design of hybrid type production facilities. *Inter. Jour. Prod. Res., Vol. 34, No. 4, pp. 897-918.*

99. Harhalakis G., *(1994),* Machine cell formation under random demand. *Inter. Jour. Prod. Res., Vol. 32, No.1, pp. 47-64.*

100.Harhalakis G. and Proth J.M., *(1990),* An efficient heuristic in manufacturing cell formation for group technology applications. *Int. Jo. Pro. Res., Vol.28, No.1, pp. 185-198.*

101.Harmonosky C. M. and Tothero G.K., *(1992),* A multifactor plant layout methodology. *Inter. Jour. Prod. Res., Vol. 30, No. 8, pp. 1773-1789.*

102.Hayer N.L. and Wemmerlow U., *(1989),* Cellular manufacturing in the USA industry: a survey of users. *Inter. Jour. Prod. Res., Vol.27, No. 9, pp. 1511-1530.*

103.Heragu S.S., *(1989.a),* Knowledge based approach to machine cell layout. *Computers in Industrial Engineering, Vol. 17, No. 1-4, pp. 37-42.*

104.Heragu S.S. and Kusiak A., *(1990),* Machine layout problem in flexible manufacturing systems. *Operations Research, Vol. 36, No. 2, pp. 258-268.*

105.Heragu S.S. and Kusiak A., *(1991),* Efficient models for the facility layout problem. *European Journal of Operational Research, Vol. 53, pp. 1-13.*

106.Heragu S.S., *(1993),* Group technology and cellular manufacturing. *IEEE Transactions in Systems, Man and Cybernetics, Vol. 24, No. 2, pp. 203-215.*

107.Hillier F.S. and Connors, M.M., *(1966),* Quadratic assignment problem algorithms and the location of indivisible facilities. *Management Sciences, Vol. 13, No. 1, pp. 42-57.*

108.Hillier F.S., *(1963),* Quantitative tools for plant layout analysis. *Journal of Industrial Engineering, Vol. 14, No. 1, pp. 33-40.*

109.Ho Y.C., *(1993),* Two sequence-pattern, matching-based, flow analysis methods for multi flowlines layout design. *Inter. Jour. Prod. Res., Vol. 31, No. 7, pp. 1557-1578.*

110.Holland J.H., *(1975),* Adaptation in natural and artificial systems. *Univ. Michigan Press.*

111.Hopcroft J., *(1974),* Efficient planarity testing. *J. Ass. Comp. Mach., Vol.21, pp.549-568.*

112.Hu T. C., *(1969),* Integer programming and network analysis. *John Wiley.*

113.Immer J.R., *(1950),* Layout planning techniques. *New York, McGraw- Hill.*

114.Ireson W.G., *(1952),* Factory planning and plant layout. *New York, Prentice-Hall.*

115.Jacobs F. R., *(1987),* A layout planning system with multiple criteria and a variable domain representation. *Management Science, Vol. 33, No. 8.*

116.Jajodia S.A., Minis I., Harhalakis G. and Proth, J.M., *(1992),* CLASS-computerised layout solutions using simulated annealing. *Int. Jo. Prod. Res., Vol. 30, No. 1, pp. 95-108.*

117.Jajodia S.A., Minis I. and Harhalakis G., *(1990),* Manufacturing cell formation with multiple, functionally identical machines. *Manuf. Review, Vol. 3, No. 4, pp. 252-261.*

118.Johnson R.V., *(1982),* SPACECRAFT for multi-floor layout planning. *Management science, Vol. 28, No. 4, pp. 407-417.*

119.Johnson D., Aragon C. and Schevon C., *(1989),* Optimisation by simulated annealing: an experimental evaluation: part 1, graph partitioning. *Oper. Res., 37, pp. 865-892.*

120.Johnson D. and Rothberg E., *(1996),* Asymptotic experimental analysis for the Held-Karp travelling salesman bound. *Proc. 7th ACM SIAM Symp. Disc. Algor., Philadelphia,.*

121.Junger M., Reinelt G. and Rinaldi G., *(1994),* The travelling salesman problem. *Report No. 92.113, Angewandte Mathematik und Informatik, Cologne, Germany.*

122.Kaebernick H. and Bazargan-Lari M., *(1996),* An integrated approach to the design of cellular manufacturing. *CIRP Annals, Vol. 45, No. 1, pp. 421-425.*

123.Kaku B.K., Thompson, G.L. and Baybars I., *(1988),* A heuristic method for the multi-story layout problem. *Eur. Jour. Oper. Res., Vol. 37, pp. 384-397.*

124.Kaku B.K., Thompson G.L. and Morton T.E., *(1991),* A hybrid heuristic for the facilities layout problem. *Computers and Operations Research, No. 18, pp. 241-253.*

125.Kandiller L., *(1998),* A cell formation algorithm-Hypergraph approximation – Cut tree. *Euro. Jour. Oper. Res. No. 109, pp. 686-702.*

126.Kaplan A., *(1993),* A probabilistic cost-base due date assignment model for job shops. *Inter. Jour. Prod. Res., Vol 31, No 12, pp. 2817-2834*

127.Kaparthi S. and Suresh N.C., *(1991),* A neural network system for shape based classification and coding of parts. *Inter. Jour. Prod. Res., Vol. 29, No. 9, pp. 1771-1784.*

128.Karisch S.E., *(1998),* Nonlinear approaches for QAP and graph partition problems. *SFB Report 120, University Graz, Austria.*

129.Karisch S.E. and Rendl F., *(1995),* Lower bounds for the quadratic assignment problem via triangle decomposition. *Mathematical programming.*

130.Karp R.M., *(1972),* A simple derivation of Edmond's algorithm for optimum branching. *Networks, Vol 1, No 3, pp 265 – 272.*

131.Kern W., *(1989),* A probabilistic analysis of the switching algorithm for the Euclidean TSP. *Math. Programming, No. 44, pp. 213-219.*

132.Kernighan B.W. and Lin S., *(1970),* An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. Jour., No. 49, pp. 291-307.*

133.King J.R., *(1980 b),* Machine-component group formation in group technology. *OMEGA, No.8, pp. 193-199.*

134.King J.R., *(1980a),* Machine-component grouping in (PFA) group analysis: An approach using rank order clustering algorithm. *Inter. Jour. Prod. Res., Vol. 18, No. 2, pp. 213-232.*

135.King J.R. and Nakornchai V., *(1982),* Machine-component group formulation in group technology: Review and extension. *Inter. Jour. Prod. Res., Vol. 20, No. 2, pp. 117-133.*

136.Kirkpatrick S., Gelatt Jr. C.D. and Vecchi M.P., *(1983),* Optimisation by simulated annealing. *Science, No.220, pp. 671-680.*

137.Koenig D., Gongaware T. and Ham I., *(1981),* Applications of group technology concepts for plant layout and management for a miscellaneous parts shop. *Proceedings of the ninth North America Manufacturing research Conference, MI, USA, pp. 497-502.*

138.Koopmans T.C. and Beckman M.J., *(1957),* Assignment problems and the location of economic activities. *Econometrica, Vol. 25, No. 1, pp. 53-76.*

139.Korte B., *(1988),* Applications of combinatorial optimisation. *Talk at the 13th International Mathematical Programming Symposium, Tokyo.*

140.Kouvelis P., *(1992),* A simulated annealing procedure for single row layout problems in flexible manufacturing systems. *Inter. Jour. Prod. Res., Vol. 30, No. 4, pp. 717-732.*

141.Kouvelis P., Chiang W.C. and Yu G., *(1995),* Optimal algorithms for row layout problems in automated manufacturing systems. *IIE Transactions, No. 27, pp. 99-104.*

142.Kusiak A., *(1989),* Layout of manufacturing cells. *Int. Jo. Pro. Res., No.5, pp. 887-904.*

143.Kusiak A., *(1987),* The facility layout problem. *European Journal of Operational Ressearch, Vol. 29, pp. 229-251, 1987.*

144.Kusiak A., *(1985),* The part families problem in flexible manufacturing systems. *Annals of Operations Research, No.3, pp. 279-300.*

145.Kusiak A. and Heragu, S.S., *(1990),* A machine layout: and optimisation and knowledge-based approach. *Inter. Jour. Prod. Res., Vol. 28, No.4, pp. 615-635.*

146.Kusiak A. and Chung Y., *(1991),* GT/ART: Using neural networks to form machine cells. *Manufacturing Review, No. 4, pp. 293-301.*

147.Kusiak A. and Cho M., *(1992),* Efficient solving of the group technology problem. *Inter. Jour. Prod. Res., Vol.30, No.11, pp. 263-2646.*

148.Kusiak A. and Chow W.S., *(1987),* An efficient cluster identification algorithm. *IEEE Transactions on Systems, Manufacturing and Cybernetics, No.17, pp. 696-699.*

149.Langevin A., Montreuil B. and Riopel D., *(1994),* Spine layout design. *Inter. Jour. Prod. Res., 32, 2, pp. 429-442.*

150.Lawler E.L., *(1963),* The quadratic assignment problem. *Manag. Scie., No.9, pp. 586-599.*

151.Lawler E.L., *(1976),* Combinatorial optimisation. *Networks and matroids, New York.*

152.Lawler E.L., Lenstra J.K., Rinnooy Kan A.H. and Shmoys D.B., *(1979),* The travelling salesman problem: a guided tour of combinatorial optimisation. *Wiley, New York.*

153.Lee R.C. and Moore J.M., *(1967),* CORELAP-Computerised Relationship Layout Planning. *Journal of Industrial Engineering, Vol. 18, No. 3, pp. 195-200.*

154.Lenstra J.K., *(1995),* Clustering a data array and the travelling-salesman problem. *Operations Research, No.22, pp. 413-414.*

155.Levin P.H., *(1964),* Use of graphs to decide the optimum layout of building, *Journal of Architects, Vol. 7, pp. 809-815.*

156.Liggett, R.S. and Mitchell W.J., *(1981),* Optimal space planning in practice. *Computer Aided Design, Vol. 13, pp. 277-288.*

157.Lin T.L. and Kumar K.R., *(1996),* A heuristic based procedure for the weighted production cell formation problem. *IIE transactions, 28, pp. 579-589.*

158.Lin S. and Kernighan B.W., *(1973),* An effective heuristic algorithm for the travelling salesman problem. *Operations Research, No.21, pp. 498-516.*

159.Lin S. and Kernighan B.W., *(1990),* An efficient heuristic for graphs partitioning. *Bell Syst. Tech. Jour., No. 69, pp. 291-307.*

160.Love R.F. and Wong J.W., *(1976),* Solving quadratic assignment problem with rectilinear distances and integer programming. *Naval Research Logistics, Vol.23, pp. 623-627.*

161.Love R., *(1973),* A multi – facility minimax location method for Euclidean distances. *Inter. Jour. Prod. Reser., Vol. 11, No. 1, pp. 37 – 45.*

162.McAuley J., *(1972),* Machine grouping for efficient production. *The Production Engineer, No. 51, pp. 53-57.*

163.McCormick W.T., Schweitzer P.J. and White T.W., *(1972),* Problem decomposition and data reorganisation by a clustering technique. *Operations Research, No. 22, pp. 993-1008.*

164.Metropolis N., Rosenbluth A., Teller A. and Teller E., *(1953),* Equation of state calculations by fast computing machines. *Jour. Chem. Phys., Vol. 21 pp. 1087-1099.*

165.Michalewicz Z., *(1995)*, Genetic algorithms+data structures. *Springer.*

166.Michalewicz, Z., *(1996)*, Genetic algorithms + data structures = evolution programs. *Second, Extended Edition. Heidelberg, New York.*

167.Mitrofanov S.P., *(1959)*, Scientific principles of group technology. *Leningrad (St. Petesburg), Maschinostroyenic.*

168.Montreuil B., *(1989)*, A modelling framework for integrating layout design and flow network design. *Proc. Material Handling Research Colloquium, Kentucky, pp. 43-58.*

169.Montreuil B. and Ratliff H.D., *(1989)*, Utilising cut trees as design skeletons for facility layout. *IIE Transactions, Vol. 21, No. 2, pp. 136-143.*

170.Montreuil B. Drolett J.R. and Moodie C.L., *(1990)*, Virtual cellular manufacturing layout planning. *Proc. 1990 Inter. Ind. Eng. Confer., San Francisco, CA, pp. 236-241.*

171.Montreuil B., Venkatadri U. and Ratliff H. D., *(1993)*, Generating a layout from a design skeleton. *IIE transactions, Vol. 25, No. 1, pp. 3-5.*

172.Montreuil B. and Tanchoco, J.M.A., *(1994)*, Material flow systems in manufacturing. *Published by Chapman and Hall, Chapter 3, pp. 75-101.*

173.Montreuil B. and Lefrancois P., *(1996)*, Organising factories as responsibility networks. *In Progress in Material Handling Research, Ed. Graves R.J., pp. 375-411.*

174.Moodie C.L. and Warren G.H., *(1994)*, Cell design strategies for efficient material handling. *In Material Flow Systems in Manufacturing, Ed. Tanchoco J.A, pp. 76-101.*

175.Moon Y.B., *(1990)*, Forming part-machine families for cellular manufacturing: a neural network approach. *Inter. Jour. of adv. Manufac. Technology, Vol. 5, No. 4, pp. 277-291.*

176.More J., *(1976)*, Facilities design with graph theory and string. *Omega, No.2, pp.193-203.*

177.Muther R., *(1973)*, Systematic layout planning. *Cahers Books, Boston.*

178.Muther R., *(1961)*, Systematic layout planning. *Industrial Education Institute, Boston.*

179.Nahar S., Sahni S. and Shragowitz E., *(1984)*, Experiments with simulated annealing. *Report No. 84-36, Computer Science Department, University of Minnesota.*

180.Nisanci H.I. and Sury R.J., *(1981)*, An application of GT concepts in shoe manufacturing. *Inter. Jour. Prod. Res., Vol. 19, pp. 267-275.*

181.Nozari A. and Enscore E.E., *(1981)*, Computerised facility layout with graph theory. *Computers Industrial Engineering, Vol. 5, No. 3, pp. 183-193.*

182.Nugent C.E. and Vollman T.E., *(1968)*, An experimental comparation of techniques for the assignment of facilities to locations. *Operation Research, Vol. 16, pp. 150-173.*

183.Ong H.L. and Moore J.B., *(1984)*, Worst-case analysis of two travelling salesman heuristics. *Operations Res. Letter, No. 2, pp. 273-277.*

184.Otten R. and Van Ginneken L., *(1989),* The annealing algorithm. *Kluwer Academic.*

185.Pardalos P.M., *(1997),* Network optimisation. *Springer-Verlag Berlin.*

186.Press W. and Teukolsky W., *(1990),* Numerical recipes. *pp. 326-334, Cambridge Press.*

187.Primrose P.L. and Leonard R., *(1986),* The financial evaluation and economic application of advanced manufacturing technology. *Proc. Inst. of Mech. Eng., Vol.20, No.1, pp.27-31.*

188.Proth J.M. and Vernadat F., *(1991),* COALA: A new manufacturing layout approach. *Proceedings of the ASME Winter Meeting, Edited by Black J.T., (ASME), pp. 47-70.*

189.Rajagoplan R. and Batra J.L., *(1975),* Design of cellular production systems: A graph theoretic approach. *Inter. Jour. Prod. Res., Vol.13, No.6, pp. 567-579.*

190.Rajamani D. and Singh N., *(1992),* A model for cell formation in manufacturing systems with sequence dependence. *Inter. Jour. Prod. Res., Vol.30, No.6, pp. 1227-1235.*

191.Reinelt G., *(1994),* The travelling salesman problem: computational solutions for tsp applications. *Lecture Notes in Computer Science 840, Springer-Verlag, Berlin, No. 102.*

192.Robinson D. and Ducstein L., *(1986),* Polyhedral dynamics as a tool for machine-part group formation. *Inter. Jour. Prod. Res., Vol. 24, No. 5, pp. 1255-1266.*

193.Ronen B. and Spector Y., *(1992),* Managing system constraints: a cost/utilisation approach. *Inter. Jour. Prod. Res., Vol. 30, No. 9, pp. 2045-2961.*

194.Ronen B. and Starr Y., *(1990),* Service organisation costing: a synchronised manufacturing approach. *Industrial Management, Sept.-Octob., pp. 24-26.*

195.Rosenblantt M.J. and Kropp D.H., *(1992),* The single period stochastic plant layout problem. *IIE* Transactions, Vol. 24, No. 2, pp. 169-176.

196.Rosenblantt M.J., *(1989),* A heuristic algorithm for the quadratic assignment formulation to the plant layout problem. *Inter. Jour. Prod. Res., Vol. 27, pp. 293-308.*

197.Rosenkrantz J., Stearns R. E. and Lewis P. M., *(1977),* An analysis of several heuristics for the travelling salesman problem. *SIAM Jour. Comput., No. 6, pp. 563-581.*

198.Sahni S. and Gonzales T., *(1976),* P-complete approximation problem. *Journal of Associated computing Machinery, Vol. 23, No. 3, pp. 555-565.*

199.Sarker B. R. and Li Z., *(1998),* Scheduling virtual cells in cellular manufacturing systems. *Industrial Engineering Research Conference, May 9-10, 1998, Banff, Canada.*

200.Sassani F., *(1990),* A simulation study on performance improvement of group technology cells. *Inter. Jour. Prod. Res., Vol. 28, No. 2, pp. 293-300.*

201.Seifodini H., *(1990),* A probabilistic model for machine cell formation. *Journal of Manufacturing Systems, Vol. 9, No. 1, pp. 69-75.*

202.Sefoddini H. and Wolfe P.M., *(1986),* Application of the similarity coefficient method in group technology. *IIE Transactions, No.18, pp. 271-277.*

203.Scriabin M. and Vergin R.C., *(1985),* A cluster analysis approach to facility layout. *Management Science, Vol. 31, No. 1, pp. 33-49.*

204.Seehof J.M. and Evans W.O., *(1965),* Automated layout design program. *The journal of Industrial Engineering, Vol. 18, No. 2, pp. 690-695.*

205.Shafer S.M. and Rogers D.F., *(1991),* A goal programming approach to the cell formation problem. *Journal of Operations Management.*

206.Shunk D. and Reed R., *(1975),* An analytical approach to measure the effects of group technology. *Proc. Third Inter. Conf., pp. 627-634, New York, NY: Chapman & Hall.*

207.Smith A., *(1715),* The wealth of nations.

208.Sirnivasan D.G., Narendran T. and Mahadevan B., *(1990),* An assignment model for the part-families problem in group technology. *Inter. Jour. Prod. Res., Vol. 28, pp.145-152.*

209.Simpson J.A. and Albus J.S., *(1982),* The automated manufacturing research facility of the National Berau of Standards. *Jour. of Man. Systems, Vol. 1, No. 1, pp. 17-32.*

210.Suresh N.C. and Meredith J.R., *(1985),* Justifying multi-machine systems: an integrated strategic approach. *Journal of Manufacturing Systems, Vol. 4, No. 2, pp. 117-134.*

211.Sule D.R., *(1991),* Machine capacity planning in group technology. *Inter. Jour. Prod. Res., Vol. 29, No. 9, pp. 1909-1922.*

212.Sule D. R., *(1994),* Manufacturing facilities location, planning, and design. *PWS, Boston.*

213.Tam K.Y., *(1992),* A simulated annealing algorithm for allocating space to manufacturing cells. *Inter. Jour. Prod. Res., Vol. 30, No. 1, pp. 63-87.*

214.Tarjan E., *(1977),* Finding optimum branchings. *Networks, No. 7, pp. 25 – 35.*

215.Tarjan E., *(1983),* Data structures and network algorithms. *John Wiley.*

216.Tarjan E., *(1986),* Algorithms for maximum network flow. *Mathematical Programming Study, No. 26, 1 - 11.*

217.Tilsley R., Lewis, F.A. and Galloway D.F., *(1977),* Flexible cell production systems: A realistic approach. *CIRP Annals, Vol. 25, No. 1, pp. 269-271.*

218.Tompkins J.A. and White J.A., *(1984),* Facilities planning. *John Wiley, New York.*

219.Vakharia A.J. and Kaku B.K., *(1993),* An investigation of the impact of demand changes on a CM system. *Technical Report No. MS/S93-001, Univ. Maryland.*

220.Vakharia A.J., *(1990),* Designing a cellular manufacturing systems: A materials flow approach based on operation sequences. *IIE Trans., Vol. 22, No. 1, pp. 84-97.*

221.Vannelli A. and Kumar K.R., *(1986),* A method for finding minimal bottleneck cells for grouping part-machine families. *Inter. Jour. Prod. Res., Vol. 24, No. 2, pp. 387-400.*

222.Venkatadri U., Rardin R. and Montreuil B., *(1997).* A design methodology for fractal layout organisation. *IIE Transactions, 29, pp. 911-924.*

223.Venugopal V. and Narendran T.T., *(1992),* Neural network models for design retrieval in manufacturing systems. *Computers in Industry, 20, pp. 11-23.*

224.Vidal (Ed.) R.V., *(1993),* Applied Simulated Annealing. *No. 396, Springer, Heidelberg.*

225.Volenant T. and Jonker R., *(1987),* On some generalisations of the travelling-salesman problem. *Journal of Operations Ressearch, Vol.38, No. 11, pp. 1073-1079.*

226.Waghodekar P.H. and Sahu S., *(1984),* Machine-component cell formation in group technology: MACE. *Inter. Jour. Prod. Res., Vol. 22, No.6, pp.* 937-948.

227.Wang T, Lin H. and Wuan K., *(1998),* Improved simulated annealing for facility, layout problems in cellular manufacturing Systems. *Comp. Ind. Eng. Vol. 34, No. 2, pp. 309-319.*

228.Webster D.B. and Tyberghein M.B., *(1980),* Measuring flexibility of jobshop layouts. *Inter. Jour. Prod. Res., Vol. 18, No. 1, pp. 21-29.*

229.Wemerlow U., *(1988),* Production planning and control procedures for cellular manufacturing systems. *APICS.*

230.Wemmerlow U. and Hayer N.L., *(1987),* Research issues in cellular manufacturing. *Inter. Jour. Prod. Res., Vol. 25, pp. 413-431.*

231.Wilhelm M.R. and Ward T.L., *(1990),* Solving quadratic assignment problems by simulated annealing. *IIE Transactions, No.19, pp. 107-119.*

232.Zoller K. and Adendorff K., *(1972)* Layout planning by computer simulation. *AIIE Transactions, Vol. 4, No. 2, pp. 116-125.*

Publication Originating from this Thesis

1. Bajic, M. M. and Baines, K., "Modelling of Virtual cells for dynamic manufacturing environments". Accepted for the 16th International conference on computer aided production engineering in 2000 (CAPE 2000), Edinburgh UK.

2. Bajic M. M., Cresswell C., Baines K. "A cellular manufacturing approach towards virtual manufacturing", World Congress Manufacturing Technology Towards 2000, Cairns 1997, Australia

3. Bajic, M. M. and Baines, K., "Towards virtual manufacturing-recent developments in cellular manufacturing", AUTOFACT Asia'96, International Conference, Singapore, July 1996, SME.

4. Bajic M. M. "An analytical approach for the design of VCMs". Transfer Report, The University of Adelaide, 1996

APPENDIX A

A.1. Graph Theory Review

From Chapter 4, which mentions network analysis and graph theory texts, this appendix section extends the definition and terminology used in the mathematical modelling of the *DCMS's* layout introduced in the main text.

The outdegree of a node v_i, denoted by $d_o(v_i)$, is the number of arcs which have the node v_i as their tail node. Similarly, the indegree of the node v_i, denoted by $d_i(v_i)$, is the number of arcs which have the node v_i as their head node. The sum of the indegrees of all nodes of a digraph is equal to the sum of the outdegrees and its value is the number of arcs, ie. $\Sigma\, d_o(v_i)\ \forall i = \Sigma\, d_i(v_i)\ \forall i = m$. A pendant node in a graph is a node v_k such that $d_o(v_k) + d_i(v_k) = 1$, i.e. the total number of arcs incident to and from the node is one. This definition does not apply to the trivial case of a graph with a single node because the only arc possible will correspond to a self-loop.

A chain between nodes v_1 and v_p is a collection of nodes $v_1, v_2, \ldots, v_p$ and $(p-1)$ arcs, one for each pair of nodes, v_1v_2 or v_2v_1, v_2v_3 or v_3v_2, …, $v_{p-1}\, v_{pm}$ or $v_p\, v_{p-1}$ $(n \geq p)$. A path between nodes $v_1, v_2, \ldots, v_p$ and $(p-1)$ arcs, one for each pair of nodes and the unidirectional sequence of arc is the tail node of its successor arc. The length of a path p_{ij}, denoted by $w(p_{ij})$, is the sum of the arc weights appearing in it, ie. $w(p_{ij}) = \Sigma\, c_{ij}\ \forall\ a_{ij} \in p_{ij}$. Node v_j is reachable from node v_i if there is a path from v_i to v_j. A sequence differs from a path in that nodes and arcs can repeat in sequence, where as they must be unique in a path. Hence, a sequence from node v_i to v_j, plus one more circuit or cycle, starting or ending on any of the nodes on the path, is either a path (no machines are visited more than once) or a sequence (one or more machines are visited two or more times) involving one or more machines. A circuit is a path containing arcs $a_1 \ldots a_q$ where the tail node of a_1 coincides with the head node of a_q. A cycle, which can contain arcs, is a chain containing arcs $a_1 \ldots a_q$, where the tail node of a_1 coincides with the head node of a_q.

A partial graph of $D(V,A)$, denoted by $D_p(V_p = V,\ A_p \subset A)$, is a digraph with the same number of nodes as $D(V,A)$ but with only a subset of the arcs of the original graph. *A* subgraph of $D(V, A)$, denoted by $D_s(V_s \subset V,\ A_s \subset A)$, is a digraph which has only a subset of nodes of the original digraph, but contains all the arcs whose initial and final nodes are both within this subset of nodes, ie. $a_{ij} \in A_s$ $[(v_i \in v_s)$ and $(v_j \in v_s)]$.

A digraph *D(V, A)* is strongly connected if all pairs of nodes v_i and v_j are mutually reachable (a path exists from v_i to v_j, as well as from v_j to v_i). The digraph is unilateral if there exists at least one path between any pair of nodes, either from v_i to v_j, or v_j to v_i. The digraph is weak if all pairs of nodes are connected by a chain, ie. in the corresponding undirected graph, a set of edges links each pair of nodes. A digraph is disconnected if it is not weak. Thus each edge of a digraph belongs to some weakly connected component but does not necessarily belong to a strongly connected component. A strong digraph is unilateral and a unilateral graph is weak, but the reverse is not true.

A.2. Arcs Classification in the *WDRT*

In this section a further description of *WDRT* arcs classification is presented. The machine grouping, intracell flowline layout and intercell layout planning problems can be integrated through the structure of the Maximal Weighted Directed Spanning Tree (*MWDRST*). This suits the in-sequence, bypass and backtrack classification (for intracell flows) and crisscrossing classification (for intercell flows) of flows in a cellular system. Hence, the *MWDRST* was preferred over several undirected graph structures such as cut trees [Montreull 1989], maximal spanning trees [Carrie 1975, Heragu 1989 and 1990], minimal spanning tree [Srinivasan 1994] and maximal planar graphs [Carrie 1978, Foulds 1985] used to solve the more general problem of facilities (or only intercell) layout design. Based on the above discussion the complete set of arcs *A* in *D(V, A)* other than those in *T*, can be decomposed into two sets:

$$A_t = \{a_{ij} \in T\},\ and\ \ A_{\bar{T}} = \{a_{ij} \in \bar{T}\},$$

ie. those arcs in the co-*WDRT* $\bar{T}$, where $(A_T \cup A_{\bar{T}} = A)$ and $(A_T \cap A_{\bar{T}} = \Phi)$. The residual digraph $\bar{T}$, *or* $D(V_{\bar{T}}, A_{\bar{T}})$, can be called a co-*WDRT* of the original digraph *D(V, A)*. Since *D(V, A)* has n vertices, then $\bar{T}$ has $(n^2 - 2n + 1) \geq (m - n - 1) = (\ |A| - n + 1)$ arcs excluding all self loops. Any arc of $\bar{T}$ is called a chord of *T*. The chords, which create the problem of intercell flows and machine duplication within or between cells, have been ignored in the flowline design and simulation models of Hillier [1975], Carrie [1976] and Aneke [1984 and 1986]. Also their heuristics ignore the problem of flow line design to minimise material handling costs.

A.3. Problem Complexity and *NP* - Problems

This section elaborates further the mathematical model complexity, focusing on the non-deterministic polynomial time complete problems, normally abbreviated to *NP*-problems. Garey's [1989] text was the primary reference for all the *NP*-complete problems found to be analogous to the machine grouping and layout design problems discussed in this section.

Cluster analysis is suitable mainly for machine grouping and detection of shared machines (see p281). The Bandwidth problem (see p200) assumes a bi-directional flow line for all the machines in the travel chart. It seeks to minimise the maximum length (travel distance) for any flow arc. Hence, it would fail to find machine groups, a suitable flow structure for each cell, and the intercell layout for the shop. For similar reasons, the Optimal Linear Arrangement problem (see p200), which seeks to minimise the total length of all arcs in the original (undirected) graph when all nodes are arranged in a sequence, is also unsuitable. However, the Directed Bandwidth (see p200) and Directed Optimal Arrangement (see p200) problems are analogous to the design of unidirectional flow line. They can also be used for machine grouping by ordering the machines in sequence such that all the high flow volume clusters form along the diagonal of the travel chart. However, they lack any ability to locate the individual machines and group them simultaneously. These two problems are related to the Directed Hamiltonian Path (see p199) and the *TSP* (see p211) if the travel chart is converted to a strongly connected digraph by a suitable transformation of its entities.

The problem encountered in this literature is an enhancement of the traditional *QAP* (see p218), since the configuration of the shop and possible locations of the individual machines are to be determined in parallel. The *QAP* ignores machine grouping and layout based on flow directions. Here the structures capture the machine grouping and intercell layout problems accurately, but cannot yield a flow line layout for each group. If travel distances are to be minimised on the tree extracted from the travel chart digraph, it becomes equivalent to the Rooted Tree Arrangement problem (see p201). If the number of paths in the tree have to be a prespecified wirh value *K*, as when the number of cells desired is fixed, then this problem is analogous to that of the Maximum Leaf Spanning Tree problem (see p206). When it is desired that the flow lines be allowed to take nonlinear shapes, ie. when machines in a path can be located in an Euclidean plane, the layout problem is similar to that of determining a Geometric Capacitated Spanning Tree (see p207). The spanning tree, which minimises the weighted moments of the flow distances and chain lengths, becomes equivalent to the Optimum Communication Spanning Tree problem (see p207). Since the paths in such trees lack direction, the concept of a Maximal Weighted Directed Rooted Spanning Tree *(MWDRST)* was adopted. The choice of arcs, which makes the *WDRT* maximal, will automatically minimise the path lengths for all pairs of nodes.

Clearly, the aforementioned problems and solutions have serious shortcomings. Thus, it was the Multiple-Choice Branching problem (see p208), which suggested the solution strategy for the machine grouping and layout design problems relevant to the proposed research program. This strategy states that:

> *"Given a directed graph G(V, A), a weight $w(a) \in z^+$, a partition of A into disjoint sets A_1, A_2, ..., A_m and a positive integer K, it is possible to find a subset $A' \in A$ with $\sum w(a) \geq K$ $\forall$ $a \in A'$ such that no two arcs in A' enter the same vertex, A' contains no cycles and A' contains at most one arc from each of the A_i, where $m \geq i \geq 1$. If all A_i have $|A| = 1$, the problem becomes simply that of finding a "maximum weight branching"...that can be solved in polynomial time (Tarjan 1977)"...that "in a strongly connected graph, a maximum weight branching can be viewed as a maximum directed spanning tree."*

Some simplifications needed to be made in order to avoid starting with a *NP*-complete problem of size equal to the number of machines. The *MWDRST* was used to identify the machine groups and flow line layouts for the machines in each of its paths. By assuming a strict linear orientation for each path, the complexity of simultaneously solving a two-dimensional Euclidean location problem and a *QAP* was avoided. Hence, the problem no longer remained *NP*-complete with respect to the original number of nodes of the digraph. The *NP*-completeness was reduced to a permutation of the branches of the *WDRT*, and it will be shown in later sections of this chapter that the number of pendant nodes in the *WDRT* is the size of this permutation problem.

Theoretically, since the *WDRT* must have at least two pendant nodes, there will always be a reduction in the number of nodes for the permutation problem, and thus was solved as a one-dimensional *QAP*. A benefit of this approach was that *QAP* worked with a feasible shop layout. Traditionally, *QAP* assumes that the layout for the machines already has a rectangular array form. Only the assignments of machines to each of the locations need to be determined.

A.4. Mathematical Formulation of the Approximate Layout Problem

The initial formulation of the machine grouping and approximate layout design problem (stage I and stage II) which is given in Chapter 4, with no machine duplication to minimise intracell *($a_{ij} \in F \cup B$)* and intercell *($a_{ij} \in C$)* flow distances, can then be explained in more detail in the next sections.

A.4.1. Stage I

From Chapter 4 the definition from *Equation A.1* and the constraints in *Equations A.2, A.3* and *A.4* constitute stage I, which is a linear programming model for obtaining the *T* tree (maximum weighted directed rooted spanning tree). The *MWDRST* is obtained using the Chu [1965] and the improved Tarjan [1977] optimum branching algorithm (from original graphical procedure, converted - coded and executed in *MATLAB*, Appendix D) - and described in integer programming form thus:

$$Maximise \quad \sum_{j \neq R} \sum_{i \neq S} f_{ij} w(p_{ij}) x_{ij} \qquad \forall \; a_{ij} \in T \qquad \text{Equation A. 1}$$

subject to:

$$\sum_{\forall i} x_{ij} \quad (or\ d_i(V_j)) = 1 \; \forall \; j \neq R \qquad \text{Equation A. 2}$$
$$= 0 \; if \;\; j = R$$

and

$$\sum_{\forall j} x_{ij} \quad (or\ d_o(V_i)) \geq 0 \; \forall \; i \neq S \qquad \text{Equation A. 3}$$
$$= 0 \; if \;\; i = S$$

and

$$x_{ij} = 1 \; if \; a_{ij} \in T \qquad \text{Equation A. 4}$$
$$= 0 \;\; otherwise$$

where:

T - *MWDRST* generated by the (linear) programming model in the first stage, (also denoted as *$D(V_t, A_t)$*),

R, S - common root (raw material store) and sink (finished goods store) nodes of the digraph $D(V,A)$ (travel chart) occurring in the operation sequences of all parts,

V_t - set of vertices in T tree,

A_t - set of arcs in T tree,

Q_k - batch quantity for part k,

S_k - operation sequence of part k, represented as $(R, 1, 2, 3, \ldots.., n\text{-}1, n, S)$, where i is the machine required for the i-*th* operation on the part,

$f_{ij} = \Sigma Q_k \ \forall S_k \mid a_{ij} \in S_k \ i \neq S; j \neq R$ - flow from machine i to machine j (the sum of the batch quantities of all parts whose operation sequences contain machines i and j),

p_{ij} - path in T tree connecting machine i to machine j,

a_{ij} - directed arc from node i to node j,

$d_i(V_j)$ - in degree of node V_j,

$d_O(V_i)$ - out degree of node V_i,

X_{ij} - binary variable to represent the membership of arc a_{ij} in T,

$w(p_{ij})$ $= 1$ if - $a_k \in T$ weight (length) of the path p_{ij},

$= k$ if $a_k \in F$ ie. node j is reachable from node i by a path p_{ij}^k containing k arcs,

$= ck$ if $a_k \in A$ or $a_k \in B$ ie. the digraph D contains an arcs a_{ji} which, together with the arcs in the path p_{ij}^k in T, creates a circuit, (the constant C reflects the increase in flow time for backward arcs compared to the forward arcs),

$= w(p_{ij}) + \mid COL(P_i) - COL(P_j) \mid$ where $COL(P_i)$ and $COL(P_j)$ are the columns to which paths P_i and P_j, respectively, are assigned if T is embedded in a rectilinear grid. (Distances being rectilinear $w(p_{ij})$ is the vertical travel distances fixed by the solution for T and $\mid COL(P_i) - COL(P_j) \mid$ is the horizontal between the two paths.),

P_i, P_j - paths in T tree rooted at R containing nodes i and j, respectively,

P - width of T tree, and

- the number of pendant nodes in T tree

If the digraph contains strong components (good solution), as shown in Appendix A.6, then disjoint subtrees will be contained initially when T is drawn. Additional constraints (shown in *Equation A.5*), assuming each group of nodes in a subtree to be a single node, needs to be added to the model to connect these subtrees.

$$\sum_{j \neq R} \sum_{i \neq S} f_{ij} w(p_{ij})(1 - x_{ij}) \qquad \forall \ a_{ij} \in F \cup B \qquad \textit{Equation A. 5}$$

The x_{ij} s naturally have values $0/1$ because the constraints are of the form $B \geq AX$, $X \geq 0$ and B is integer-valued with an upper bound of *1*. Each column of the *A* matrix will contain only two non-zero entries of *1* and *–1* for the head and tail nodes respectively, of the arc that it represents. The unimodularity of the *A* matrix gives an all-integer optimal solution for the Stage I linear program. This will indicate whether the arc a_{ij} is contained in *T*, or not. The flow distances $w(p_{ij})$, is equal to *1* for any $a_{ij} \in T$ (consecutive machines in each path are adjacent to each other). *Equation A.5* can thus be rewritten as:

$$\sum_{j \neq R} \sum_{i \neq S} k(f_{ij} + cf_{ij}) \quad \text{where } a_{ij} \in F \text{ and } a_{ij} \in B \text{ for arcs in } F \cup B.$$

If there is a path p_{ij}^{k} in *T* connecting nodes *i* and *j*, then $w(p_{ij})$ will be the distance travelled by the flow in the arc $a_{ij} \in F$, and if $a_{ij} \in B$ backtracking on the same path p_{ij} in *T* is implied. This results in a higher flow penalty, given by c_k. Since a pure flowline layout is assumed for each path in *T*, the flow distances for the arcs in $F \cup B$ are automatically fixed. Hence, the linear model from stage I yields a layout which minimises intracell travel distances, assuming a flowline layout for each cell. Every flowline originates from a single root, unlike a *MST* (maximal spanning tree) whose every pendant node can be a root.

$$\sum_{j \neq R} \sum_{i \neq S} f_{ij} \left[w(p_{ij}) + \left| COL(P_i) - COL(P_j) \right| \right] \left(1 - x_{ij} \right) \qquad \forall \; a_{ij} \in C \qquad \textit{Equation A. 6}$$

Every path reaching from the root to a pedant node is connected to one or more other paths by chords which corresponds to crisscrossing arcs only. Stage II was developed because of minimisation of the intercell flow distances corresponding to the crisscrossing arcs. The *WDRT* is a planar graph because it can be drawn in the plane with no edges crossing. However, the different branches can be pivoted about the root node and any other branching nodes without changing the individual paths (shown in *Equation A.6*). If the vertical distances $w(p_{ij})$ in *Equation A.6* are ignored, it can be rewritten as:

$$\sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \sum_{\forall k \in P_i \notin P_j} \sum_{\forall l \notin P_i \in P_j} f_{kl} \left| COL(P_k) - COL(P_l) \right| \qquad \textit{Equation A. 7}$$

This is similar to the objective function for the *NP*-complete *QAP* [Francis 1974] restricted to a linear permutation. It seeks to rearrange the initial *WDRT* from stage 1 such that total flow distances for the crisscrossing arcs are minimised. Each path becomes a node in the sequence describing the left-to-right order of the paths in *T*.

Traditionally, an exact solution of the *QAP* is computationally infeasible for fifteen or more machines [Francis 1974], necessitating the use of pairwise-interchange heuristics. Since the original *n*-node *WDRT* is now being treated as a linear ordering of *p* nodes $(p < n)$, the resulting *QAP* could still be solved optimally, using integer programming. For values of *p* greater than 15, then possibly a few flowlines could merge, dependent on whether these paths have several branching nodes in common, or are node disjointed.

Traditionally, each machine is a node in the *QAP*, whereas now each set of machines contained in a path in *T* is treated as a node. Hence, the *WDRT* reduces the dimensionality of the original *QAP* problem and allows the permutation of its branches to be solved as an Optimal Linear Arrangement problem.

Crisscrossing arcs, connecting nonadjacent paths in T after stage II, are the arcs definitely requiring machine sharing to eliminate their intercell flows. Ignoring the lengths of the arcs in $F \cup B$ above, then a simplified explanation is possible. The concept of Tree *(T)* permutation is explained in the Appendix A8.

A.4.2. Stage II

This section extends in more detail a mathematical model description given in Chapter 4, which finds the optimal permutation of the *MWDRST* branches to minimise intercell flow distances. The term $|COL(P_i) - COL(P_j)|$ from the previous section is replaced with the term $(R_{ij} + L_{ij})$, where R_{ij}*'s* and L_{ij}*'s* yield $(n^2 - n)$ variables and the A_{ik}*'s* are another (n^2) variables. Introducing X_i*'s* dummy variables which describes the positioning in the linear arrangement to which a path is assigned gives:

$$\textit{Minimise} \qquad \sum_{i=1}^{p-1}\sum_{j=i+1}^{p}\{F_{ij} + \sum_{k\in P_i \cap P_j} W_k\}(R_{ij} + L_{ij}) \qquad \textit{Equation A. 8}$$

subject to:

$$R_{ij} - L_{ij} = X_i - L_j \qquad i = 1,\ldots., p\text{-}1;\ j = 1,\ldots., p\text{-}1;\ i \neq j \qquad \textit{Equation A. 9}$$

$$X_i = \sum_{k=1}^{p} kA_{ik} \qquad i = 1,\ldots., p \qquad \textit{Equation A. 10}$$

$$\sum_{k=1}^{p} A_{ik} = 1 \qquad i = 1,\ldots., p \qquad \textit{Equation A. 11}$$

$$\sum_{i=1}^{p} A_{ik} = 1 \qquad k = 1,\ldots., p \qquad \textit{Equation A. 12}$$

$$A_{ik} = 0,\ 1 \qquad \forall a_{ik} \mid i \neq k;\ i = 1,\quad , p;\ k = 1,\ldots., p$$

$$R_{ij} \geq 0\ \forall(i, j) \qquad L_{ij} \geq 0\ \forall(i, j)$$

where:

P - number of paths corresponding to the pendant nodes in the T tree,

F_{ij} - $\sum_{l}\sum_{k} f_{kl}$ $\forall k \notin P_i \in P_j,\ \forall l \in P_i \notin P_j,\ \alpha_{kl} \in C$; sum of flow volumes of arcs that connect paths P_i and P_j (with either the head or tail node of each arc lying on P_i or P_j),

W_k - flow weight assigned to a machine common to paths P_i and P_j,

R_{ij} - column difference between P_i and P_j if P_i is to the right of P_j,

= *0*, otherwise,

L_{ij} - column difference between P_i and P_j if P_i is to the left of P_j,

= *0*, otherwise,

X_i - position in the linear sequence to which P_i is assigned,

A_{ik} = *1* if P_i is assigned to position k in the optimal sequence of pendant nodes, and

= *0* , otherwise.

Since the intercell flow matrix is symmetric, the value of the objective function could also have been multiplied by *0.5*. In any basic feasible solution, for the stage II problem, either R_{ij} or L_{ij} will be zero. If the constraints in *Equation A.9* are written in matrix form, then the column of the matrix for some R_{ij} will be -1 times the column for L_{ij}. This will make two columns linearly dependent whereas a base consists of linearly independent vectors [Francis 1974]. In a linear arrangement of nodes, a node i can either be to the left of node j, or vice versa. Constraints in *Equation A.11* and *Equation A.12* force each path to be uniquely assigned to a particular position in the optimal linear arrangement. The constraints in *Equation A.10* assign a path to any one position of the total of p (number of pendant nodes) available. *Equation A.9* is the most important and the x_{is} from *Equation A.10* are directly absorbed into these constraints. The important part of stage II is the assignment of the w_{ks} as flow weights to branching nodes common to two or more paths in the *MWDRST*. As shown in the next section several conditions need to be fulfilled, which can be expressed as follows:

- The weight w_2 assigned to the branching nodes attached to the root node, must be greater than $(p - 1)\ max\ (F_{ij})$, and
- The w_k branching nodes values at *levels 3,4,* etc., must be greater than or equal to w_2.

Stage II was felt necessary because the progressively increasing weights used successfully in the examples solved later were obtained iteratively from trial runs for each example. As illustrated in later examples, during the permutation of the branches of T, two critical tasks needed to be performed. The first task is necessary, although it does not influence intercell flow distances. It ensures that if a pair of paths shares paths, which share the machines at the branching nodes, then such a shared path will always permute as a group, even if the machines are at two or more levels. The second task is to permute the branches without violating the restrictions imposed during the first task. This task minimises intercell flow distances without allowing any machine sharing.

The successful use of this restricted permutation strategy has been shown in the second stage of the experimental study. However, in the worst case, where a one dimensional optimal linear arrangement becomes a two dimensional *QAP,* then this is a *NP*- problem. Traditionally, an exact solution of the *QAP* is computationally infeasible for fifteen or more machines [Francis 1974], necessitating the use of pairwise-interchange heuristics. Recently, Karisch [1998] compared results from the steepest descent pairwise interchange heuristic (*SDPI*) with respect to *EA* and *SA* methods, when applied to the *QAP* problem. Here the outcome was that the *SDPI* still gave better performance then the other two methods. Following Karisch's paper result, the *QAP* steepest descent pairwise interchanges heuristic was coded (Appendix D) in *MATLAB* and applied in the experimental study.

A.5. Weighted Directed Rooted Spanning Tree's Average Path Length - The Proposed First Theorem

This proposed first theorem proves that the average length of path in an *MWDRST* is less than that for a *DOLA*, ie. $\bar{\mu}_{DOLA} \geq \bar{\mu}_{MWDRST}$.

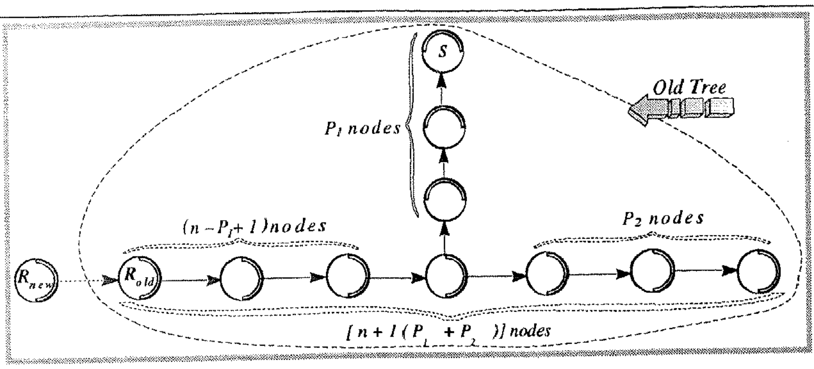


Figure A. 1 - Path lengths in a Maximal Spanning Weighted Directed Rooted Spanning Tree

A.5.1. Proof:

Figure A.1 shows the *MWDRST D* (V_T, A_T) rooted at R_{old}. Only two branches containing p_1 and p_2 nodes with one branch containing the Sink node have been shown, since a tree with only two pendant nodes constitutes the dominant case. If the number of pendant nodes in the tree is greater than two, then the average length of a path in that tree will further decrease due to an increase in the number of paths of the same length. A further assumption is that all arcs have an equal length of one. Hence, the length of a path will be the number of arcs in it. The average length of a path may be derived by dividing the sum of the lengths of all the paths that are possible by the total number of paths. It is required to prove that the average length of a path or chain in the *MWDRST* is less than or equal to the average length of a path or chain in the *DOLA* with the same number of nodes. Cancelling the number of paths in the denominator on both sides of the inequality, it can be stated that:

$$\sum_{k=1}^{n-p_1} k[(n-p_1+1)-k]+\sum_{k=1}^{p_1-1} k(p_1-k)+\sum_{k=1}^{p_2-1}\left(\sum_{k'=1}^{p_1-1} k'+k\right)+\sum_{k=1}^{n+1-(p_1+p_2)}\left(\sum_{k'=1}^{p_1-1} k'+k\right)\leq\sum_{k=1}^{n-1} k(n-k) \quad \textit{Equation A. 13}$$

Assuming that the inequality in the previous equation is true for a *MWDRST* in n nodes (including root and sink nodes), it can be shown by induction on the n-node *MWDRST* that this inequality is true for *MWDRST* with $(n + 1)$ nodes. Thus with reference to *Figure A.1*, by adding the extra node, a new Root node R_{new} can be attached to the old *MWDRST* without increasing the number of pendant nodes in the tree. This case dominates the other case where the additional node is attached to any intermediate nodes on either one of the two chains. It can be seen that the chain to which this node is attached is converted into a tree, which can be proved to have a smaller average path length. However, the following inequality needs to be proved:

$$\sum_{k=1}^{n-p_1+1} k[(n-p_1+2)-k]+\sum_{k=1}^{p_1-1} k(p_1-k)+\sum_{k=1}^{p_2-1}\left(\sum_{k'=1}^{p_1-1} k'+k\right)+\sum_{k=1}^{n+2-(p_1+p_2)}\left(\sum_{k'=1}^{p_1-1} k'+k\right)\leq\sum_{k=1}^{n-1} k[(n+1)-k]$$

Equation A. 14

Subtracting *Equation A.13* from *Equation A.14*, begets the following result:

$$\sum_{k=1}^{n-p_1+1} k[(n-p_1+1-k)+1]+\sum_{k=1}^{n-p_1} k[(n-p_1+1)-k]+$$
$$\sum_{k=1}^{n+2-(p_1+p_2)}\left(\sum_{k'=1}^{p_1-1} k'+k\right)+\sum_{k=1}^{n+1-(p_1+p_2)}\left(\sum_{k'=1}^{p_1-1} k'+k\right)\leq\sum_{k=1}^{n} k[(n-k)+1]+\sum_{k=1}^{n-1} k(n-k)$$

Equation A. 15

Simplification of the previous equation gives the following terms:

$$\sum_{k=1}^{n-p_1+1} k+[n+2-(p_1+p_2)-n-1+(p_1+p_2)]\sum_{k'=1}^{p_1-1} k'+\left(\sum_{k=1}^{n+2-(p_1+p_2)} k-\sum_{k=1}^{n+1-(p_1+p_2)} k\right)\le\sum_{k=1}^{n} k \qquad \textit{Equation A. 16}$$

Equation A.16 yields to the following inequality:

$$\frac{(n-p_1+1)(n-p_1+1)}{2}+\frac{p_1(p_1-1)}{2}+n+2-(p_1+p_2)\le\frac{n(n+1)}{2} \qquad \textit{Equation A. 17}$$

for the limiting *MWDRST* case having *(n + 1)* nodes, ***max** $(p_1 + p_2) = n$*, ***min** $(p_1) = 2$*, and ***max** $(p_2) = (n - 2)$*. Hence, *Equation A.17* will be true for all $n \geq 3$. This is valid condition since, when $n = 3$, the *MWDRST* is simply a path (or chain) linking the root and sink nodes with the third node located between them. This is because the arc *(R, S)* is inadmissible. Only when *n* is *4* or greater, can a tree or a weighted directed rooted spanning tree *(WDRST)* with pendant nodes other than the root or sink nodes be formed. These results supports the proof that in a tree, due to the existence of pendant nodes, nodes will lie on different paths but at the same distance from a common branching node. However, paths with equal weight exist, and in a *DOLA* no two nodes can be at the same distance from any other node, if the paths are measured in the same direction.

A.5.2. Usual context

When the *DOLA* approach is taken to form machine groups along the diagonal of a travel chart, the travel distances correspond to those for a undirectional flowline. In the case of an *MWDRST*, since several groups of machines can be located parallel to each other, the travel distances in each group is reduced. This is equivalent to breaking up a flowline into different shapes (U, L, S, W). In the worst case, an *MWDRST* may be a *DOLA* since a *DOLA* is also a *WDRST*. Otherwise, the *MWDRST* based layout can be expected to yield lower travel times and flow density compared to a flowline structure for the entire shop.

A.6. Weighted Directed Rooted Spanning Tree's Adjacencies - The Proposed Second Theorem

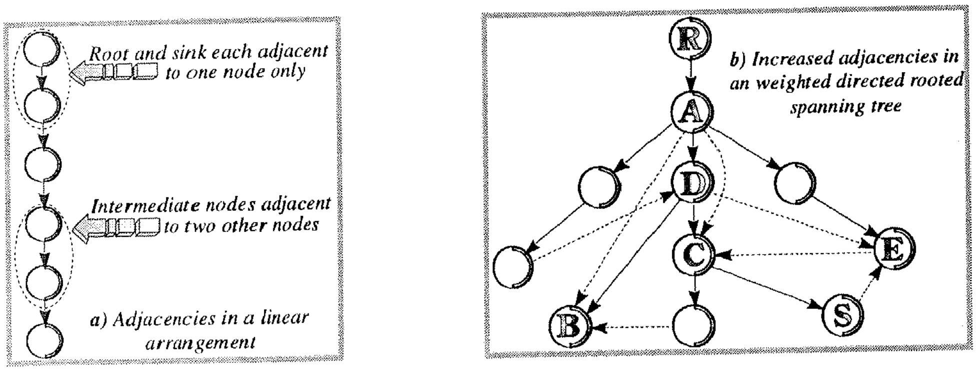
This proposed second theorem proves that the *MWDRST* representation for a layout of *CM* flowlines allows a higher machine adjacency than the *DOLA* for the travel chart. Two nodes in the *WDRST* are said to be adjacent if they neither:

- lie on the same path in the *WDRST* and are connected by an arc, not a path, or
- lie on two different paths in the *WDRST* but can be connected by an arc which does not cross an intermediate path.

A.6.1. Proof:

Given a digraph for a travel chart, a directed optimal linear arrangement of the *n* node allows only *(n - 1)* nodes to be adjacent to each other, as shown in *Figure A.2.a* because only *(n - 1)* arcs constitute the path. The root and sink nodes are adjacent to only one node each. The remaining *(n - 2)* interior nodes are each adjacent to a predecessor and a successor node. Even if each node is treated as a machine group, at most three groups can be considered located adjacent to each other.

In contrast, from *Figure A.2.b,* in the *MWDRST* each node can be located adjacent to at least four other nodes. Since a *MWDRST* is a planar graph, it can be represented in two dimensions with no flow arcs intersecting each other. However, in a planar (undirected) graph of *n* nodes, a total of *(3n - 6)* edges can be drawn without intersecting in the planars. In comparison, from a flow based layout design perspective, reachability and adjacency constraints allow only *(2n - 4)* arcs to be placed in the *MWDRST.* This reduction in the number of arcs is due to differences in the concepts of planarity versus reachability and adjacency in layout design. For example, in ***Figure A.2.b*** arc *(A, B)* is planar and node *B* is reachable from node *A*. However, in the case of arc *(A, C),* it is nonplanar because it crosses arc *(D, E),* but node *C* is still reachable from node *A*. Arc *(D, E)* is planar but node *E* is not reachable from node *D*. Since they lie on neighbouring paths, nodes *(D, E)* are adjacent and reachable from a layout perspective. From a flowline layout and machine sharing perspective, the concepts of reachability and adjacency in *WDRST* were found to be more relevant to the approach adopted in this research program, compared to the concept of the planarity of undirected graphs.


Root and sink each adjacent to one node only
Intermediate nodes adjacent to two other nodes
R
A
D
C
E
B
S

a) Adjacencies in a linear arrangement

b) Increased adjacencies in an weighted directed rooted spanning tree

Figure A. 2 - Adjacencies in paths and weighted directed rooted spanning tree

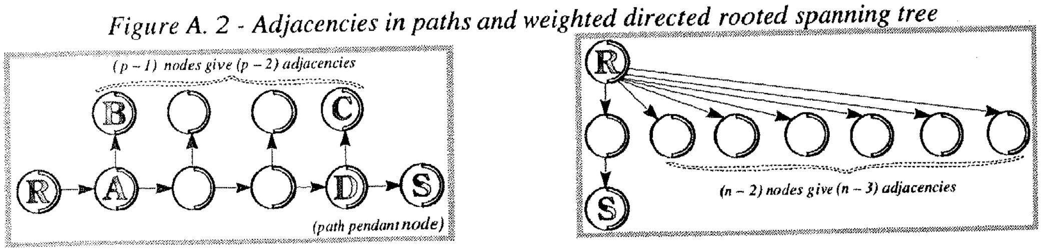

(p - 1) nodes give (p - 2) adjacencies
B
C
R
A
D
S
(path pendant node)
R
S
(n - 2) nodes give (n - 3) adjacencies

Figure A. 3 - Adjacencies in a weighted directed rooted spanning tree

Figure A.2 shows an *n*-node *WDRST* with *p* pendant nodes, including the root and sink nodes. The arc *(R, S)* is not allowed as it implies (say) flow directly from the raw material store to the finished goods store. There is a path/chain from the root *R* to all other nodes. Based on the grid graph representation of the *WDRST*, the total number of adjacencies possible can be calculated as follows: from *Figure A.3* the number of arcs in rectangle *ABCD* + number of arcs in the *MWDRST* + number of arcs in the *MWDRST* are eliminated by rectangle *ABCD* and *(p - 1)* pendant nodes, ie.

$$[2(p-2)+(p-1)] + [(n-1)]+[(p-1) + (p-2)] = (n+p-2).$$

Since the maximum number of adjacencies possible is *(2n - 4)*, this is greater then *(n - 1)* for the linear arrangement, but less than *(3n - 6)* for a planar graph. If the *n*-node directed linear arrangement is reconfigured into the form of a U, the total number of adjacencies (assuming that *n* is even and subtracting *1* for the inadmissible arc connecting root and sink):

$$= [\,(n-1) + 2\,(\,n/2 - 1\,)-1] = (2n-4).$$

Although this matches the number obtained from the *MWDRST*, it should be noted that the total weight of the arcs in the linear arrangement would be less than for the *MWDRST*, as will be proved later in this section. In addition, the linear arrangement lacks the structure of the *WDRST* to find machine groups, cannot define a flowline layout for each group, and does not show the different flowlines related by shared machines as branch nodes. However, methods which obtained solutions for the *NP*-complete optimal linear arrangement (directed or undirected) problems are not preferable to those for extracting an *MWDRST* from the digraph for the travel chart (the *MWDRST* can be obtained by a polynomial time algorithm leaving a reduced *NP*-complete problem for the solution).

A.6.2. Usual Context

In the *MWDRST*, a path is adjacent to two paths if it is not on the left or right edge of the tree. In that case, it will be adjacent to only one other path. Thus, each machine can be considered adjacent to at least four other machines, placed orthogonally around it. Each pair of adjacent flowlines can be assumed connected by an aisle that allows intercell flows. Hence, with an *MWDRST* based layout, higher intracell and intercell machine adjacencies can be expected, which would allow handling to eliminate the need for machine duplication. Thus, this traditional approach of permuting the rows and columns of travel chart is bound to give higher machine duplication than necessary because it is not a two dimensional representation.

A.7. Weighted Directed Rooted Spanning Tree's Maximal Weight - The Proposed Third Theorem

This proposed third theorem proves that the *MWDRST* representation for the travel chart can have a weighting at least as good as that obtained by a *DOLA* solution,

$$\sum_j \sum_i f_{ij}\left(\forall a_{ij} \in MWDRST\right) \geq \sum_j \sum_i f_{ij}\left(\forall a_{ij} \in DOLA\right) \qquad \textit{Equation A. 18}$$

A.7.1. Proof:

First, utilise *Figure A.4* to prove that the weighting of *MWDRST* is at least as great as that of the arcs included in *DOLA*. As shown in *Figure A.4.a*, if all consecutive nodes in the *DOLA* are connected by arcs, then an *WDRST* with higher weighting need not be sought, because, with the exception of the root and sink nodes, all intermediate nodes in the *DOLA* are of the second degree and there are no cycles in the Hamiltonian path so obtained (it is already in the *WDRST*).

However, assuming that arc *(D,E)* does not exist in the *DOLA* obtained, as has been shown in *Figure A.4.b*, then for node *E* in at the most *O(n)* steps, it is possible to check the weights of all arcs from its predecessors (with the exception of node *D* which is the immediate predecessor) in the path. If an arc is obtained, in this case arc *(B,E)* whose weight is greater than that of arc *(D,E)*, the path can be changed into the *WDRST* shown in *Figure A.4.c*. Hence, in the general case, assuming that an optimal solution can be found to the *NP*-complete *DOLA* problem, it can then be converted to an *WDRST* of equal or greater weight in polynomial time solution $O(n^2)$ steps, to check all nodes *B* through *S*.

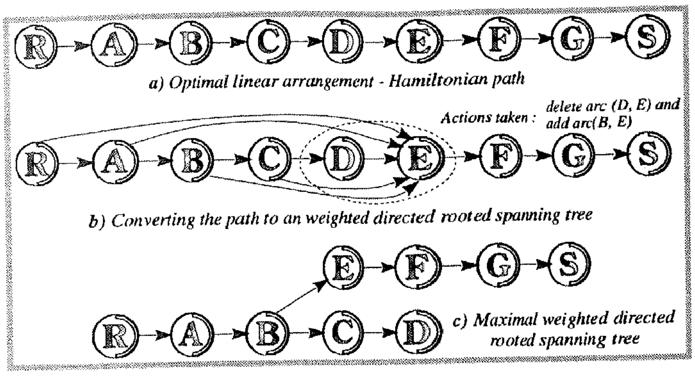

R
A
B
C
D
E
F
G
S
a) Optimal linear arrangement - Hamiltonian path
Actions taken :
delete arc (D, E) and add arc(B, E)
b) Converting the path to an weighted directed rooted spanning tree
c) Maximal weighted directed rooted spanning tree

Figure A. 4 - Converting from a path to a weighted directed rooted spanning tree

Clearly, if the traditional travel chart permutation approach is adopted for the machine grouping, a solution to the *NP*-complete *DOLA* problem may fail to obtain a layout for the machine groups. In contrast, by attempting to first identify an *MWDRST* whose branches are then permuted, the *WDRST* in polynomial time has to be obtained before an *NP*-complete optimal linear arrangement problem with a reduced number of nodes (the number of pendant nodes in the *MWDRST*) can be attempted.

The other advantage of using the *WDRST* is that, unlike a linear arrangement, several pairs of nodes are connected by paths of the same length. In the strict linear arrangement no two paths can have the same length, for example, paths *(B – C - D)* and *(B – E - F)* have the same length in *Figure A.4.c*, which is clearly not the case, as shown in *Figure A.4.a*. In the previous section a more rigorous proof for this condition has been presented. *Figure A.5* gives an example of how the *DOLA* structure for machine grouping in the travel chart gives rise to longer paths, since, in this linear ordering, the benefits of adjacency of paths available in the two dimensional *WDRST* are lost.

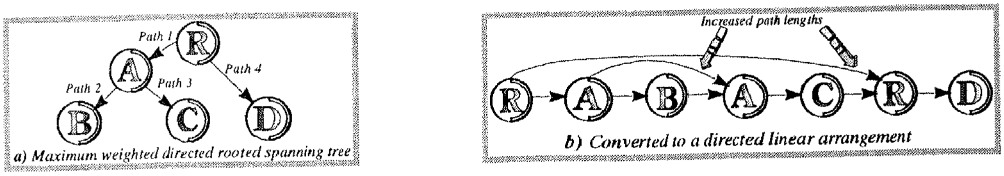

Path 1
R
A
Path 4
Path 2
Path 3
B
C
D
a) Maximum weighted directed rooted spanning tree
Increased path lengths
R
A
B
A
C
R
D
b) Converted to a directed linear arrangement

Figure A. 5 - Increased lengths in linear arrangement

From *Figure A.5* it can shown that, the conversion of the *MWDRST* to a *DOLA* is analogous to the determination of a Hamiltonian path or the more complicated *QAP*. Thus, let F_{ij} be the total weight (flow) on arcs from nodes in path *i* to nodes in path *j*. Using only the arcs not included in the *MWDRST*, a flow matrix can be set up *(Table A.1)*. The off-diagonal values of $-\infty$ are important to prevent *paths* 2 and *3* from preceding *path 1*. *Path 1* must proceed these two paths, although it is possible for *path 4* to separate *path 1* from *paths* 2 and *3* in the *DOLA*.

	P1	**P2**	**P3**	**P4**
P1	$-\infty$	F_{12}	F_{13}	F_{14}
P2	$-\infty$	$-\infty$	F_{23}	F_{24}
P3	$-\infty$	F_{32}	$-\infty$	F_{34}
P4	F_{41}	F_{42}	F_{43}	$-\infty$

Table A. 1 - Flow matrix for obtaining a Hamiltonian path

Enumeration of the tree of the different *DOLA*'s, which are feasible for the given *WDRST*, is shown in *Figure A.6*. The distances between different pairs of paths depends upon the order in which they are placed in the *DOLA*. To find a maximal weight of the Hamiltonian path is the aim, because that will place paths which have high flow interactions consecutively. However, if the length of each path (number of arcs) multiplies its flow volume, then the objective becomes the one dimensional *QAP*.

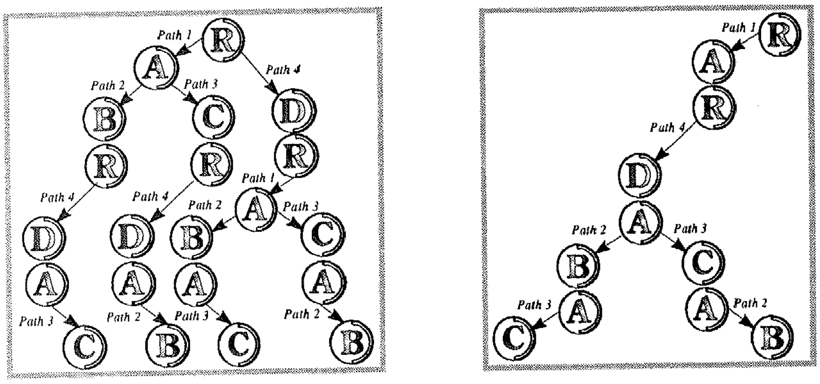
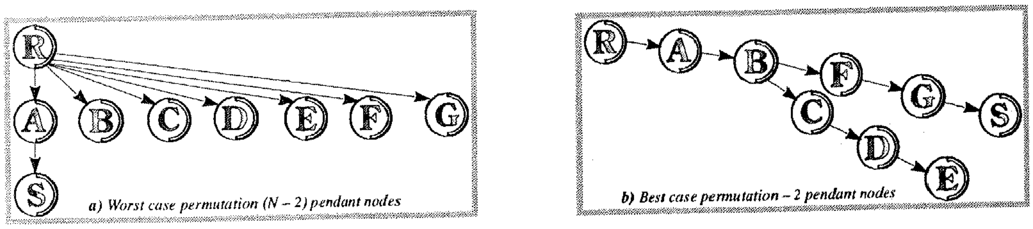


Figure A. 6 - Enumeration tree for feasible arrangements

A.7.2. Usual Context

This result implies that the arcs in the *MWDRST* will include at least the same total flow volume as the *DOLA*, even if it lowers the average travel distances. It is unclear why the *DOLA* model for machine grouping has been persisted with in so many research papers. When this model is used, the problem is already *NP* complete, and thus it is unable to integrate the machine grouping, machine sharing, intercell layout and location conditions. In contrast the *MWDRST* has polynomial time algorithms (Tarjan 1977, 1985) and gives a layout, which would have lower travel distances than a pure linear arrangement. The *MWDRST* also distinguishes between arcs in $F \cup B \cup C_r$ that may not merit machine duplication compared to those in C_n which will. It easily separates the handling and machine duplication decisions to eliminate intercell flows for arcs in C_r and C_n, respectively.



a) Worst case permutation (N – 2) pendant nodes

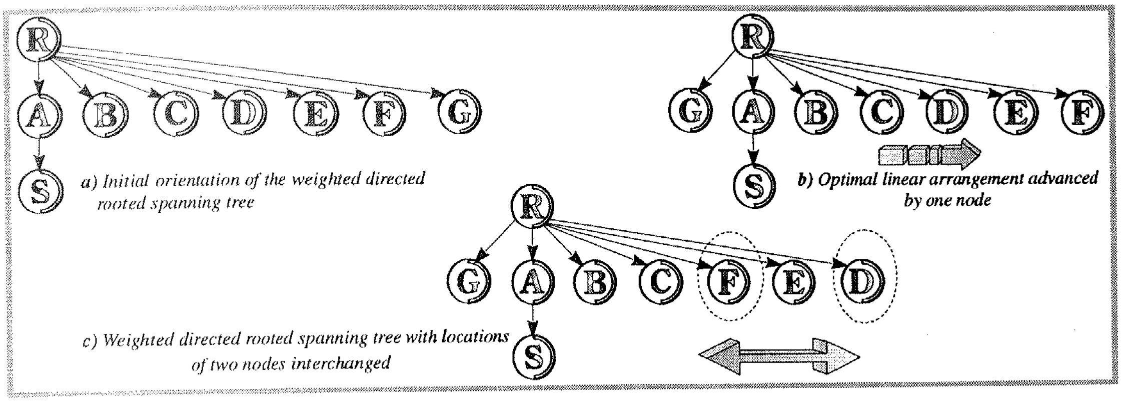
b) Best case permutation – 2 pendant nodes

Figure A. 7 - Best and worst case permutations

A.8. Weighted Directed Rooted Spanning Tree and Permutations – The Proposed Fourth Theorem

In the *DOLA* with *N* nodes the maximum number of permutations that need to be enumerated is *(N - 2) !.* The locations of the root and sink nodes are fixed at either end of the *DOLA*. This allows the remaining *(N - 2)* nodes to be assigned to an equal number of possible intermediate locations. *Figure A.7.a* shows the worst case that can arise in stage II of the suggested cell formation method. Since the arc *(R, S)* cannot be allowed in the *MWDRST*, at least one node *A* must be located on one path. Thus path *(R- A - S)* is equivalent to a pendant node. This makes the maximum number of permutations equal to *(N - 2)!.*

Figure A.7.b gives the lower bound for the problem size for stage II, a *WDRST* with only two pendant nodes. In this case, it can have only two orientations after the branches are permuted. Hence, for an *MWDRST* with *p* pendant nodes, only in the worst case will the stage II problem be equivalent to the *DOLA*.



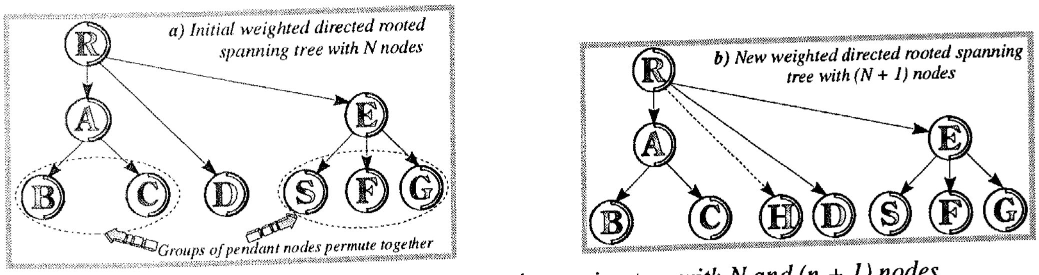
a) Initial orientation of the weighted directed rooted spanning tree

b) Optimal linear arrangement advanced by one node

c) Weighted directed rooted spanning tree with locations of two nodes interchanged

Figure A. 8 - Approaches for weighted directed rooted spanning tree reorientation

Figure A.8.b shows one interpretation of the stage II problem for the given *WDRST* shown in *Figure A.8.a.* The determination of the optimal linear arrangement of the pendant nodes is equivalent to breaking an optimal closed *TSP* tour by deleting one arc. For example, the orientation of the *MWDRST* shown initially in *Figure A.8.a* was obtained by deleting the arc *(G, A)* from the cycle *(A – B – C – D – F – G – A)*. Thus, the orientation in *Figure A.8.b* required the deletion of arc *(F, G)*. If viewed as a one dimensional *QAP*, the locations of any pair of nodes can be interchanged if *(G – A – B – C – D – E – F)* as shown in *Figure A.8.b.* Here the locations of nodes *F* and *D* were swapped to obtain the arrangement in *Figure A.8.c*. Both the *TSP* and *QAP* are analogues of the cluster analysis problem described by Lenstra [1975].



a) Initial weighted directed rooted spanning tree with N nodes

b) New weighted directed rooted spanning tree with (N + 1) nodes

Figure A. 9 - Weighted directed rooted spanning tree with N and (n + 1) nodes

Using *Figure A.9.a* a simple analytical derivation for the results in this section will now be presented. These *WDRST* nodes have a common predecessor. If the root node is such a predecessor, then the path corresponding to that node is simply an arc (node *D* and arc *(R, D)*). Let p be the number of groups of pendant nodes, and N_i the number of nodes in the *i*-th group. The total number of permutation of the branches of the *MWDRST* can be derived as follows:

$$\left(\prod_{i=1}^{p} \textit{(number of permutations with each group)}\right)\textit{(number of permutations of the groups)}=$$

$$=\left(\prod_{i=1}^{p} N_i!\right)p! \qquad \textit{Equation A. 19}$$

A.8.1. Proof

Thus proof is required of the following statement:

$$\left(\prod_{i=1}^{p} N_i!\right)p! \le (N-2)! \qquad \textit{Equation A. 20}$$

As before, using the induction approach, it will be assumed that *Equation A.20* is valid and the case will be considered for *(N + 1)* nodes. The addition of this extra node has been shown in *Figure A.9.b* to be achieved by attaching an arc *(R, H)* to the root. This is the worst case scenario, since this additional arc could have been attached to any one of the existing pendant nodes. Hence, it is required to prove the following:

$$\left(\prod_{i=1}^{p} N_i!\right)(p+1)! \le (N-1)! \quad or \quad \left(\left(\prod_{i=1}^{p} N_i!\right)p!\right)(p+1) \le (N-2)!(N\text{ - }1) \qquad \textit{Equation A. 21}$$

Which, on simplification using *Equation A.20*, requires the prove that $(p + 1) \le (n - 1)$. That is, $(p + 2) \le N$ must be true. With reference to *Figure A.10.a*, an *MWDRST* with only the arc *(R, S)* is inadmissible. When $N = 3$, as in *Figure A.10.b*, the path *(R – A – S)* could be considered to be a tree with $p = 1$. Only when $N \ge 4$, can operational sequences such as *(R - A – S)*, *(R – A – B – S)* or *(R – B – A – S)* give a completely connected digraph, from which a *WDRST* with $p = 2$ is the result, as shown in *Figure A.10.c*. Arcs *(B, S)* and *(B, A)* are the only feasible crisscrossing and backward arcs, respectively, in this feasible *WDRST*.

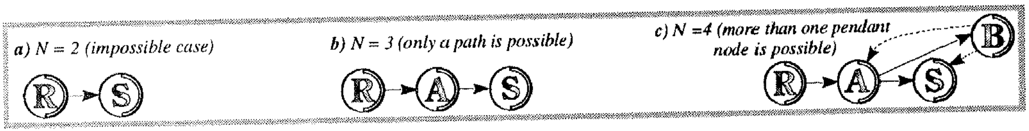

a) N = 2 (impossible case)
R
S
b) N = 3 (only a path is possible)
R
A
S
c) N =4 (more than one pendant node is possible)
R
A
S
B

Figure A. 10 - Minimum number of nodes for a weighted directed rooted spanning tree

A.9. Weighting Scheme – The Proposed Fifth Theorem

This proposed fifth theorem proves that, given an *WDRST* with *p* pendant nodes and maximum intercell flow of *max* (F_{ij}) between any pair of paths, a feasible weighting scheme for the branching nodes at the *k-th* level of the *WDRST* can be $w_2 = w_3 = \ldots = w_k\ (k \ge 2) = p_{max}\ (F_{ij})$.

A.9.1. Proof:

Solving the second stage for any *WDRST* as an *OLA* problem requires path adjacencies due to common branching nodes to be maintained. The paths must be pivoted about one or more of their branching nodes. This will prevent machine divisioning and altering the structure of the *WDRST* to minimise the lengths of the crisscrossing arcs. From *Figure A.11.a*, assuming that $F_{12} \ge F_{23} > F_{14}$, the desired permutations should be restricted to *paths 3* and *4* about the pivot node *C*. It is now required to control paths permutations, starting with pendant nodes at the lowest level in the *MWDRST*.

However, if the node weighting scheme is not correctly incorporated, it is possible that the stage II solution can give a sequence such as *P4 – P1 – P2 – P3*. Thus, for example, if *path 4* is placed to the left of *path 1* as shown in *Figure A.11.b*, this will minimise the distance between *paths 1* and *4*. However $(R_{14}$

$+ L_{14})=1$ and will force a duplication of machine *C*. Since it is preferred to maintain a functional layout for all machine types, this machine splitting should be prevented. It will be shown that this constrained permutation of the branches of the *MWDRST* can be achieved by suitable weighting of the branching nodes. Therefore, this problem can be solved by having the stage II model simulate the traditional pairwise node exchange heuristic adopted in the *QAP* analogues.

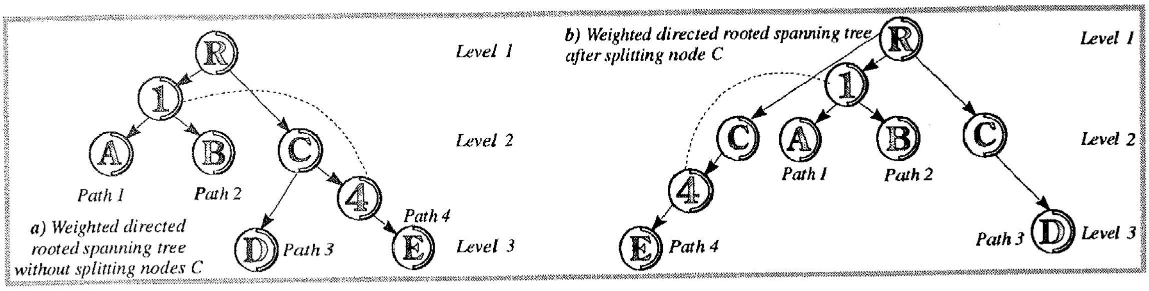

R
1
A
B
C
4
D
E
Path 1
Path 2
Path 3
Path 4
Level 1
Level 2
Level 3
a) Weighted directed rooted spanning tree without splitting nodes C
b) Weighted directed rooted spanning tree after splitting node C
R
1
C
A
B
C
4
Path 1
Path 2
E
Path 4
Path 3
D
Level 1
Level 2
Level 3

Figure A. 11 - Splitting of paths sharing a branching node

Assuming the root node at *level 1* can be assigned a weight $w_1 = 0$, and the most critical weight is that given to all branching nodes at *level 2*, w_2, it can be shown that, having decided the value for w_2, it is sufficient to set $w_3 = w_4 = \ldots = w_k\ (k > 2) = w_2$. No penalty is incurred for nodes that lie on a path that does not contain any branching node (the pendant nodes *A* and *B* at *level 2* or nodes *D* and *E* at *level 3*). The reason is that these paths can be absorbed into the last branching node that they contain without disrupting the stage II solution. Thus, letting *P* be the total flow penalty for the *WDRST* in *Figure A.11.a*:

$$P = w_2\,(R_{34} + L_{34}) + F_{14}\,(R_{14} + L_{14})$$

Letting *P'* be the total flow penalty for the *WDRST* in *Figure A. 11 b*:

$$P' = w_2\,(R'_{34} + L'_{34}) + F_{14}\,(R'_{14} + L'_{14})$$

Since $P'>P$, it is required to seek a value for w_2 which will give the result $(p - P) > 0$, ie.,

$$w_2\,[(R'_{34} + L'_{34}) - (R_{34} + L_{34})] - F_{14}\,[(R_{14} + L_{14}) - (R'_{14} + L'_{14})] > 0$$

given an *WDRST* with *p* paths, the minimum value of :

$$[(R'_{34} + L'_{34}) - (R_{34} + L_{34})] \text{ is } 1$$

and the maximum value of :

$$[(R_{14} + L_{14}) - (R'_{14} + L'_{14})] \text{ can be } (p - 1), \text{ and}$$

hence to make $(p' - p) > 0$, it is required that:

$$W_2 > (p - 1)\,F_{14}.$$

If a maximum intercell flow matrix of $max\ (F_{ij})$ is assumed due to the crisscrossing arcs, the highest flow penalty contribution due to any particular intercell flow will be $(p - 1)\ max\ (F_{ij})$. This is obtained by placing the two paths on either edge of the *WDRST (Figure A.11.b)*. Hence, if w_2 is set to equal $p\ max\ (F_{ij})$, this will also prevent the divisioning of paths sharing most branching node, such as *paths 3* and *4* in *Figure A.11.a.*

An earlier statement (section A.4.2. Stage II) gave that, having decide the value for w_2, it is sufficient to set $w_3 = w_4 = \ldots = w_k\ (k > 2) = w_2$. With reference to *Figure A.12*, it can be seen that the total weight due to branching nodes common to a pair of paths is directly proportional to the number of branching nodes they share. *Path 2*, attached to node *F*, will have a lower weight due to machine sharing with *paths 3* and *4*. This is because, at node *F*, *paths 3* and *4* constitute a subtree with an extra common branching node *C*. Hence, neglecting the intercell flow between *paths 3* and *4*, simply by setting $w_k = \ldots = w(i+1) = w_i = w(i-1) = \ldots = w_2$, will force stage II model to permute *paths 3* and *4* together, thus avoiding assigning these two paths to locations such that $(R_{34} + L_{34}) > 1$. This is because paths with common branching nodes at level *k* will have a machine sharing weight of $(k-1)w_2$, far exceeding the maximum intercell flow penalty in stage II from any crisscrossing arcs.

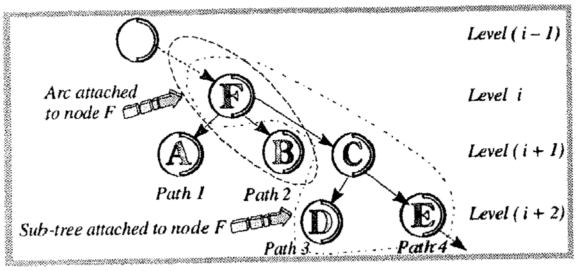


Figure A. 12 - Weighting of nodes at other levels

Due to the acyclic structure of the *WDRST*, the number of branching nodes common to two paths also suggests which pairs of paths should not be split in the second stage *OLA* solution. In *Figure A.12*, *paths 3* and *4* have an extra branching node *C* in common compared to *path* 2. Hence, they should be moved together when the stage II model is solved even though they can permute with respect to each other.

APPENDIX B

B.1. Spanning Tree Algorithms

In this Appendix B, spanning tree algorithms, definition and pseudo codes are presented with simple illustrative examples. Also illustrated are branching and cut tree algorithms.

B.1.1. Minimum Spanning Tree

The minimum spanning tree problem was first formulated in 1926 by Boruvka. Here it was required during the electrification of Southern Moravia to find out the most economical layout of a power-line network. Some polynomial-time algorithms have been developed to solve this problem. Amongst them the Prim and Dijkstra algorithms are the most famous ones.

Now suppose the edges of the graph have weights or lengths. The weight of a tree is just the sum of the "weights" of its edges. Obviously, different trees have different lengths. The problem is how to find the minimum length spanning tree. What is a tree? A tree is a connected graph without cycles, with the following properties:

- Every pair of vertices in a tree is joined by exactly one path.
- If an edge is deleted from a tree, then the resulting graph has exactly two components.
- If a tree has n vertices and m edges, then $m = n - 1$.
- If a new edge is added to a tree joining a pair of non-adjacent vertices, then there is exactly one cycle in the new graph.

A spanning tree of a graph G is a subgraph of G such that it is a tree and it contains all vertices of G. A graph has a spanning tree if and only if it is connected. Now consider an edge-weighted graph G. The Minimum Spanning Tree (*MST*) problem asks to find a spanning of G with the minimum weight.

Minimum Spanning Tree Property:

- Let $G = (V, E)$ be a connected graph with the edges - cost (u, v) of the tree.
- Let U be a subset of V.

- If *(u, v)* is an edge of lowest cost such that u is in *U* and v is in *V - U*, then there is a minimum spanning tree that includes *(u, v)*.

This problem can be solved by Prim's algorithm, which builds a tree one vertex at a time, and is one of its many variations.

Input. An edge-weighted graph *G* with n vertices represented by adjacent lists.

Output. The set of edges in a spanning tree *T*.

```
Begin take a vertex v; count := 0; current vertex := v; S := {v}; T := *;
while count < n - 1 do;
begin label the other vertices i not in S adjacent to current vertex as follows:
if i is not labeled, label it by length(current vertex, i) and let reference(i) be current vertex;
if i is labeled, re-label it by min{label(i), length(current vertex, i)} and;
if label is changed, reference(i) := current vertex;
find a vertex u not in S with the minimum label;
add u into S; add the edge (u, reference(u)) in to T; current vertex:= u; Count := count + 1;
end; end.
```

Figure B. 1 - MST Prims algorithm pseudo code

The algorithm from *Figure B.1* can be organised into a table. Thus every vertex corresponds to a column and every step corresponds to a row in the table. When it gets a label, the label and the reference vertex are listed in the row. Then find a vertex with the smallest label, which is added into *S*, and the edge joining this vertex and its reference is added into *T*. Next, update the label and the reference vertex of all vertices adjacent to the newest vertex in *S*.

A[1] : 2(12) - 5(18) - 6(14),
A[2] : 1(12) - 3(9) - 4(11) - 5(17),
A[3] : 2(9) - 4(16),
A[4] : 2(11) - 3(16) - 5(10),
A[5] : 1(18) - 2(17) - 4(10) - 6(13),
A[6] : 1(14) - 5(13).

Table B. 1 - Graphs, adjacency input list

Vertices							
1	*2*	*3*	*4*	*5*	*6*	*S*	*T*
0						1	{}
	12/1			18/1	14/1	2	{12}
		9/2	11/2	17/2	14/1	3	{23}
			11/2	17/2	14/1	4	{24}
				10/4	14/1	5	{45}
					13/5	6	{56}

Table B. 2 - Algorithms procedure

Since each edge added is the smallest connecting *T* to *G - T*, only edges are added that should be part of the *MST*. Again, it looks like the loop has a slow step in it. But again, some data structures can be used to speed this up (use a heap algorithm procedure, for each vertex, the smallest edge connecting *T* with that vertex). The complexity of the algorithm is $O(n^2)$ and a simple example is shown below. *Table B.1* shows the adjacency lists of the graph. Take vertex *1* as the first vertex and *T = {1}*, *T* = *. In the following *Table B.2* vertices that are not labelled are omitted, from were a resulting *MST* is, *T = {12,23,24,45,56}*.

B.1.1.1. Kruskal's Algorithm

Kruskal's algorithm (*Figure B.2*) is now presented, which is the easiest to understand and probably the best one for solving problems by hand. Note that, whenever an edge *(u,v)* is added, it is always the smallest connecting the part of *S* which is reachable from u with the rest of *G*, so it must be part of the *MST*. This algorithm is known as a greedy algorithm, because it chooses at each step the "cheapest" edge to add to *S*. Care is required when trying to use greedy algorithms to solve other problems, since it usually doesn't work, ie. if the shortest path from *a* to *b* is required, it might be a bad idea to keep taking the shortest edges. The greedy idea only works in Kruskal's algorithm because of the essential property:

```
sort the edges of G in increasing order by length;
    keep a subgraph S of G, initially empty;
for each edge e in sorted order;
if the end points of e are disconnected in S; add e to S;
return S.
```

Figure B. 2 – Kruskal's algorithm

The algorithm analysis will show a slow iteration time ie. linear time per iteration, or *O(mn)* total when line testing whether two endpoint's are disconnected. But actually there are some complicated data structures that enable each test to be performed in close to a repetitive time; this is known as the union-find problem. The slowest part turns out to be the sorting step, which takes *O(m log n)* time.

B.1.1.2. Boruvka's Algorithm

Although this seems a little complicated to explain, it's probably the easiest one for computer implementation since it does not require any complicated data structures. The idea is to do steps, like Prim's algorithm, in parallel all over the graph at the same time. Next, in *Figure B.3* is presented Boruvka's algorithm.

```
make a list L of n trees, each a single vertex;
        while (L has more than one tree);
        for each T in L, find the smallest edge connecting T to G-T;
add all those edges to the MST (causing pairs of trees in L to merge).
```

Figure B. 3 - Boruvka's algorithm

As was noted in Prim's algorithm, each edge added must be part of the *MST*, so it must be acceptable to add them all at once. Analysing Boruvka's algorithm can be found similar to a merging sort (graph theory algorithm procedure). Each pass reduces the number of trees by a factor of two, so there are *O(log n)* passes. Each pass takes time *O(m)* (first assess which tree each vertex is in, then for each edge test whether it connects two trees and check if it is better than the ones seen before for the trees on either endpoint) so the total time is *O(m log n)*.

B.1.2. Minimum Cost Spanning Tree

For the applied *MST* to find the minimum cost spanning tree *(MCST)*, the defined *MCST* property is required as follows:

- Let $G=(V, E)$ be a connected graph where for all (u, v) in E there is a cost vector $C[u,v]$.

- A graph is connected if every pair of vertices is connected by a path.

- A spanning tree for G is a free tree that connects all vertices in G.

- A connected acyclic graph is also called a free tree.

- The cost of the spanning tree is the sum of the cost of all edges in the tree and usually it is required to find a spanning tree of minimum cost.

From the previous section, the two main properties of the trees are:

Property 1: A tree with $N \geq 1$ vertices has exactly $N - 1$ edges.

Proof by Contradiction: Let $G=(V, E)$ be the smallest tree that does not satisfy property 1. G must have more than 1 vertex, since the only 1-vertex tree has 0 edges, which satisfies property 1. Furthermore, there must be a vertex v with exactly 1 incident edge (v, w). No vertex can have 0 edges, as it would not be connected, and so G would not be a tree. If every vertex has at least two incident vertices, consider the path created from a vertex v, entering and leaving vertices by a different edge. Such paths will eventually form a cycle, so G is not a tree. If edge (v, w) is deleted, we get a new tree G_2 produced that satisfies property 1 (thus the new tree is smaller than G but is a tree, contradicting the assumption about G). Since G_2 has $N - 1$ vertices and $N - 2$ edges, G must have N vertices and $N - 1$ edges, also contradicting our assumption about G.

Property 2: Adding an edge to a tree introduces a cycle.

Proof: According to property 1, every tree has N vertices and $N-1$ edges. If an edge is added to a free tree than, a connected graph with N vertices and N edges exists. If the edge does not introduce a cycle, than a connected acyclic graph, with N vertices and N edges, which is a tree that violates property 1 (a contradiction) could exist.

Minimum Spanning Tree Property was explained in the previous section (B.1.1.), here only a proof by contradictions is presented, assuming the contrary. Consider T, a *MCST* for G. According to property 2 of trees, adding (u, v) to T introduces a cycle involving (u, v). There must be another edge (u_2, v_2) in T such that u_2 is in U and v_2 is in $V - U$; otherwise there is no way for the cycle to leave vertices in U, enter vertices in $V - U$, and return without using (u,v) twice. Deleting (u_2,v_2), breaks the cycle and gets a spanning tree T_2. The cost of T_2 is the cost of $T - (u_2,v_2), + (u,v)$. Since (u,v) is the least cost edge between vertices in U and V-U, the cost of T_2 is less than or equal to the cost of T. This contradicts the assumption that a minimum cost spanning tree would not include (u, v).

Prim's algorithm to find the minimum cost spanning tree exploits this property. Initially, Prim's algorithm has one node in the spanning tree, and no edges. The algorithm adds nodes to the spanning tree one at a time, in order of the edge cost to connect to the nodes already in the tree. Analysis of Prim's Algorithm shows:

- Prim's algorithm is $O(N^2)$.
- The while loop is executed n-1 times, requiring $O(N)$.
- One vertex to U is added at each iteration.
- Exiting the loop when $U = V$.
- Finding the lowest cost edge from U to V-U in $O(N)$ time.
- Maintaining two arrays: *Closest[v]* and *Lowcost[v].*
- For v in V-U, *Closest[v]* is the vertex u in U closest to v.
- *Lowcost[v]* is the cost of (v,u).

Prim's Algorithm for finding minimum cost spanning tree, is shown in *Figure B.4.*

```
int C[N][N];
PrimMCST (set <edge> *T);
{ set <vertex> U; int Lowcost[N], Closest[N]; int i,j,k;
/* T = set of edges in spanning tree and U = set of all connected vertices */;
Insert(U,0);
for (i = 1; i < N; i++) { Lowcost[i] = C[0][i]; Closest[i] = 0; };
for (i = 1; i < N; i++) {;
            min = Lowcost[2];  k = 2; for (j = 2; j < N; j++);
            if (Lowcost[j] < min) ;
            { min = Lowcost[j]; k = j; } /* Add edge to T, k to U */;
Insert(T,k,Closest[k]); Lowcost[k] = INFINITY; /* Adjust costs to enter U */;
for  (j = 1; j < N; j++);
if (C[k][j] < Lowcost[j] - Lowcost[j] < INFINITY) {;
Lowcost[j]  =  C[k][j]; Closest[j] = k; } } } } /* PrimMCST */.
```

Figure B. 4 - Pseudo code of Prim's algorithm for MCST

B.1.3. Maximum Spanning Tree

Description of the original maximum spanning tree problem is as follows:

Input: A connected and undirected graph $G = (V, E)$ where $V = \{v_1, v_2,...,v_n\}$ is a finite set of vertices on the graph, and $E = \{e_{ij} \mid e_{ij} = (v_i, v_j), v_i, v_j \in V\}$ is a finite set of edges representing connections between vertices. Each edge has an associated positive real number denoted with $W = \{w_{ij} \mid w_{ij} = w(v_i, v_j), w_{ij} > 0, v_i, v_j \in V\}$ representing weight, distance, cost and so on. The vertices and edges are sometimes referred to as nodes and links. Thus:

```
let T be a single vertex x;
          while (T has fewer than n vertices);
          {find the longest edge connecting T to G-T;
          add it to T};
```

Figure B. 5 Modified Prim's algorithm applied to maximum spanning tree

Output: A maximum spanning tree is a spanning tree with a maximal set of edges from E that connects all the vertices in V. Therefore at least one spanning tree can be found in graph G. The maximum spanning tree, denoted as T^*, is the spanning tree whose total weight of all edges is maximal. It can be formulated as follows:

$$T^* = max \sum w_{ij} \qquad \text{Equation B. 1}$$

Prim's algorithm *(Figure B.5),* builds a tree one vertex at a time rather than builds a subgraph one edge at a time. Also, as described before, it finds a minimum (weight) spanning tree. It is easy to see from *Figure B.5*, that the simple modification of replacing min by max in algorithm pseudo code, will cause the algorithm to find a maximum (weight) spanning tree.

```
begin  Let v1 be the origin vertex, and initialise W and ShortDist[u] as W := {v1};
            ShortDist[v1] :=0;
for each u in V - {v1};
            ShortDist[u] := T[v1,u];  Now repeatedly enlarge W until W includes all vertices in V;
while W < V; Find the vertex w in V - W at the minimum distance from v1;
            MinDist := INFINITE;
for each v in V – W;
if ShortDist[v] < MinDist; MinDist = ShortDist[v]; w := v;
end {if};
end {for}; Add w to W;
            W := W U {w}; Update the shortest distance to vertices in V – W;
for each u in V – W; ShortDist[u] := Min(ShorDist[u],ShortDist[w] + T[w,u]);
end {while}.
```

Figure B. 6 - Pseudo code for Dijkstra's algorithm

B.1.4. Dijkstra's Algorithm.

This is another greedy technique often known as *Dijkstra's* algorithm for finding the shortest path between a given pair of vertices. The algorithm works by maintaining a set *S* of vertices whose shortest distance from the source is already known. Initially, *S* contains only the source vertex. Each step adds to *S* a remaining vertex *v* whose distance from the sources is as short as possible. Assuming all arcs have nonnegative costs, finding a shortest path from the sources to *v* that passes only through vertices in *S*, is possible. At each step of the algorithm, an array *D* is used to record the length of the shortest special path to each vertex. Once *S* includes all vertices, all paths are special, so *D* will hold the shortest distance from the source to each vertex.

Input. A digraph *G* with *N* vertices represented by adjacency lists with a length field, and two vertices *S* and *T*.

Output. A shortest path from *S* to *T*, and its length.

The Dijkstra algorithm for solving shortest path problems is presented in *Figure B.6*. There are also other algorithms to solve these problems. The complexity of the algorithm is $O(N^2)$. For simple graphs, this algorithm can be organised into a table, as in the following example, *Table B.3*. Below is the adjacency lists representing a graph *G* with *8* vertices:

A[1] : 2(2) - 3(8) - 4(1),
A[2] : 1(2) - 3(6) - 5(1),
A[3] : 1(8) - 2(6) - 4(7) - 5(4) - 6(2) - 7(2),
A[4] : 1(1) - 3(7) - 7(9),
A[5] : 2(1) - 3(4) - 6(3) - 8(9),
A[6] : 3(2)- 5(3) - 7(4) - 8(6),
A[7] : 3(2) - 4(9) - 6(4) - 8(2),
A[8] : 5(9) - 6(6) - 7(2).

Table B. 3 - Graphs adjacency list

In a list *A[i], j(l)* means an edge v_iv_j with length *l*. The algorithm for finding the shortest path from vertex v_1 the source, to vertex v_8 the sink, is shown in the following *Table B.4*, with the resulting shortest path as v_1 - v_2 - v_5 - v_3 - v_7 - v_8.

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
0							
	2/ v_1	8/ v_1	1/ v_1				
	2/ v_1	8/ v_1				10/ v_4	
		8/ v_1		3/ v_2		10/ v_4	
		7/ v_5			6/ v_5	10/ v_4	12/ v_5
		7/ v_5				10/ v_4	12/ v_5
						9/ v_3	12/ v_5
							11/ v_7

Table B. 4 - Result of Dijkstra example

The Dijkstra algorithm (*Figure B.7)* is applied to the directed graph. For a given directed graph *G = (v, E)* (where *V – {1...n}* and vertex *1* is the sources), and *C* is a two dimensional array of costs (where *C[i, j]* is the cost of going from vertex *i* to vertex *j* on arc *i →j).* If there is no arc *i →j*, assume that *C [i, j]* is ∞, some value much larger than any actual cost. At each step, *D[i]* contains the length of the current shortest special path to vertex *i*.

```
begin
1. S : = {1};
2. For I ; = 2 to n do;
3. D [I] : = C [1, I]; {initialise S};
4. For I : = 1 to n – 1 do begin;
5. Choose a vertex w in V – S such that D [w] is minimum;;
6. Add w to S;
7. For each vertex v in V – S do;
8. D [v] ; = min (D[v], D[w] + C [w, v]);
End; End; {Dijkstra, computes the shortest cost from vertex 1 to every vertex of a directed graph}.
```

Figure B. 7 - Dijkstra's algorithm for directed graph

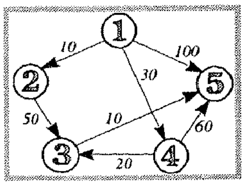


Figure B. 8 - Digraph with labelled arcs

Now apply Dijkstra algorithm to the directed graph of *Figure B.8.* Initially, *S = {1}, D[2] = 10, D[3] =* ∞*, D[4] = 30 and D[5] = 100.* In the first iteration of the loop of lines *(-4) – (8), w = 2* is selected as the vertex with minimum *D* value. Then setting *D[3] = min (*∞*, 10 + 50) = 60, D(4)* and *D(5)* does not change, because reaching them from *1* directly is shorter than going through vertex 2. The sequence of *D* values after each iteration of the loop is shown in *Table B.5*.

Iteration	*S*	*w*	*D [2]*	*D [3]*	*D [4]*	*D [5]*
Initial	{1}	-	10	∞	30	100
1	{1, 2}	2	10	60	30	100
2	{1, 2, 4}	4	10	50	30	90
3	{1, 2, 4, 3}	3	10	50	30	60
4	{1, 2, 4, 3, 5}	5	10	50	30	60

Table B. 5 - Computation of Dijkstra on diagraph of Figure B.7

B.2. Optimum Branching

Chu and Liu [1965], Edmonds [1967] and Bock [1971] have independently given efficient algorithms for finding optimum branching in a graph *G*. The Chu – Liu and Edmonds algorithms are virtually identical, the Bock algorithm is similar but stated as an algorithm on matrices rather than on graphs. This part of the research appendix gives an efficient implementation of the Chu –Liu – Edmonds algorithm. If *G* has *n* vertices and *m* edges the algorithm is *O(m log n)* time [Tarjan 1985].

How do these algorithms work? Let $G = (V, E)$ be a weakly connected directed graph, having edge values $c(v, w)$. The Chu – Liu – Edmonds algorithm finds an optimum branching in G. The algorithm constructs a set of edges *H* defining a subgraph *G (H)* such that for each strongly connected component *S*, there is at most one edge $(v, w) \in H$ such that $w \in S$, $v \in V - S$. If *S* is such that no edge $(v, w) \in H$ satisfies $w \in S$, $v \in V - S$, than it is called *S* a root component of *G (H)*. Initially $H = 0$; thus initially each vertex in *V* defines both a weakly connected and a strongly connected component of *G (H)*. In the next section are the main steps of the algorithm for constructing the set *H*, shown in *Figure B9*.

1. ***Choose** any vertex v with an edge (x, v) of value c(x, v) > 0;*
2. *Select the edge (u, v) of largest value;*
3. *Add (u, v) to H;*
4. *Choose any root component S of G(H) having an unexamined edge (x, v) with $v \in S$ and c(x, v) > 0;*
5. *Find the largest unexamined edge (u, v) such that $v \in S$;*
6. ***If** $u \notin S$, discard the edge and stop. Otherwise go to step 7;*
7. *$U \notin S$. Let W be the weakly connected component of G (H) containing v. **If** $u \notin W$ add (u, v) to H and stop. Otherwise go to step 8;*
8. *$U \notin S$, $u \in W$. Find the sequence S1, (x_1, y_1), S_2, (x_2, y_2), …, S_k, (x_k, y_k) such that each S_i is a strongly connected component of G(H), $(x_i, y_i) \in H$, $y_i \in S_i$, and $x \in S_{i+1}$ for all i, $S_k = S$, $(x_k, y_k) = (u, v)$, and $x_k \in S_1$;*
9. *Find the edge (x_i, y_j) with minimum value among the (x_i, y_i);*
10. ***For** each unexamined edge (x, y) state as follows:*
11. *$C(x, y) = c(x, y) - c(x_i, y_i) + c(x_j, y_j)$;*
12. *Add (u, v) to H (This combines $S_1, \dots, S_k$ into a single strongly connected component which is a root component of G(h).);*
13. ***Repeat** the general step until there is no root component S of G(H) having an unexamined edge (u, v) with $v \in S$ and c(u, v) > 0.*

Figure B. 9 - Optimum branching algorithm's step

B.2.1. Maximum Branching Algorithm

Chu –Liu – Edmonds algorithm from the previous section is for minimum branching. Now in the preceding subsection the original algorithm to find maximum branching was modified. A maximum branching algorithm finds a subgraph of a digraph which is a maximum weight, which does not necessarily mean a spanning tree, and here a forest of out trees is called a maximum branching. An outline of the algorithm is shown in *Figure B.10.*

```
1.  $BV \leftarrow BE \leftarrow \Phi$;
2.  $i \leftarrow 0$;
3.  if $BV = V_i$ then go to 14;
4.  For some vertex $v \not\in BV$ and $v \in Vi$ do;
      begin;
5.    $BV \leftarrow BV \cup \{v\}$;
6.    Find an edge $e = (x, v)$ such that $W(e) = \max\{w(y, v) \mid (y, v) \in E_i\}$;
7.    if $0 \geq w(e)$ then go to 3;
      end;
8.  if $BE \cup \{e\}$ contains a circuit then ;
      begin ;
9.    $i \leftarrow i + 1$;
10.   construct $G_i$ by shrinking $C_i$ to $u_i$ ;
11.   Modify BE, BV and some edge – weights;
      end ;
12. $BE \leftarrow BE \cup \{e\}$;
13. go to 3;
14. while $i \neq 0$ do;
      begin;
15.   reconstruct $G_{i-1}$ and rename some edges in BE;
16.   if $u_i$ was a root of an out – tree in BE then $BE \{ e \mid e \;\; C_i$ and $e \neq e_o^i \}$;
17.   else $BE \{ e \mid e \;\; C_i$ and $e \neq e_i \}$;
18.   $i \leftarrow i\text{-}1$;
      end ;
19.   Maximum branching weight $\leftarrow \sum_{e \in BE} w(e)$.
```

Figure B. 10 Pseudo code for algorithm to find a maximum branching

B.2.1.1. Algorithm Procedure

The algorithm traverses the digraph, examining vertices and edges. It places vertices in a so-called vertex-bucket *BV* when they have been examined, and edges in an edge-bucket *BE* if they have been provisionally chosen for the branching. Throughout the course of the algorithm, *BE* always contains a branching, that is an acyclic collection of directed edges with at most one edge incident to any given vertex. The examination of the vertex *v* consists simply of choosing an edge of maximum positive weight e that is incident to v. It may be noted that no edge of negative weight would be chosen for a maximum branching (a digraph consisting of negative weighted edges only has a maximum branching of zero weight and no edges). The edge *e* is checked to see if it forms a circuit with the edges already in *BE.* If it does not, then *e* is added to *BE* and a new vertex is examined. If it does form a circuit then the graph is restructured by shrinking this circuit to a single vertex and assigning new weights to those edges which are incident to this new “artificial” vertex.

The process of examining vertices then continues until *BV* contains all the vertices of the final graph. It contains just these vertices, several of which may in general be "artificial", because whenever a circuit is shrunk to form a new graph, the edges and vertices of the circuit are removed from *BE* and *BV*. *BE* at this stage contains the edges of a maximal branching for the final graph. The reverse process of replacing in turn each of the artificially created vertices by its associated circuit then begins. At each replacement the choice of edges placed in *BE* is such that for the currently reconstructed graph *BE* contains the edges of a maximum branching. The crucial element of the algorithm is the rule for reassigning weights to edges when circuits are shrunk. It is this rule which forces the choice of edges to be included in the branching when the reconstruction phase is underway.

B.2.1.2. Application of a Maximum Branching Algorithm

Figure B.11 shows an application of an maximum branching algorithm. The vertices are imagined to be examined in alphabetical order, with artificial vertices being added at the tail and in the order they are created. Starting with G_0, (a) shows the successive graphs G_1 and G_2 obtained in the circuit reduction stage of the algorithm. Coincidentally, for it would not generally be the case, *BV* and *BE* are empty when the processing of G_1 and G_2 starts. For G_0, G_1 and G_2 the final values of *BE* and *BV* are shown.

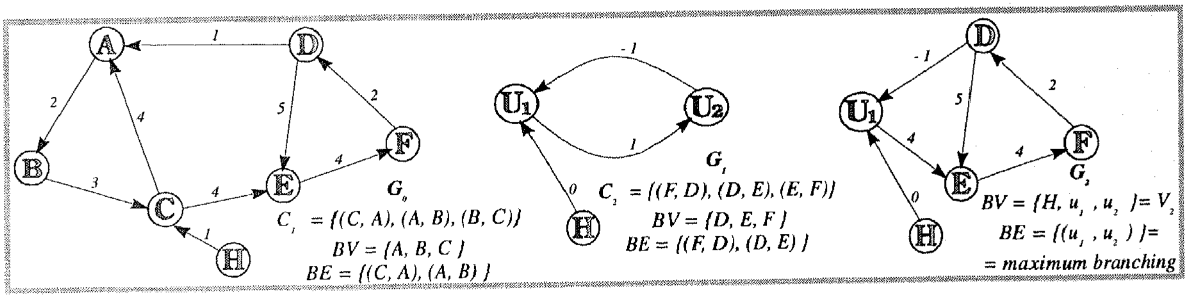


Figure B. 11 - Application of a maximum branching algorithm, showing construction of G_1, G_2,..., G_k

Now the successive contents of *BE,* as G_1 and G_0 are constructed, results in a maximum branching with the weight of 17 *[* G_2 *: BE = {(*u_1, u_1*)},* G_1 *: BE = {(*u_1*, E), (E, F), (F, D)},* G_0 *: BE = {(C, E), (E, F), (F, D), (C, a), (B, C)}].* It may be noted that the final set of edges in *BE,* which define a maximum branching for G_0 , is in fact a single out tree rooted at *B* which does not, incidentally, include the edge of maximum weight G_0.

If a minimum branching rather then a maximum branching is required, it is easy to modify the maximum branching algorithm to find one. As has been described, this is achieved by simply replacing the weight of the edge by it's negative and then apply the algorithm. Obviously, a maximum branching for the graph with a modified edge (weights), provides a minimum branching for the original graph.

APPENDIX©

C.1. Quadratic Assignment Problem *(QAP)* Model

From Chapter 2 the problem of assigning new facilities to sites or location when there is an interchange between new facilities, is referred to as a quadratic assignment problem *(QAP)*. Kopmanns and Beckmann [1957] first introduced the *QAP*, which is a combinatorial optimisation problem. The problem involves assigning *n* facilities to *m* locations so that two total cost matrices can be formatted. One matrix gives the magnitude of the flow of materials between any two facilities whilst the other specifies the distance between any two locations.

Due to the complex nature of the *QAP*, algorithms for producing optimum solutions to the *QAP* have not been computationally feasible for the problems with dimension *n >15* facilities [Gilmore 1962, Nugent 1968]. Furthermore, the *QAP* of which the travelling salesman problem *(TSP)* is a special case, has been shown to belong to a special class of *NP* complete. These problems, for which it appears that there exists no efficient solution algorithm, are considered by Sahni [1976]. Foulds [1983] has suggested that *QAP* solution times are likely to be an exponential function of the problem size *n*. Francis [1992] formulated the *QAP* as follows:

$$Minimize\ Z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ik,jl}\, x_{ik}\, x_{jl} \qquad \textit{Equation C. 1}$$

which can be rewritten as:

$$Minimize\ Z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij}\, d(a(i),a(j))\quad x_{ik}\, x_{jl} \qquad \textit{Equation C. 2}$$

subject to :

$$\sum_{i=1}^{n} x_{ik} = 1\ , \qquad k = 1,\dots,n \qquad \textit{Equation C. 3}$$

$$\sum_{k=1}^{n} x_{ik} = 1\ , \qquad i = 1,\dots,n \qquad \textit{Equation C. 4}$$

$$x_{ij} = \begin{cases} 1, & \textit{if facility } i \textit{ is situated at site (location) } j \\ 0, & \textit{otherwise} \end{cases} \qquad \forall i,j = 1,\dots,n \qquad \textit{Equation C. 5}$$

where,

$c_{ik,jl}$ -Interaction cost of having facility *i* located at location *k* and facility *j* located at location

l, and $C_{ik,jl}$ is the product of flow volume times distance.

f_{ij} -The flow of material from facility i to facility j.

$d(a(i),a(j))$ -The distance between facility i at location $a(i)$ and facility j at location $a(j)$. (These are assumed rectilinear distances).

Notice that if facility i is at location k and facility j is at location l, then the decision variable x_{ij} is equal to 1, as shown in constraint (*Equation C.5*) and the cost term $c_{ik,jl}$ (or, $f_{ij}\, d(a(i),a(j))$) is included in the objective function. The constraint (*Equation C.3*) ensures that *exactly* one facility is assigned to each location, and the constraint (*Equation C.2*) ensures that each facility is assigned to *exactly* one location.

In addition to its application in facility location problems [Apple 1977, Francis 1992], the *QAP* has been found useful for such applications as production line scheduling, the backboard wiring problem in electronics, building layout problems, problem of minimising latency in magnetic drums on disc storage computers, and others.

Coupled with the fact that the *QAP* arises in such a variety of applications, many researchers have been drawn towards this class of problem because of the massive computational challenge that it possesses. Because of its combinatorial complex nature, interest has focused on developing efficient procedures to find least-cost solutions to the *QAP*.

C.1.1. Exact Solution Methods

With the exception of total enumeration, most exact algorithms are branch and bound, or implicit enumeration procedures [Gilmore 1962, Lawler 1963], and cutting plane algorithms [Bazaara 1982, Burkard 1984] (presented in more detail within Appendix B). These methods do find the least total cost. Furthermore, these branch and bound algorithms are single construction algorithms (ie. they build up a solution by adding an assignment at a time). Pair assignment algorithms [Gavett 1966] that add or exclude two assignments at a time have been found less useful. One exact solution method, the total enumeration approach, should be ruled out immediately as the size of n increases, since the total enumeration requires the explicit consideration of all $n!$ assignments.

Although branch and bound and cutting plane methods do guarantee an optimal solution to the *QAP*, their computational times depends largely on the hardware used. Both the speed of a computer and the computer memory capacity affect the outcome.

C.1.2. Integer Linear Programming Problems

In addition to the *QAP*, several integer linear programming formulations for the facility layout problem have been developed. Lawler [1963] was the first researcher to formulate the facility layout problem as a integer linear programming problem.

By defining: $y_{ijkl} = x_{ij}\, x_{kl}$ *Equation C. 6*

the *QAP* in these equations *(Equation C.8)*, *(Equation C.9)*, *(Equation C.10)*, *(Equation C.11)* can be represented as an integer linear programming problem, with the objective to:

$$Minimize \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c_{ik,jl}\, y_{ijkl} \qquad \text{Equation C. 7}$$

subject to:

$$\sum_{i=1}^{n} x_{ik} = 1 \,, \qquad k = 1,\ldots,n \qquad \text{Equation C. 8}$$

$$\sum_{k=1}^{n} x_{ik} = 1 \,, \qquad i = 1,\ldots,n \qquad \text{Equation C. 9}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} y_{ijkl} = n^2 \qquad \text{Equation C. 10}$$

$$x_{ij} + k_{kl} - 2y_{ijkl} \geq 0 \,, \qquad i, j, k, l = 1,\ldots,n \qquad \text{Equation C. 11}$$

$$x_{ij} \in \{0,1\}, \qquad i, j = 1,\ldots,n \qquad \text{Equation C. 12}$$

$$y_{iklj} \in \{0,1\}, \qquad i, j, k, l = 1,\ldots,n \qquad \text{Equation C. 13}$$

Lawler [1963] showed that the integer linear programming formulation and the *QAP* are equivalent. Love and Wong [1976] also proposed an integer programming formulation for the *QAP*. The integer linear programming solution techniques applied to the facility layout problems by both Lawler and Love are computationally infeasible, especially for problems with nine or more facilities [Kusiak 1987].

C.1.3. Heuristic Solution Methods

Since there has been limited success in finding exact solution methods for the *QAP*, attention has turned to finding better solutions using heuristics. Ham [1985] suggested that superior performance (in terms of solution quality and computation time) could be achieved by combining a construction method with an improvement procedure. Liggett [1981] contended that the improvement methods can be used to generate "good" initial solutions. Subsequently, heuristic solution approaches have been developed since the beginning of the 80's. Heuristic solution methods to the *QAP* do not guarantee optimal solutions, however their primary purpose is to find solutions near to the optimum. Their major advantage is that they are computationally feasible, even when $n > 15$. A number of heuristic procedures have been developed for solving the *QAP* [Bazaara 1982, Buffa 1965, Burkard 1984, Volenant 1987].

Another heuristic method by Scriabim [1985] consists of three stages. Firstly, it locates facilities on a floor plan as a scatter diagram such that the distance between the facilities are inversely related to the material flows. Then, the space constraints are taken into account for the facility allocation. Finally, a fine adjustment procedure is applied to obtain an acceptable solution. The heuristic approach by Kaku, Thompson, and Morton (*KTM*) [1991] also consisted of three parts. In the first part, several partial assignments are obtained by the development of a limited bread-first search tree. Serving as starting points for a construction procedure, these partial assignments are used to construct complete layouts in the second part. Finally, the third part attempts to improve the constructed layouts in the second part by applying triple and pairwise exchange routines. Computational experience of *KTM* shows that the performance of the heuristic as a stand alone solution method deteriorates as the problem gets large ie. *n >15*.

Although heuristic procedures have been found useful, they have a major shortcoming which is inherent in their very nature. It is never possible to tell, except by comparison with the answers provided by exact procedures, whether or not the final heuristic solutions are least-cost solutions. Sometimes, however, when a heuristic solution cannot be compared with the exact solution, then it has to be compared with other heuristic procedures under two major evaluate criteria. These criteria are the total cost of the final solution provided, and the computational time required for the final solution. Heuristic procedures resulting in a better solution often involve more computational effort.

C.1.3.1. Pairwise exchange heuristic

A pairwise exchange heuristic (only two machines at time are considered for exchange) is an improvement procedure where machines trade locations in an attempt to lower the cost for the layout [Francis 1992, Harmonsky 1992]. A starting layout is needed before a pairwise exchange procedure may be used. The pairwise exchange heuristic is as follows:

1. *Input starting layout.*
2. *Calculate total cost for the layout.*
3. *Initialise maximum cost savings for exchanging two machines to zero, and initialise the maximum cost savings for moving a machine to an unoccupied location to zero.*
4. *Calculate the cost savings for exchanging each possible pair machines. If the cost savings is greater than the maximum cost savings, set the maximum cost saving to the cost savings for this exchange and record machine pair to exchange.*
5. *If the shape is to be kept the same then go to step 7, otherwise, continue with step 7.*
6. *Scan all machines and calculate the cost saving by placing each machine in each unoccupied location. If the cost savings are greater than the maximum cost savings for a move, then set the maximum cost saving for a move to this cost saving, and machine numbers and new location are stored.*
7. *If the cost saving for both a move and an exchange are equal to zero, then go to step 11, otherwise, go to step 8.*
8. *If the cost saving for a move is greater than the cost saving for an exchange, then go to step 10, otherwise go to step 9.*
9. *Exchange two machines recorded in step 4 and calculate the total cost for the new layout. Go to step 3.*
10. *Move machines recorded in step 7 to locations recorded and calculate the total cost for the new layout. Go to step 3.*
11. *End*

In the next section the *SDPI* algorithm will be explained with a simple example.

C.1.3.2. Steepest-Descent Pairwise-Interchange method (SDPI)

The *SDPI* procedure is one of the best established and most easily understood heuristic solutions to the *QAP*. This procedure considers all possible pairwise interchange of facility locations (there are $n(n - 1)$ for an asymmetric matrix and $n(n - 1)/2$ for a symmetric matrix possible pairwise interchange). Then the pair, which causes the greatest decrease in the total cost, is interchanged, the assignment revised, and the process repeated until the total cost can be decreased no further. The *SDPI* heuristic does not guarantee optimal solution, however, since it only seeks local optima, it does not consider any higher order of interchange procedures.

This heuristic procedure finds a least cost assignment for a given initial assignment *'a'*, a distance matrix *D*, and weight matrix *W*. Thus if *a*, *D*, and *W* are given amongst all the pairwise interchanges of facility (machines) locations, the one that cause the greatest decrease in the total cost is found and this interchange made. The assignment is revised, and then the process is repeated until the total cost cannot be further decreased. The *SDPI* procedure is stated explicitly, given an initial assignment *a* and matrices *D* and *W*, in an algorithm as follows:

1. *Compute total cost of initial assignment 'a',*
2. *Set e to 0 (e is the maximum and the greatest decrease in total cost of initial assignment 'a' found so far for the given assignment),*
3. *Set i machine to 1 and j to 2 machine.*
4. *Compute interchanging the location cost of machines.*
5. *If interchanging the location cost of machines (assignment 'a') is greater then e, go to step 6; otherwise, go to step 7.*
6. *Set e to interchanging the location cost of the machines (assignment 'a'), set u to i and set v to j.*
7. *If j = n, go to step 8; otherwise go to step 10.*
8. *If n = n – 1, go to step 11; otherwise go to step 4.*
9. *Increment j by 1 and go to step 4.*
10. *Increment i by 1, let j = i + 1 and go to step 4.*
11. *If e is positive, go to step 12; otherwise go to step 14.*
12. *Replace total cost of 'a' by total cost of a – e.*
13. *Revise the assignment 'a' by interchanging the location of machines u and v and go to step 2.*
14. *Stop.*

The next paragraph explains how the *SDPI* algorithm works.

The six machines to be located on the shop floor are as shown in *Figure C.1.a.* Then for a given assignment of machines *a = (2, 4, 5, 3, 1, 6)* for six site-locations on the shop floor as shown in *Figure C.1.b,* with distance matrix *D* shown in *Table C.1* and weight *W* matrix shown in *Table C.2.* The distances between locations are assumed to be the rectilinear distances, ie. measured units of location widths, between centers of locations.

1	2	3
4	5	6

a)

⑤	①	④
②	③	⑥

b)

Figure C. 1 - Examples of a) locations and b) initial assignment of machines to locations

Sites

	D	*1*	2	*3*	4	*5*	6
Sites	*1*	0	1	2	1	2	3
	2	1	0	1	2	1	2
	3	2	1	0	3	2	1
	4	1	2	3	0	1	2
	5	2	1	2	1	0	1
	6	3	2	1	2	1	0

Table C. 1 - Distance matrix

The total cost for the assignment shown in *Figure C.1.b,* using the *W* matrix from *Table C.2,* is 114. When the procedure (*SDPI* algorithm) is applied to the example, for the given assignment (2, 4, 5, 3, 1, 6), the values shown in *Table C.1 and C.2* are computed, and the locations of the machines 2 and 6 are interchanged, resulting in the assignment (2, 6, 5, 3, 1, 4) with a corresponding total cost of 98. When the algorithm is applied a second time, given assignment *a = (2, 6, 5, 3, 1, 4),* the exchange remains zero and so the procedure stops; thus for the machines: 1 is in location 2; 2 is in 6; 3 is in 5; 4 is in 1, and 6 is in 4. The final assignment is shown in *Figure C.2.*

		Machines					
	W	*1*	2	*3*	*4*	*5*	*6*
Machines	*1*	0	4	6	2	4	4
	2	4	0	4	2	2	8
	3	6	4	0	2	2	6
	4	2	2	2	0	6	2
	5	4	2	2	6	0	10
	6	4	8	6	2	10	0

Table C. 2 - Weight matrix

⑤	①	④
⑥	③	②

Figure C. 2 - Final assignment of machine locations resulting from applying the SDPI algorithm

A concluding remark made to date [Karisch 1998] is that the *SDPI* procedure is the best procedure available for solving the *QAP* problem. Also Foulds [1998] reported the high computational time needed for solving integer programming, describing the *QAP* problem (5000sec seconds on Hewlett-Packard 900) without the assurance of finding an optimal solution. These problems were two dimensional *QAP,* and even when they restricted the problem to a one dimensional *QAP,* the problems were difficult to solve using the integer programming formulation, with poor computational results being obtained for such a model. When they modified the model to apply to a more restricted area, ie. a simpler formulation with up to three facilities per block (cell), computational difficulty occurred again in solving the *QAP.* Further details about this modified model are given by Foulds and Wilson [1995].

However, results from previous sections, indicate that only a small size one dimensional problem can be solved using a formulation which restricts the placement of facilities into blocks, with high computational time experienced when the problem was formulated in an integer programming form.

These conclusions from Foulds [1998, 1995] research results however were unavailable when the mathematical modelling of the *DCM* layout was undertaken. However Karisch's [1998] analysis states that the *SDPI* is still the best algorithm for solving the *QAP* assignment and the *NP*-problem (outperforming *SA* and *EA).* Experience from this current research work still confirmed the use of the *SDPI* algorithm with integer programming (branch and bound method) to improve the proposed methodology for the second stage solution. Also, because of the nature of the placement problem these remarks justify the decision to employ *SA* for solving the automatic placement of the *DCM* layout in stages III and IV.

C.2. Travelling Salesman Problem

C.2.1. Introduction

The *Travelling Salesman Problem* (*TSP*) consists of conducting the best possible tour amongst a given set of cities. This problem seems to be a very simple one at first, but on looking at it with a bit more attention and detail we find it to be very complex indeed. Thus in the *TSP* for a given set of cities there is for each pair of distinct cities a particular distance. The goal is to find an ordering of the cities that minimises the tour length, where the length of the tour a salesman would make when visiting the cities, returning at the end to the initial city, is given in the order specified by the permutation.

Consider a set of *n* cities. The travelling salesman must start from city one (any of the *n* cities) and travel to all of the cities exactly once and then return to the first city, for a tour to qualify as valid. Therefore, a tour is a *Hamiltonian Cycle* in a graph with '*n*' nodes. A Hamiltonian Cycle is a path in a graph which connects all the nodes in the graph exactly once and returns to the starting node. Now the problem is to find the shortest (or best) path of all the possible paths. The cities may be separated by distances, or their maybe different costings to travel from one city to another. It can be viewed as a *Complete Weighted Graph* for which it is required to find the *Shortest Hamiltonian*.

The *TSP* has many applications, from *VLSI* chip fabrication [Korte, 1988] to X-ray crystallography [Bland and Shallcross, 1989]. For a detailed history Lawler [1985] may be referenced. The *TSP* is *NP* difficult [Garey, 1989] and so any algorithm for finding optimal tours must have a worst-case running time that grows faster than any polynomial equation (assuming the widely believed conjecture that $P \neq NP$). This leaves researchers with two alternatives: either looking for heuristics that merely find near optimal tours, but do so quickly, or attempt to develop optimisation algorithms that work well in the "real-world".

An algorithmic complexity of the *TSP* is to get the solution of the *TSP* by examining all possible tours and selecting the best. However, this would mean that we would be examining *n!* possibilities, which means an algorithmic complexity of *O(N!)*, so although we would be getting our "best" it would not necessarily be imminent.

Another way in which the *TSP* may be atypical lies in the high quality of results that can be obtained by traditional heuristic's. In addition to the local search approaches, there are many different successive augmentation heuristics for the *TSP*. Such heuristics build a solution (tour) from scratch by a growth process (usually a greedy one) that terminates as soon as a feasible solution has been constructed. In the context of the *TSP*, we call such a heuristic a tour construction heuristic. Whereas the successive augmentation approach performs poorly for many combinatorial optimisation problems, in the case of the *TSP* many tour construction heuristics do surprisingly well in practice. The best typically get within roughly 10-15% of optimal in relatively little time. Furthermore, "classical" local optimisation techniques for the *TSP* yield even better results, with the simple 2-Opt heuristic typically getting within 3-4% of the optimal and the algorithm of Lin and Kernighan [1973] typically getting within 1- 2%.

Moreover, for geometric data the above mentioned algorithms all appear to have running time growth rates that are $O(N^2)$, ie., subquadratic, at least in the range from 100 to 1,000,000 cities. These successes for traditional approaches leave less room for new approaches like tabu search, simulated annealing, genetic algorithms etc. to make meaningful contributions.

C.2.1.1. The Problem and its Importance

If a salesman, starting from his home city, is to visit exactly once each city on a given list and then return home, it is plausible for him to select the order in which he visits the cities so that the total of the distances travelled in his tour is as small as possible. This is assuming that the salesman knows, for each pair of cities, the distance from one to the other. Then he has all the data necessary to find the minimum, but it is by no means obvious how to use this data in order to get the answer.

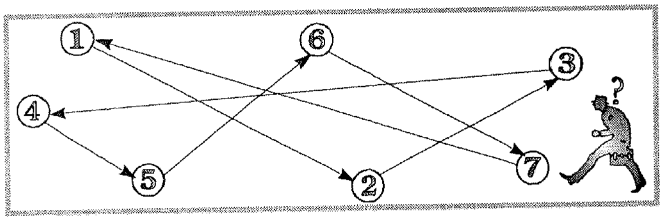


Figure C. 3 - Showing that the salesman must decide which tour will allow him to minimise travel cost

The importance of the *TSP* comes not from the wealth of applications, since the number of cases where the mathematical model of the *TSP* precisely fits an engineering or scientific situation have not to date been numerous, but from the fact that it is typical of other problems of its type ie. *combinatorial optimisation,* which are also trying to minimise the total distance. Thus, the problem is one of optimisation. Nevertheless, one cannot immediately employ the methods of differential calculus by setting derivatives to zero, because it's a combinatorial situation: choice is not over a continuum but over the set of all tours.

By the late 1960's it was appreciated that there appeared to be a significant difference between hard problems such as the *TSP*, for which the only available optimisation algorithms were of an enumerative nature, and other problems for which *good* algorithms existed. *Good* is a term which was coined by Edmonds [1965] to describe solution methods whose running time increases polynomially with problem size. It was empirically concluded that the computational effort demanded by problems like the *TSP* for their solution would grow as a super polynomial with problem size. Insight emerged that many of the problems suspected to be inherently hard are all computationally equivalent. Thus, in a sense, a polynomial time algorithm for one of them could also be used to solve all others in polynomial time. These problems are called *NP*-hard. From the discussion above it is clear that not only is the *TSP* important for its real world applications but also for the rest of the *NP*-hard problems.

C.2.2. Methods for Solving the *TSP*

One of the more practical consequences of the *NP*-hardness of a problem is that it limits the choice to three solution strategies. To start with, one may not accept the apparent difficulty of the problem at hand and try to find some special structure that places it in a well-solved subclass. If that does not work out, there is no guarantee that an optimal solution can be found in a reasonable amount of time. To compromise on either of the two dimensions one can insist on the optimality of the solution (and risk spending a lot of time), or insist on a fast solution method (and accept the possibility of a suboptimal solution). The use of *cutting planes* and *branch and bound* and their combinations have impressive algorithmic consequences. Another enumerative optimisation method, the recursive technique of dynamic programming, works as follows:

The Dynamic Programming algorithm for the N city problem finds, for each i, the shortest path from city 1 to city i, that visits all the other nodes in $\{2, 3... N\}$. Once these paths are found, it is a simple matter to compute the shortest tour. To find these paths, the algorithm solves a more general problem. Thus for any $S \subseteq \{2, \dots, N\}$ and $i \in$ S, let an (S, i) path be a path which starts at city 1, visits each city in S exactly once, and no other city, and ends up in the city i. Let Cost $[S, i]$ stand for the length of the shortest (S, i) path. Then Cost $[S, i]$, where $S \geq 2$, satisfies the following equation:

$$Cost[S, i] \;=\; min\,\{Cost[S - \{i\}, k] + c_{ki}\}. \qquad\qquad K \in S\text{-}\{\,i\,\}$$

The algorithm described above 'builds up' the values of *Cost* $[S, i]$ for larger and larger sets of S until Cost $[\{2,3, \dots, N\}, i]$ is obtained. Thus dynamic programming works by putting together solutions of bigger and bigger subproblems. The estimated running time of the above described method is $O(n^2 2^n)$.

The Greedy algorithm travels from the present city to the next nearest city, which has not yet been visited. This seems to give a very 'nice' and close to optimal solution, and simple observation would give the running time of this method $O(n^2)$. This is very fast, but does not necessarily give the best solution as demonstrated in the diagram below.

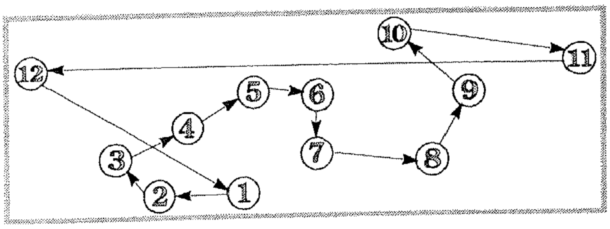


Figure C. 4 - Solution of 13 City Problem by Greedy Algorithm

Function	***Approximate Values***		
N	*10*	*100*	*1,000*
NlogN	*33*	*664*	*9,966*
N^3	*1,000*	*1,000,000*	10^9
10^6N^8	10^{14}	10^{22}	10^{30}
2^N	*1,024*	$1.27\ 10^{30}$	$1.05\ 10^{301}$
N^{logN}	*2,099*	$1.93\ 10^{13}$	$7.89\ 10^{29}$
N!	*3,628,800*	10^{158}	$4\ 10^{2567}$

Table C. 3 - Time complexities and number of calculations

From the diagram above it is clear that even after following the greedy algorithm the best possible tour was not achieved. By simple inspection, if the routes had been *9-11-10-12*, or (*4-12-5* and *7-10-11-9-8-1*), both these tours would have yielded better results than the one demonstrated. The greedy algorithm of course falls in the second option, ie. speed is preferred over optimality. In fact, this second option of *approximation* is the one most frequently chosen in practice.

As seen in *Table C.3* the execution time, on the left-most column, gives the time complexities. It is clear from the table that while functions with *O(N)* increase linearly, *O(N!)* functions have a very visibly astronomical growth with increase in elements. It is now very clear that a polynomial time complexity will be much more acceptable (provided it gives a reasonable accuracy), than a 2^N or a worse *N!* order. Consider for example 20 elements. If the time complexity is *N!,* then the solution on one of the most powerful computers, would take a few thousand years, which is not at all acceptable. This only shows how important it is to take into consideration the time complexity of the considered algorithm.

C.2.2.1. Heuristic Methods

The *NP*-Hardness of the *TSP*, as discussed above, makes it unlikely that any efficient algorithm can be guaranteed to find optimal tours when the number of cities is large. Design of 'heuristic' algorithms, while not guaranteed to find optimal tours, do find what one hopes are '*near-optimal*' tours. There is the possibility of polynomial-time heuristics that provide good guarantees for all *TSP* instances, but this is only in the unlikely event of *P = NP*.

C.2.2.1.1. Heuristics for Tour Construction

Every *TSP* heuristic can be evaluated in terms of two key parameters: its running time and the quality of the tours that it produces. Presented here is the most important undominated heuristics, where a heuristic is undominated if no competing heuristic finds better tours and runs more quickly, and four heuristics that only work for two-dimensional classes. The four tour construction heuristics covered in detail are Nearest Neighbour, Greedy, Clarke-Wright, and Christofides. Each of these has a particular significance in the context of a local search.

The first three provide plausible mechanisms for generating starting tours in a local search procedure, and interesting lessons can be learned by evaluating them in this context. The fourth represents in a sense the best that tour construction heuristics can currently do, and so it is a valuable benchmark. Readers interested in the full picture are referred to more extensive studies such as those of Bentley [1990a, 1992], Reinelt [1994], Junger, Reinelt, and Rinaldi [1994], and Johnson and Rothberg [1996].

The Clarke and Wright Savings Algorithm [1964] is as follows. First select any node as the central depot, and denote it as node one. Compute savings by:

$$S_{ij} = C_{ii} + C_{ij} - C_{ij} \; for\ i, j = 2,3, \ldots , n.$$

Order the savings from largest to the smallest. Starting at the top of the savings list and moving downwards, form larger sub-tours by linking appropriate nodes *i* and *j*. Repeat this process until a tour is formed. The number of computations is about $O(N^2 logN)$.

Perhaps the most natural heuristic for the *TSP* is the famous ***Nearest Neighbour*** algorithm *(NN)*. In this algorithm one mimics the traveller whose rule of thumb is always to go next to the nearest as yet unvisited location. Now, to construct an ordering *c p(1),..., c p(N)* of the cities, with the initial city *c p(1)* chosen arbitrarily, in general *c p(i + 1)* is chosen to be the city c_k that minimises $\{d(c\ p(i)\ ,\ c_k\ ) : k =/$

p(j)}. The corresponding tour traverses the cities in the constructed order, returning to *c p(1)* after visiting city *c p(N)* . The running time for *NN* as described is *Q(N 2)*. No substantially better guarantee is possible, however, there are instances for which the ratio grows as *Q(logN)* [Rosenkrantz, Stearns, and Lewis, 1977].

Some authors use the name ***Greedy*** for Nearest Neighbour, but it is more appropriately reserved for the following special case of the "greedy algorithm" of matroid theory. In this heuristic, an instance as a complete graph with the cities as vertices and with an edge of length $d(c_i , c_j)$ between each pair $\{c_i , c_j \}$ of cities is viewed. A tour is then simply a Hamiltonian cycle in this graph, ie, a connected collection of edges in which every city has degree two. To build up this cycle one edge at a time, the procedure starts with the shortest edge, repeatedly adding the shortest remaining available edge, where an edge is available if it is not yet in the tour, and if adding it would not create a degree three vertex or a cycle of length less than *N*. The Greedy heuristic can be implemented to run in time *Q(N 2 logN)* and is thus somewhat slower than *NN* [Ong and Moore, 1984].

The Insertion Procedure for tour construction takes a sub-tour on *k* nodes at iteration *k* and attempts to determine which node, not in the sub-tour, should join the next subtour (the selection step) and then determines where it should be inserted in the subtour (the insertion step).

The Nearest Insertion starts with a subgraph consisting of a node *x* only, then finds a node *y* such that c_{xy} is minimal and forms sub-tour *x-y-x*. The selection step, given a sub-tour, finds node *y* which is not in the sub-tour closest to any node in the sub-tour. The insertion step finds the *arc(x,z)* in the sub-tour which minimizes $c_{xy} + c_{yz} - c_{xz}$, then inserts *y* between *x* and *z*. The selection and insertion is repeated until a Hamiltonian cycle is obtained. The number of computations in the nearest insertion algorithm is $O(N^2 logN)$.

The next method is the ***Arbitrary Insertion***. Here again a subgraph is started consisting of *x* only. Then node *y* is located such that c_{xy} is minimal and forms subtour *x-y-x*. Finally node *z* is arbitrarily select, which is not in the sub-tour, to enter the sub-tour. The complexity is $O(N^2)$.

The Cheapest Insertion starts with a subgraph consisting of *x* only and finds node *y* such that c_{xy} is minimal and forms sub-tour *x-y-x*. Then finds *(x, z)* in subtour and *y*, such that $c_{xy}+c_{yz}-c_{xz}$ is minimal, then inserts *y* between *x* and *z*. The number of computations required is $O(N^2 logN)$.

The Farthest Insertion starts with a subgraph containing only *x* and finds node *y* such that c_{xy} is minimal and forms sub-tour *x-y-x*. Then given a subtour, finds node *z* which is not in the subtour, and is farthest from any node in the sub-tour. This again requires $O(N^2)$ computations.

The final method for tour construction considered here is the ***Christofide's Heuristic*** [1976], which begins by finding the minimum spanning tree *T* of graph *G*, then identifies all the odd degree nodes using the original cost matrix. The branches from the matching solution are added to the branches already in *T*, obtaining a *Euler* cycle. In this sub-graph every node is of even degree, although some nodes may have a degree greater than *2*. The next step is to remove the polygons over the nodes with degree greater than *2*, and transform the Euler cycle into a Hamiltonian cycle. The number of computations in this procedure is $O(N^3)$. In most cases, the number of odd cases will be considerably less than *n* [Edmonds 1965, Gabow 1973, and Lawler 1976]. A modification of the Christofides algorithm with the same worst-case guarantee and an $O(N^{2.5})$ running time, can be obtained by using a matching algorithm and halting once the matching process is achieved. This is guaranteed to be no longer than $1 + (1/N)$ times optimal [Gabow 1991]. However, this approach has never been implemented, and the competition from local search algorithms is sufficiently strong that the programming effort needed to do so would not be justified.

Summarised results obtained by Johnson and Rothberg [1996] for the four tour construction heuristics on random Euclidean instances and random distance matrices are as follows:

1. Results for instances from the *TSPLIB* were similar to those for the random Euclidean instances.
2. Running times for instances of similar size were comparable with the tour quality for *NN*.
3. Greedy being slightly worse on average than for the random instances.
4. The tour quality for Clarke and Wright, and Christofides being slightly better.

C.2.2.1.2. Heuristics for Tour Improvements

Tour improvement procedures includes branch exchange heuristics. Branch exchange heuristics work as follows:

- First, find an initial tour, generally this tour is chosen randomly from the set of all possible tours.
- Improve the tour using one of the branch exchange heuristics.
- Continue trying to improve by the branch exchange heuristic, until an additional improvement can be made, at which the branch exchange procedure terminates at a local optimum.
- For a given *k*, define a *k*-change of a tour as consisting of the deletion of *k* branches in a tour and their replacement by *k* other branches to form a new tour, then a tour is *k*-optimal if it is not possible to improve the tour via a *k*-change.

The following aspects are noted:

- The final two steps in the above procedure will generate this *k*-optimal tour.
- Since the 2-opt exchange procedure is weaker than the 3-opt exchange procedure, it will generally terminate at an inferior local optimum.
- Similarly, a *k*-opt exchange procedure will generally terminate with a better local optimum than will a 3-opt exchange procedure.

C.2.2.1.3. Other Heuristic Methods

The ***Composite Procedure*** can be stated as follows: Obtain an initial tour using one of the tour construction procedures. Apply a 2-opt procedure to this tour. Then apply a 3-opt procedure to the new tour. This composite method is found to be relatively fast computationally and gives exceptional results.

The idea behind the composite procedure is to get a good initial solution rapidly and hope that the 2-opt and 3-opt procedures will then find an almost optimal solution. In this way, the 3-opt procedure, which is the computationally most expensive of the three, need be used only once. There can, of course, be many possible variants to the above procedure, some of which are now discussed. One variant would be to run this procedure without the 2-opt procedure. The next would be to run it without the 3-opt procedure, this one in particular will give very fast running times and very accurate results but would not expect to out-perform the basic procedure discussed previously. However, one could also run the procedure a few times, using different tour construction algorithms every time. This variant does give better results than the basic procedure.

Next is the ***Local Search*** algorithm. The algorithm begins with a notion of a neighbourhood structure of the set of all feasible solutions (tours). Then define the neighbourhood of a tour T to be all those tours that can be obtained by changing at most k edges of T. One can search for local k-opt tours by starting with a random tour *T1* and constructing a sequence of tours *T1, T2, ...* Each tour is obtained from the previous one by performing a k-change, ie. by deleting k links and reconnecting the loose ends so as to still have a legal tour. The k-changes are required to decrease the length of the tour. When the process stops at a tour for which there is no possible improvement under a k-change, the tour is k-opt. In order to find the globally optimal tour, we have to repeat the search for many other random starts.

C.2.2.1.4. Some Modern Heuristics

Simulated Annealing is used to approximate the solution of very large combinatorial optimisation problems. Here it is required to find the configuration that minimises a certain cost function $f(x)$. The algorithm can then be formulated as follows:

Starting off at an initial configuration, a sequence of iterations is generated. Each iteration consists of the random selection of a configuration from the neighbourhood of the current configuration, and the calculation of the corresponding change in cost function ΔE. The neighbourhood is defined by the choice of a generation mechanism, ie. a "prescription", to generate a transition from one configuration into another by a small perturbation. If the change in cost function is negative, the transition is unconditionally accepted. If on the other hand the cost function increases, the transition is accepted with a probability based upon the *Boltzmann Distribution*: $P = e^{-\Delta E/T}$, where T is the current "temperature" - the control parameter. This temperature is gradually lowered throughout the algorithm, from a sufficiently high starting value, to a "freezing" temperature, where no further changes occur. In practice, the temperature is decreased in stages, and at each stage the temperature is kept constant until the thermal quasi-equilibrium is reached. In order to apply the Simulated Annealing algorithm one must decide in advance the following: the initial temperature, the termination condition, the reduction function α and the stopping condition.

The Tabu Search is an iterative procedure for solving discrete combinatorial optimisation problems. It was first suggested by Glover [1990] and since then has become increasingly used. It has been successfully applied to obtain optimal or sub-optimal solutions to scheduling, timetabling, lay out optimisation and of course, the *TSP*.

The basic idea of this method is to explore the search space of all feasible solutions by a sequence of moves. A move from one solution to another is the best available. However, to escape from locally optimal but not globally optimal solutions and to prevent cycling, some moves, at one particular iteration, are classified as forbidden or tabu (or taboo). Tabu moves are based on the short-term and long-term history of the sequence of moves. A simple implementation, for example, might classify a move as tabu if the reverse move has been made recently or frequently. Sometimes, when it is deemed favourable, a tabu move can be overridden. Such aspiration criteria might include the case which, by forgetting that a move is tabu, leads to a solution that is the best obtained so far. Halting when a certain threshold for an acceptable solution has been achieved, or when a certain number of iterations have been completed, may be employed to solve sub-optimal problems.

C.2.2.2. Other Methods for the TSP

Presented in this section are some other earlier methods that were developed and used to solve the *TSP*.

C.2.2.2.1. Integer Programming

As an introduction for *S* (path), let *S'* be a partition of the integers $q = 1, 2, \ldots, n$; *ie.* $S \cap S' = \varnothing$ and $S \cup S' = \{1,2,\ldots,n\}$. For symmetric distances let $x_{pq} = 0$ if the undirected arc (p, q) is not in a tour and $x_{pq} = 1$ if the undirected arc (p, q) is in a tour. An optimal tour can be found by solving the integer program:

$$min\ Z = \sum \begin{pmatrix} q = n \\ q = 2 \end{pmatrix} \sum \begin{pmatrix} q - 1 \\ p = 1 \end{pmatrix} c_{pq}\, x_{pq}$$

subject to: $x_{pq} = 0, 1, (p= 1,\ldots, q\text{-}1;\ q= 2,\ldots, n)$.

The difficulties in finding an optimal tour in solving the integer program of this theorem are the enormous number of loop constraints $(2^{n-1} - 1)$ and the requirement that the $(n^3 - n)/2$ variables x_{pq} equal 0 or 1 for symmetric distances. The solution of a linear program with the loop constraints and $0 \leq x_{pq} \leq 1$ generally will not satisfy $x_{pq} = 0$ or 1.

However, in 1954 an optimal solution to a 42-city problem was found using this formulation. A large number of loop constraints were overcome, beginning with only a few, and then adding new ones only as they were needed to block sub-tours. Combinatorial arguments were used to eliminate fractional solutions and to find an optimal tour. Finally, it was demonstrated that for the problem at hand, an ordinary linear program could be devised whose solution gave integer valued x_{pq}'s representing the optimal tour. The constraints that rule out some fractional solutions, but no integer solutions, were forerunners to the 'cutting plane' constraints for solving any integer linear program.

C.2.2.2.2. Branch and Bound

Enumerative (branch and bound, implicit enumeration) methods solve a discrete optimisation problem by breaking up its feasible set in to successively smaller subsets, calculating bounds of the objective function value over each subset, and using them to discard certain subsets from further consideration. The bounds are obtained by replacing the problem over a given subset with an earlier (relaxed) problem, such that the solution value of the latter bounds equal that of the former. The procedure ends when each subset has either produced a feasible solution, or has been shown to contain no better solution than the one already in hand. The best solution found during the procedure is a global optimum. The outline for two versions of a branch and bound procedure for the *TSP* is presented as follows:

Prior to using any of these versions, a relaxation *R* of the *TSP* must be chosen. Both versions carry at all times a list of active subproblems. They differ in that the first version solves a relaxed subproblem R_k only when node *k* is selected and taken off the list, whilst the second version solves each relaxed subproblem as soon as it is created, ie. before it is placed on the list. Although the branch and bound procedures used in practice differ among themselves in many details, nevertheless all of them can be viewed as variants of one of these two versions. The first version of the branch-and-bound for the *TSP* is as follows:

1. *Initialisation: Put TSP on the list (of active subproblems). Initialise the upper bound at $U = \infty$.*
2. *Sub-problem Selection: If the list is empty, stop; the tour associated with U is optimal (or, if $U = \infty$, TSP has no solution). Otherwise choose a subproblem TSP_k according to the sub-problem selection rule and remove TSP_k from the list.*
3. *Lower Bounding: Solve the relaxation R_k of TSP_k or bound $v(R_k)$ from below, and let L_k be the value obtained.*
4. *If $L_k \geq U$, return to the Sub-problem selection step.*
5. *If $L_k < U$ and the solution defines a tour for TSP, store it in place of the previous best tour, set $U \Leftarrow L_k$ and go to the Optional Reduction Step below(Note, $L_k < U$ and the solution does not define a tour).*
6. *Optional Upper Bounding: Use heuristic to find a tour for TSP. If a better tour is found than the current best, store it in place of the latter and update U.*
7. *Optional Reduction: Remove from the graph of TSP_k all the arcs whose inclusion in a tour would raise its value above U.*
8. *Branching: Apply the branching rule to TSP_k, ie. generate new sub-problems $TSP_{k1}, \ldots, TSP_{kn}$, place them on the list and go back to the Sub-problem selection step.*

Second version is as follows:

1. *Initialisation: As in the previous version, but solve R before putting TSP on the list.*
2. *Sub-Problem Initialisation: Same as in the version above.*
3. *Optional Upper Bounding: Same as in the version above.*
4. *Optional Reduction: Same as in the previous version.*
5. *Branching: Use the branching rule to define the set of sub-problems $TSP_{k1}, \ldots, TSP_{kn}$ to be generated from the current sub-problem TSP_k.*
6. *Lower Bounding: If all the sub-problems to be generated from TSP_k according to the branching rule have already been generated, go back to the Sub-problem selection step. Otherwise generate the next sub-problem TSP_{kl} defined by the branching rule, solve the relaxation R_{kl} of TSP_{kl} or bound of $v(R_{kl})$ from below, and let L_{kl} be obtained.*

   If $L_{kl} \geq U$, repeat this step.

   If $L_{kl} < U$ and the solution defines a tour for TSP, store it in place of the previous best tour, set $U \Leftarrow L_{kl}$ and repeat this step.

   If $L_{kl} < U$ and the solution does not define a tour, place TSP_{kl} in the list and repeat this step.

In both versions the procedure can be represented by a rooted tree (search tree or branch and bound tree) whose nodes correspond to the sub-problems generated, with the root node corresponding to the original problem, and the successor nodes of a given node *i* associated with TSP_k corresponding to the sub-problems TSP_{k1} , ..., TSP_{kn} defined by the branching rule.

C.3. Solving *TSP* by Simulated Annealing

The invention of simulated annealing actually preceded that of tabu search, like tabu search simulated annealing allows uphill moves. However, whereas the tabu search in essence only makes uphill moves when it is stuck in local optima, simulated annealing can make uphill moves at any time. Moreover, simulated annealing relies heavily on randomisation, whereas the tabu search in its basic form chooses its next move in a strictly deterministic manner. Nevertheless, simulated annealing is still basically a local search algorithm, with the current solution wandering from neighbour to neighbour as the computation proceeds. The key difference from other approaches is that simulated annealing examines neighbours in random order, moving to the first one seen that is either better or else passes a special randomised test. As originally proposed by Kirkpatrick [1983] and Cerny [1985], the randomised test is the one invented by Metropolis [1953] for simulating the physical behaviour of atoms in a heat bath. It involves a control parameter called the temperature, and in simulated annealing that control parameter is continually lowered as the search proceeds in a simulation of the physical annealing process.

The *TSP* was one of the first problems to which simulated annealing was applied, serving as an example for both Kirkpatrick [1983] and Cerny [1985]. Since then the *TSP* has continued to be a prime test for the approach and its variants. In this section, consideration is given to the resulting *TSP* algorithms and how they perform. Most adaptations have been based on the simple schema presented in *Figure C.5*, with implementations differing as to their methods for generating starting solutions (tours) and for handling temperatures, as well as in their definitions of equilibrium, frozen, neighbor, and random. Note that the test in *Step 3.1.4* is designed so that large steps uphill are unlikely to be taken except at high temperatures *T*. The probability that an uphill move for a given cost *D* will be accepted, declines as the temperature is lowered. In the limiting case, when $T = 0$, the algorithm reduces to a randomised version of iterative improvement, where no uphill moves are allowed at all.

```
1. Generate a starting solution S and set the initial champion solution $S^* = S$.
2. Determine a starting temperature T.
3. While not yet frozen do the following:
        3.1. While not yet at equilibrium for this temperature, do the following:
        3.1.1. Choose a random neighbour S' of the current solution.
        3.1.2. Set $D = Length(S') - Length(S)$.
        3.1.3. If $D > 0$ (downhill move):
                Set $S = S'$.
                If $Length(S) < Length(S^*)$, set $S^* = S$.
        3.1.4 Else (uphill move):
                Choose a random number r uniformly from [0 , 1].
                If $r < e - D/T$, set $S = S'$.
        3.1.5 End "While not yet at equilibrium" loop.
        3.2 Lower the temperature T.
        3.3 End "While not yet frozen" loop.
    4. Return S*.
```

Figure C. 5 - General schema for a simulated annealing algorithm.

From the beginning, there has been a dichotomy between the way the schema of *Figure C.5* is implemented in practice and in theory. In theory, simulated annealing can be viewed as an optimisation algorithm. The process can be interpreted in terms of Markov chains and proved to converge to an optimal solution if one ensures that the temperature drops no more quickly than $C/\log n$, where C is a constant and n is the number of steps taken so far. Typically, however, the convergence to an optimal solution under such a temperature schedule will take longer than finding such a solution by an exhaustive search. Thus, such theoretical results are essentially irrelevant to what can be accomplished in practice.

Instead, starting with Kirkpatrick [1983] and Cerny [1985], researchers have tended to use cooling schedules that drop the temperatures much more rapidly, say roughly as C^n , where $C < 1$. This can be realised for instance by performing a fixed number of trials at each temperature, after which one arbitrarily declares "equilibrium" and reduces the temperature by a standard factor, say 0.95. Under such an exponential cooling regime the temperature will, after a polynomially bounded amount of time, reach values sufficiently close to zero that uphill moves will no longer be accepted and can be cleared for freezing to have set in. This happens even when, as Kirkpatrick originally suggested, one starts at a temperature T that is sufficiently high that essentially all uphill moves are accepted. With such a polynomially bounded cooling schedule, simulated annealing is only an approximation algorithm (like all our other local search variants), but it would be hard to expect more from *SA* for an *NP-hard* problem like the *TSP*. Theory so far has little to say about this polynomial-time bounded version of simulated annealing, except in the context of specially invented problems and neighbourhood structures. Much of the comments is derived from the study of Johnson [1996], to which the reader is referred for additional technical details.

The next section presents a baseline implementation similar to that of the original Kirkpatrick paper and reports on its behaviour. Section C.3.2 describes and evaluates two key ideas, neighbourhood pruning and low temperature starts. These provide more than constant factor speedups and are essential if annealing is to be competitive with the more traditional *TSP* heuristics.

C.3.1. A Baseline Implementation of Simulated Annealing for the *TSP*

In adapting simulated annealing to the *TSP*, both Kirkpatrick [1983] and Cerny [1985] suggested using a neighbourhood structure based on 2-Opt moves, just as was later done for tabu search. Cerny [1985] also considered the simpler move in which the positions of two cities are interchanged, but the segment between them is left unchanged. However experiments demonstrated that this was not as an effective approach. The results in these two original papers were unfortunately limited mainly to small examples and running times were not reported. Kirkpatrick [1983] did run the algorithm on one problem of reasonable size (some 6000 cities), but they did not provide any detailed information on the quality of the solution found, other than that, it was "good". Thus, the value of simulated annealing for the *TSP* was initially unclear.

There does, however, appear to be a serious defect in the above straightforward approach to adapting simulated annealing to the *TSP*. As will be seen, the number of steps at each temperature (called the temperature length) needs to be at least proportional to the neighbourhood size if a worthwhile tour quality is to be contained. This is not too onerous a restriction for problems like graph partitioning, where typical neighbourhood sizes are *O(N)* (Kirkpatrick [1983], Johnson [1989]). For the 2-Opt *TSP* neighbourhood, however, the size is proportional to N^2 , so that even if the number of distinct temperatures considered does not grow with *N*, an algorithm is still available whose running time is at least $O(N^2)$ with a large constant of proportionality.

In summary, from Johnson, Aragon and Schevon [1996], this baseline implementation produces worse tours than 3-Opt on average, whilst taking almost 300 times as long when $N = 100$, and over 7500 times as long when $N = 1000$. With even more annealing time, however, better results can be obtained. Increasing running time by a factor of 10 (by increasing temperature length to $10N(N - 1)$) reduces the average final percentage excess for $N = 100$ from 3.4% to 1.9%; significantly better than 3-Opt (but taking 3000 times as long).

Thus if simulated annealing is to be useful for the *TSP*, more ways are required to get the effect of longer temperature lengths whilst somehow simultaneously reducing the overall running time substantially. It is intuitively clear that speeding up the algorithm by simply reducing the temperature length (or using fewer temperatures) can only make the average tour length worse. Early follow-ups on Kirkpatrick [1983] that in effect took this approach, such as Nahar, Sahni, and Shragowitz [1984] and Golden and Skiscim [1986], bear this out. Nahar et. al restricted their runs to just six temperatures (obtained using a reduction factor of 0.90) and adjusted the temperature length. Golden and Skiscim used 25 temperatures (evenly rather than geometrically spaced) as well as an adaptive temperature length that varied from temperature to temperature, but whose average value was not likely to grow as N^2. In so far as comparisons can be made, both resulting algorithms appear to have found worse tours than those found by *SA*, whilst still remaining significantly slower than the neighbour list implementations of 3-Opt and Lin - Kernighan [1990].

C.3.2. Improvement or Speed - Up Techniques

Effective speed - up approaches do exist, fortunately. Bonomi and Lutton [1984] described two key ones in another early follow - up to the Kirkpatrick [1983] paper.

C.3.2.1. Neighbourhood Pruning

The first (and more crucial) idea was to prune the neighbourhood structure. Although there are $O(N^2)$ possible 2-Opt moves that can be made from any tour, a 2-Opt move that introduces a long edge into a good tour will typically make things much worse, and hence is unlikely to be accepted. If a way can be found to exclude such moves a priori, it might be able to greatly speed up the algorithm without significant loss in performance. Bonomi and Lutton suggested the following approach, applicable to geometric instances:

- As in the implementation of 2-Opt itself, view: a 2-Opt move as determined by a choice of a city t_1, one of its tour neighbours by t_2, and a third city by t_3 which is to replace t_1 as a tour neighbour of t_2. Now restrict the choice of t_3 as follows:
- Identify the smallest square containing all the cities, and divide it up into m^2 grid cells for some integer m. Then consider for t_3 only those cities that are either in the same cell as t_2 or in a neighboring cell.

Bonomi and Lutton combined neighbourhood pruning with the additional key idea outlined in the next section.

C.3.2.2. Low Temperature Starts

The second idea was to abandon the high temperature portion of the annealing schedule and instead start with a relatively low temperature. To reduce the probability that the process might be immediately trapped in an unproductive region, they (Bonomi and Lutton) proposed using a tour construction heuristic to generate the starting tour (as in traditional local optimisation). This idea of low temperature starts was also proposed by Kirkpatrick [1983], who used it without neighbourhood pruning, but still got sufficient speed-ups to make longer temperature lengths possible and thus improve on the results obtainable by *SA* baseline implementations. Using Nearest Neighbour starting tours on 400 and 900 city random geometric instances, he appears to have obtained average percentage excesses of roughly 2.1% and 2.4% respectively, only slightly worse than those of Lin-Kernighan. Kirkpatrick used the rectilinear as opposed to the Euclidean metric, and average percentage excesses for 2-Opt, and 3-Opt. Typically *LK* are not significantly effected by such a change in the Euclidean metric. He did not compute the Held-Karp lower bounds for these instances, so his average percentage excess to be estimated based on the average Held-Karp lower bound for instances of this type and size, are determined using the techniques of Johnson and Rothberg [1996].

C.3.2.3. Combining the Two

Combining the two ideas of low temperature starts and neighbourhood pruning, as is done by Bonomi and Lutton [1984], should yield even better results. For starting tours, Bonomi and Lutton [1984] used a simple heuristic that generates a tour by stringing together tours for the individual cities into one overall tour, with the tour for each city connected to tours in adjoining cities. They concentrated on random Euclidean instances and used an annealing schedule consisting of 50 temperatures with an initial temperature of $L\sqrt{N}$, where L is the side of the square in which the cities are randomly placed. They used a reduction factor of 0.925. They do not specify their temperature length, but it apparently grew more slowly than their neighbourhood size, which for random geometric instances can be expected to grow as $O(N)$. This means that the ratio of their temperature length to neighbourhood size decreased as N increased. Johnson [1996] suggested that average percentage excess should have increased significantly as N increased, and this indeed seems to have been the case. Although for instances from 200 to 400 cities they claim to have beaten 2-Opt substantially, this is not true for the one large instance they considered, a 10,000-city random geometric. Based on the Held-Karp estimates of Johnson [1996] for such instances, the tour length they report is likely to be roughly 6.9% in excess of the Held-Karp bound. This is far worse than the 5.0% that the neighbour list 2-Opt averaged on tested examples (200 to 400 cities). Just how well their implementation performs for the smaller instances is difficult to judge, as they only report normalised differences between the annealing and 2-Opt tour lengths, and the quality of the latter depends heavily on the details of their (unspecified) implementation of 2-Opt.

C.3.3. 2-OPT

In this section, local improvement algorithms for the *TSP* based on simple tour modifications (exchange heuristics) are considered. Such an algorithm is specified in terms of a class of operations (exchanges or moves) that can be used to convert one tour into another. Given a feasible tour, the algorithm then repeatedly performs operations from the given class, so long as each reduces the length of the current tour, until a tour is reached for which no operation yields an improvement (a locally optimal tour). Alternatively, this can be viewed as a neighbourhood search process, where each tour has an associated neighbourhood of adjacent tours, ie., those that can be reached in a single move, and thus one continually moves to a better neighbour until no better neighbour exists.

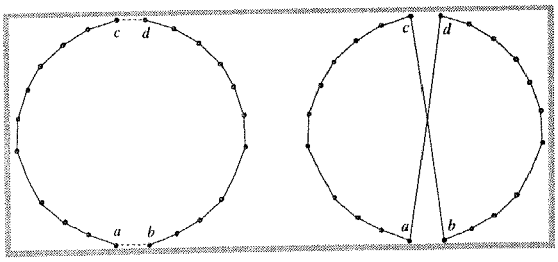


Figure C. 6 A 2 – opt change: original tour on the left and resulting tour on the right

Among simple local search algorithms, the most famous is the 2-Opt. Croes [1958] first proposed the 2-Opt algorithm, although the basic move had already been suggested by Flood [1956]. This move deletes two edges, thus breaking the tour into two paths, and then reconnects these paths in the other possible way. *(Figure C.6)*. Note that this picture is a schematic; if distances were as shown in the figure depicted here, it would be counterproductive and so would not be performed.

C.3.3.1. Bounds on Expected Behaviour

Many of the questions raised in the previous sections have also been addressed from an overall case study point of view, in particular for random Euclidean instances and their generalisations to higher dimensions and other metrics. At present it is not know how to prove tight bounds on the expected performance ratios for sophisticated *TSP* heuristics like the 2-Opt in these models. In the 2-Opt, however, a first step may have been provided by Chandra, Karloff, and Tovey [1994], who have shown that for any fixed dimension d, the expected ratio of the length of the worst 2-optimal tour to the optimal tour length is bounded by a constant. This means that on average 2-Opt can be no worse than some constant times optimal, which is a significant improvement over many worse cases.

Analogous improvements have been obtained with respect to the running time, on the assumption of comparing the unrestricted worst case to the above 2-dimensional average case model. Whereas in the worst case an exponential number of moves may be required before local optimally is reached, the expected number of moves (even starting from the worst possible tour) is polynomial bounded by the two-dimensional model. Under the Euclidean metric the bound is $O(N^{10}logN)$ and under the rectilinear metric it is $O(N^{6}logN)$, as shown by Chandra and Tovey [1994], improving on earlier results of Kern [1989]. Given a method for generating starting tours with expected length $cN^{1/2}$ (presumably achieved from most of the tour construction heuristics), these bounds can be reduced by a factor of $N^{1/2}$.

This paragraph summarises the experimental results obtained using the neighbour list implementations of the 2-Opt by Johnson [1996]. The full picture and results can be found in the reference itself. However it should be noted that these implementations make certain tradeoffs, giving up the guarantee of true 2-Optimally in favour of greatly reduced running times. Neither tour quality, nor the numbers of moves made, appear to be substantially affected by these changes, at least however, for the considered random Euclidean instances. Interestingly, the presented experimental results from Johnson [1996] show that the algorithms perform much better in practice than the theoretical bounds might indicate.

C.3.4. *SA* applied to *TSP*

To get a clearer idea of the advantages of low temperature starts and neighbourhood pruning for the *TSP*, a *TSP* example is now implemented. An obvious drawback to the pruning scheme of Bonomi and Lutton [1984] is that its ability to substantially reduce the overall neighbourhood size depends crucially on the fact that the cities are uniformly distributed. A robust neighbour list 2-Opt is now considered in terms of the notation described. Thus t_3 is simply restricted to the nearest 20 neighbours of t_2. This results in at most *40N* neighbours for a given tour. All *N* cities are candidates for t_1, both tour neighbours of t_1 are candidates for t_2, and t_4 is uniquely determined, given prior choices for t_1, t_2, and t_3. (Note that some of these *40N* neighbours may actually represent the same tour). To improve performance, augmenting the above static pruning rule for t_3 can be combined with an additional dynamic one thus:

- At the beginning of each temperature *T* the neighbour list for each city *c* is restricted as follows:
- Let *c'* be the current tour neighbour of *c* that is farther away. When $t_2 = c$ at this temperature, candidates for t_3 will be restricted to those cities *c''* on the neighbor list for *c* such that a 2-Opt move that increases tour length by *d(c, c'') - d(c, c')* would be accepted with probability at least 0.01, ie. those cities *c''* for which $e^{-(d(c,\,c'')\,-\,d(c,\,c'))/T} \geq 0.01$. If the neighbour lists are sorted by increasing distance, this can be accomplished by simply maintaining a pointer to the last acceptable city on each list, with the pointers updated once per temperature. As to temperature length in this dynamic environment, this is fixed at some constant multiple "a" of the total lengths of the initial neighbour lists. Note that this means that the effective temperature length actually increases as the dynamic pruning starts shrinking the neighbour lists. The increase is

typically from the initial "a" × current total neighbourhood size, to 4 or 5 times that amount.

As to low temperature starts, Kirkpatrick [1983] in this instance uses the Nearest Neighbour starting tour, utilising Bonomi and Lutton [1984] in letting the initial temperature be proportional to $L/\sqrt{N}$ for random Euclidean instances, although the value 1.5 $L/\sqrt{N}$ is chosen rather than the $L/\sqrt{N}$ of Bonomi and Lutton [1984]. This results in an initial acceptance rate of about 50%, and it allows the tour length initially to grow by about a factor of two from its starting value. For other types of instances 50% initial acceptance rate is considered as a criterion determining an appropriate starting temperature by trial and error, although in retrospect it appears that multiplying the length of the Nearest Neighbour tour by *(1.5/N)* typically gives a reasonable value, at least for geometric instances. The temperature reduction factor of 0.95 is retained from the previous baseline implementation.

A new improved *SA* is now used to denote the algorithm obtained from the baseline implementation *SA* by adding neighbourhood pruning and low temperature starts. Note that under this new *SA* the time for an $a = 10$ annealing run is substantially less than that for performing 10,000 runs of 2-Opt, and the average tour quality is better. This illustrates the observed phenomenon that, given enough time, simulated annealing will outperform multiple runs of the corresponding local optimisation algorithm. Note that if the new *SA* is viewed simply as a method for improving a random start 2-Opt, it is very successful. It reduces the average percentage excess from 290% to 99%, almost a factor of 3, whilst the best of 10,000 runs of random start 2-Opt still has an average excess of 240%.

This improved *SA* algorithm, which was written-developed in *MATLAB* and codes, is attached in Appendix D. Presented now is an example of sample instances from the travelling salesman library *(TSPLIB)* in ftp://softlib.es.rice/pub/tsplib on the 100 cities. Here is applied the improved *SA* with the nearest neighbour tour construction with 2 - Opt local search and simulated annealing (Metropolis test) for the 2D Eucledian *TSP*. Initial data is as follows:

```
Initial tour = [1  47    93    28    67    58    61    51    87    25    81    69
          64    40    54    2     44    50    73    68    85    82    95    13
          76    33    37    5     52    78    96    39    30    48    100   41
          71    14    3     43    46    29    34    83    55    7     9     57
          20    12    27    86    35    62    60    77    23    98    91    45
          32    11    15    17    59    74    21    72    10    84    36    99
          38    24    18    79    53    88    16    94    22    70    66    26
          65    4     97    56    80    31    89    42    8     92    75    19
          90    49    6     63] (cities number)
```

```
Cities corresponding to initial tour, X – Y coordinates in km= [1380    939;    2848   96;     3510
          1671;   457     334;    3888    666;    984     965;    2721    1482;   1286   525;    2716
          1432;   738     1325;   1251    1832;   2728    1698;   3815    169;    3683   1533;   1247
          1945    ;       123     862;    1234    1946;   252     1240;   611     673;    2576   1676;
          928     1700;   53      857;    1807    1711;   274     1420;   2574    946;    178    24;
          2678    1825;   1795    962;    3384    1498;   3520    1079;   1256    61;     1424   1728;
          3913    192;    3085    1528;   2573    1969;   463     1670;   3875    598;    298    1513;
          3479    821;    2542    236;    3955    1743;   1323    280;    3447    1830;   2936   337;
          1621    1830;   3373    1646;   1393    1368;   3874    1318;   938     955;    3022   474;
          2482    1183;   3854    923;    376     825;    2519    135;    2945    1622;   953    268;
          2628    1479;   2097    981;    890     1846;   2139    1806;   2421    1007;   2290   1810;
          1115    1052;   2588    302;    327     265;    241     341;    1917    687;    2991   792;
2573   599;    19     674;   3911   1673;   872    1559;   2863   558;    929    1766;
839    620;    3893   102;   2178   1619;   3822   899;    378    1048;   1178   100;
2599   901;    3416   143;   2961   1605;   611    1384;   3113   885;    2597   1830;
2586   1286;   161    906;   1429   134;    742    1025;   1625   1651;   1187   706;
1787   1009;   22     987;   3640   43;     3756   882;    776    392;    1724   1642;
198    1810;   3950   1558]
```

After multiply running of the SA program, the best recorded lowest bound (tour length) was 24474. Presented in *Figure C.7* is the number of searched solutions and in *Figure C.8* the search temperature diagram. In addition, the output run from a *MATLAB* program is as follow:

```
Temperature_of_best_tour_length =  8.8097; Solution_count =  189183; Best_tour_length = 24474 km
Best tour =  1   92   8  42  56  80  31  89  54  40  64   2  44  50  82  95  13  76  33  37   5
52  78  96  30  39  85  68  73  69  81  25   9   7  57  87  51  61  58  67  28  93  47  63   6
49  90  19  75  97   4  65  26  66  70  22  94  16  88  53  79  18  24  38  99  36  84  10
72  21  74  59  17  15  11  32  45  91  98  23  77  60  62  35  86  27  20  12  55  83  34
29  46  43   3  14  71  41  100  48  -1 (cities number)
Search_stop_temperature =   4.9388; Elapsed_time = -2.5918e+006; Solutions_generated =     316533
Floating_point_operations =   3293974
```

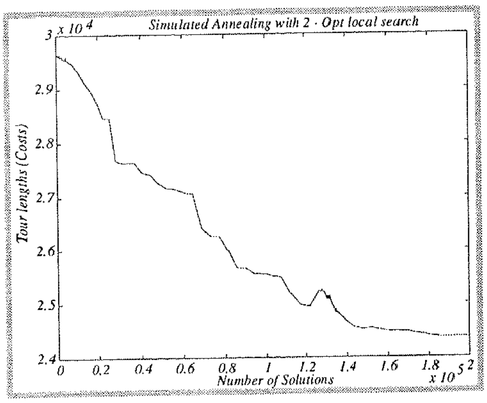


Figure C. 7 - Number of solutions vs. tour length

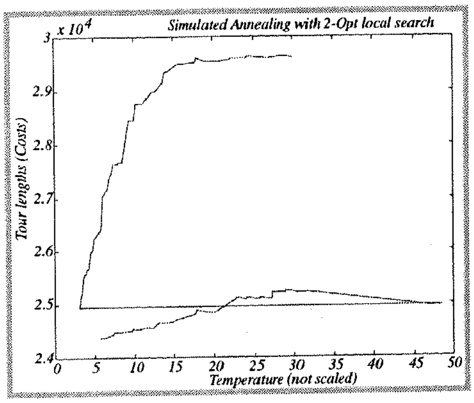


Figure C. 8 - Temperature vs. tour length

C.4. Evolutionary Algorithms in *TSP*

Any abstract task to be accomplished can be thought of as solving a problem, which, in turn, can be perceived as a search through a space of potential solutions. Finding "the best" solution can be viewed as an optimisation process. For small spaces, classical exhaustive methods usually suffice; for larger spaces, special artificial intelligence techniques must be employed. Genetic algorithms *(GAs)* are amongst such techniques; they are stochastic algorithms whose search methods model some natural phenomena. The idea behind genetic algorithms is to do what nature does.

Take rabbits as an example: at any given time there is a population of rabbits, some of them are faster and smarter than other rabbits. These faster, smarter rabbits are less likely to be eaten by foxes, and therefore more of them survive to do what rabbits do best: make more rabbits. Of course, some of the slower, dumber rabbits will survive just because they are lucky. This surviving population of rabbits starts breeding. The breeding results in a good mixture of rabbit genetic material: some slow rabbits breed with fast rabbits, some fast with fast, some smart rabbits with dumb ones and so on. In addition, nature throws in a 'wild hare' every once in a while thus mutating some of the rabbit genetic material. The resulting baby rabbits will (on average) be faster and smarter than those in the original population, because more faster, smarter parents survived the foxes. A genetic algorithm follows a step-by-step procedure that closely matches the story of the rabbits.

Genetic algorithms use a vocabulary borrowed from natural genetics, which talks about *individuals* (or *genotypes, structures*) in a population. Quite often these individuals are called also *strings* or *chromosomes*. This might be a little bit misleading: each cell of every organism of a given species carries a certain number of chromosomes (man, for example, has 46 of them); however discussion here is only about one-chromosome individuals. Chromosomes are made of units – *genes* arranged in a linear succession; every gene controls the inheritance of one or several characters. Genes of certain characters are located at certain places of the chromosome, which are called *loci* (string positions).

Thus, a *GA* can be considered as the implementation of a search strategy on a set of potential solutions (search space). The objective of a *GA* is to find the best solution in the search space while exploring it. Now briefly presented is the commonly used representation by *GA* programmers. Usually representation of the individuals in the population is in the form of Bit Strings (typically about 20 to 100 bits per individual), a typical individual may look like:

1100010110111100110111000011

A population of such individuals is common (typically between 20 and 100), per *generation*. A generation of individuals is then *bred* using genetic techniques (normal ones in which two *parents* are used to form one or two *offspring*, a technique called *crossovers*, and 'nature's throw in', which is called a mutation). There are of course several breeding techniques to proceed from one generation to the next. Of course it is not required for the parents from one generation to randomly breed amongst each other to give the next generation. This occurrence leading to a better next generation would then be pure chance. Instead the program must *select* the best individuals from the present population and derive the next generation using these individuals. In fact the technique is called *selection*, and plays a very important role in *GA*'s. Clearly imitating nature's selective evolution. First, what exactly is needed for representing a Genetic Algorithm? According to Michalewicz [1996] the following five components are necessary for a *GA*:

- a method for representing possible solutions,
- a method for generating the initial population,
- a function to evaluate fitness of the individuals,
- a method to generate a new population from the existing one by applying genetic operators,
- values for various parameters that the *GA* uses such as: population size, probability of applying genetic operators like crossover and mutation, etc.

After initial initialisation of the population, parents are selected based on their relative fitness. In other words, those individuals with higher relative fitness are more likely to be selected as parents. *N* children are created via a recombination from the *N* parents. The *N* children are mutated and survive, replacing the *N* parents in the population. In a *GA* mutation, "flip bits" with some small probability are often considered to be a background operator. Recombination (crossover), on the other hand, is emphasised as the primary search operator. *GA's* are often used as optimisers.

GA's belong to the class of probabilistic algorithms, yet they are very different from random algorithms as they combine the elements of directed and stochastic search. Because of this, *GA*'s are also more robust than existing directed search methods. The following figure outlines a typical *GA*:

```
Procedure GA
$t = 0$
initialize population $P(t)$
evaluate $P(t)$
until (done)
{       $t = t + 1$
            parent_selection $P(t)$
            recombine $P(t)$
            mutate $P(t)$
            evaluate $P(t)$        }
```

Figure C. 9 - Basic procedure for a genetic algorithm

Another important property of such genetic based search methods is that they maintain a population of potential solutions - all other methods process a single point of the search space.

Hillclimbing - methods here use the iterative improvement technique; the technique is applied to a single point (the current point) in the search space. During a single iteration, a new point is selected from the neighbourhood of the current point (this is why this technique is known also as neighbourhood search, or local search). If a new point provides a better value of the objective function, the new point becomes the current point. Otherwise, some other neighbourhood point is selected and tested against the current point. The method terminates if no further improvement is possible. It is clear that the hillclimbing methods provide local optimum values only, and these values depend on the selection of a starting point. Moreover, there is no information available on the relative error (with respect to the global optimum) of the solution found. To increase the chances to succeed, hillclimbing methods usually are executed for a number (large) of different starting points. There are a few versions of the hillclimbing algorithms. One version of a simple (iterated) hillclimbing algorithm is illustrated in *Figure C 10.*

```
procedure iterated_hillclimber;
begin    $t \leftarrow 0;$
         repeat   local $\leftarrow$ FALSE;
                  select individual randomly ($v_c$);
                  evaluate $v_{c;}$
                  repeat;
                       select 30 new strings in the neighborhood ;
                       by flipping bits of $v_c$ ;
                       select string $v_n$ from the set of new strings;
                       with the largest value of function f;
                     if $f(v_c) < f(v_n)$;
                     then $v_c \leftarrow v_{n;}$ ;
                     else local $\leftarrow$ TRUE;
```

```
        until local t ←t + 1;
    until t = MAX; end;
```

Figure C. 10 - A simple version of an iterated hillclimbing algorithm

Initially, all 30 neighbours are considered, and the one v_n which returns the largest value of $f(v_n)$ is selected to compete with the current string v_c. If $f(v_c) < f(v_n)$, then the new string becomes the current string. Otherwise, no local improvement is possible; the algorithm has reached (local or global) an optimum (local = *TRUE*). In such a case, the next iteration $(t \leftarrow t + 1)$ of the algorithm is executed with a new current string selected at random. It is interesting to note that the success and failure of the above algorithms is determined by the starting string.

C.4.1. Genetic Algorithms: Operators

This section now inspects some operators that are used in *GA*, with the default representations of individuals in populations in binary form, and with the *crossover* as one of the most basic operator.

Crossover - As mentioned earlier, a *GA* performs a multi-directional search by maintaining a population of potential solutions and encourages information formation and exchange. Every new population in a new generation consists of individuals formed from the previous generation by some genetic operation.

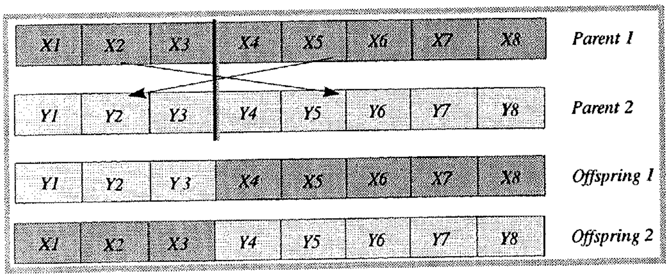


Figure C. 11 - Simple one point crossover

Crossover combines the features of two parent chromosomes to form two similar offspring by swapping corresponding segments of the parents. For example, if the parents are represented by chromosomes *(a, b, c, d, e)* and *(p, q, r, s, t)*, then crossing the chromosomes after the second gene would produce the offspring *(a, b, r, s, t)* and *(p, q, c, d, e)*. The basic idea behind the crossover is to exchange information of both parents to produce two offspring. A point is chosen at random and that point is where the 'cut' is made in the two participating parents. Then the information "beyond" these points is exchanged with the information from the other parent. This process is called the Single Point Crossover. This is the most basic kind of crossover. The idea is simple, having a preset probability of crossover (p_c), generates a random number between 0 and 1, for each individual under consideration, and if it is less than the probability of crossover, then that individual is selected to be one of the *parents*. Then two parents are selected to gave two *offspring*. This process is demonstrated in *Figure C.11*.

Similarly, it's possible to have two cutting points or more, plus the facility to add various flavours to the two-parent crossover. However there may still exist different variations of the simple crossover, which involve more than two parents. In such a case two cutting points in the parents can exchange two parts between three parents, in any of one of eight possible ways, to get three offspring.

Mutation - As the name suggests, this operator involves *random* change of bits, and just like the rare occurrence of mutation in the real world of evolution, the probability of mutation in the little world of *GA* is also much less (typically $p_m < 0.1$). Mutations in the real sense means that one or a few of genes in the chromosome are altered naturally, on their own, within one generation to the next. Therefore, mutations may throw in various interesting results. The results of mutations usually produce large *steps* in the evolution process.

Similarly, a mutation is also the change of a gene in one chromosomed individual. Of course, here it means a bit flip. So a typical mutation is shown below.

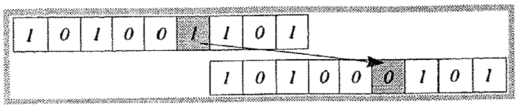


Figure C. 12 - Mutation (Bit Flip)

Now the mutation usually flips a couple or more of bits in the individual. If the individual has floating point numbers rather than bits, then it simply changes one of the floating point numbers randomly. Then the *bit flip mutation,* which is shown in *Figure C.12,* occurs only when one bit is flipped. A random number between 0 and 1 is generated. If the number is less than p_m then a bit is selected at random and flipped.

Inversion - In this operation an individual is selected, and two points are randomly selected in this individual. Now the individual is cut at these two points and the 'string' of genes between them is reversed, ie. *(a, b, |c, d, e, f|, g, h, i)* will result as *(a, b, f, e, d, c, g, h, i),* but at the same time the positions of *c...f* are remembered for further use. So for example, *c* will be remembered as element 3 in the chromosome, though when fitness is derived, the chromosome will be treated as it appears after inversion.

Selection - on numerous occasions the word *select* has been mentioned in the context of picking individuals from the whole population. To decide which individuals will be the lucky ones to continue the evolution process, a commonly used selection process called the *roulette wheel selection method* is utilised. The construction of the roulette wheel is as follows:

- First of all calculate the fitness value *(*x_i*)*, for each and every individual x_i (*i = 1, ..., population size*).
- Next find the total fitness of the whole population.
- Thus, total Fitness = Summation (evaluation (x_0), ..., evaluation $(x_{population\_size})$).
- Next generate the probability of selection p_{si} for each individual in the population. p_{si} = evaluate(x_i)/Total Fitness.
- Calculate a cumulative probability q_i for each individual x_i.

  q_i = Summation(p_{s0}, ..., $p_{spopulation\ size}$).

This selection is based on spinning the roulette wheel the number of times equal to the population size, to generate a random number between 0 and 1, and then to compare it with the respective cumulative probability q for the individual under consideration. If the random number falls in the interval $q_{i-1} < r <= q_i$, or if it is less than the cumulative probability of the first individual when considering it, then select the individual. It is very clear from the selection process that some healthy individuals would be selected more than once, and then weaker individuals have a very small chance of getting selected.

The above description is the basic selection process in the *GA*, and a couple of variations of the basic selection process exist. For example, to have a selection method in which weights are associated with individuals in the population. In this case the individuals are then selected proportional to their rank, rather than the actual evaluation value. This method is called *ranking selection.*

One more selection method is called *tournament selection.* Here, some number *i* of individuals is selected and the best one from this set is sent into the next generation. This process is repeated *population size* number of times; typically a tournament has a size (*i*) of 2. Larger tournament sizes increase the selective pressure. If the whole population size (p_s) is *i*, then after every processing time it is very simple to inspect the whole population for the best individual. This is repeated p_s times. Similarly, as *i* goes on decreasing, with fewer and fewer individuals available for consideration, the selection space becomes smaller and smaller.

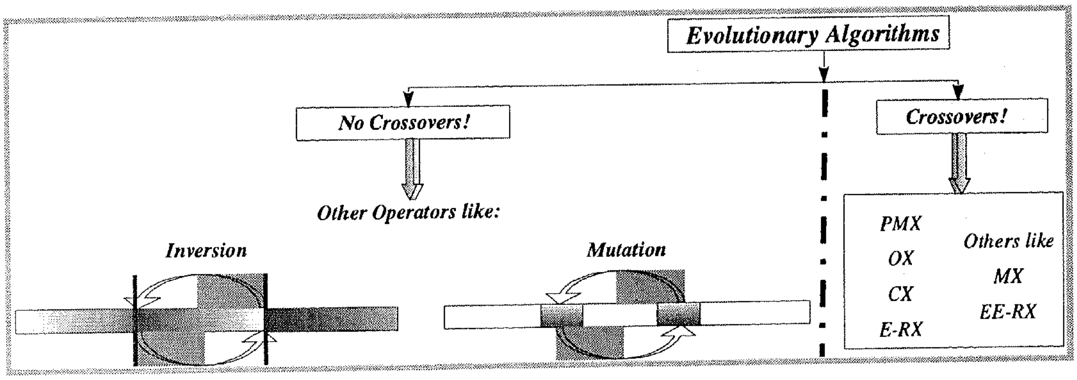


Figure C. 13 – Evolutionary approaches to solve the TSP

C.4.2. Genetic Algorithms: Application for solving the *TSP*

During the past few years, the *TSP* has become a target for the genetic algorithms community. The use of genetic algorithms as an approach to optimisation can be traced back at least to the 1970's. See Goldberg [1989] for some of the history. The best adaptations of this approach to the *TSP* follow the basic schema presented in *Figure C.14,* where each performance of the loop consisting of Steps 3.1 through to 3.5 can be viewed as the processing of a single generation in the evolutionary process. Note that the operations on different solutions can be performed in parallel if desired, and so this is sometimes called the parallel genetic algorithm. As with the schema for simulated annealing in Section C.3, this *TSP* schema leaves several operations and definitions unspecified. A specific adaptation of the schema to the *TSP* needs to specify: *k* and *k'*, the methods for generating starting solutions (tours), the local optimisation algorithm A, the mating strategy, the nature of the crossover and mutation operators, the selection strategy, and the criterion for convergence.

1. Generate a population of k starting solutions $S = \{S_1, \ldots, S_k\}$.

2. Apply a given local optimisation algorithm A to each solution S' in S, letting the resulting local optimal solution

```
replace S' in S.
3. While not yet converged do the following:
        3.1. Select k' distinct subsets of S of size 1 or 2 as parents (the mating strategy).
        3.2. For each 1-element subset, perform a randomised mutation operation to obtain a new solution.
        3.3. For each 2-element subset, perform a (possibly randomised) crossover operation to obtain a new
        solution that reflects aspects of both parents.
        3.4. Apply local optimisation algorithm A to each of the k' solutions produced in Step 3.3, and let S' be the
        set of resulting solutions.
        3.5. Using a selection strategy, choose k survivors from S ∈ S', and replace the contents of S by these
        survivors.
4. Return the best solution in S.
```

Figure C. 14 - General schema for a genetic optimisation algorithm.

It should be noted that the schema in *Figure C.14* is not what was meant by a "genetic" algorithm in such early references as Holland [1975]. In particular, the application of local optimisation to the individual solutions in Steps 2 and 3.4 could be viewed as an almost heretical addition. In the context of the original biological motivation for the genetic approach, it embodies the discredited Lamarckian principle that learned traits can be inherited. Nevertheless, such local optimisation steps appear to be essential if competitive results are to be obtained for the *TSP*, and attention is restricted to what follows to *GA's* that use them.

Even without the local optimisation steps of *Figure C. 14*, a genetic algorithm can properly be classified as a variant of a local search. Implicit is a neighbourhood structure in which the neighbours of a solution are those solutions that can be reached by a single mutation or mating operation. With the local optimisation steps, the schema can also be viewed simply as a variant on an algorithmic approach discussed extensively in the previous section, ie. the best-of-*k*-runs approach to local optimisation. Here, instead of independent random starts, the genetically motivated operations to construct is used with hope for better starting tours, ones that incorporate knowledge we have obtained from previous runs. It will be seen that this latter way of viewing the schema is the more productive.

As presented above, these *GA's* also work under the same principle, ie. search for a near optimal solution from a huge search space by employing populations of potential solutions. This is done similar to what was described in the previous section, ie. trying and using the same operators (crossover, mutation and inversion), selecting individuals, and trying for a good solution. The *TSP* has a simple evaluation function, all that is needed is to select a valid tour, and to calculate the length of that tour. Also required is to store the distances between cities in a file, or better to store the co-ordinates of the cities in arrays. Thus, when needed, one simply calculates the distance between any two cities by using the *distance formula*. It is also required to maintain a population of tours (say in another array) so that comparing tours would not also pose a problem. However, care is needed before deciding how to represent the tours, as this could mean substantial computational time. Thus it is better, first of all, to consider another small problem, one requiring only slight alteration to the basic operators for the *GA's*.

Thus, it is known that *GA's* are usually represented by individuals having binary representation, and that the operators are designed to work essentially with binary operators. Thus the *TSP* cannot be represented in binary form without going an "extra mile" to take care of the glitches that would result. Suppose, for a five city problem (hardly a problem, but just for analogy) the five cities were represented as 00001, 00010, 00011, 00100 and 00101. Consider the tour of 00001000|10000110010000101 (the pipe indicating the cutting point in a simple crossover in the tour 1, 2, 3, 4, 5), and another tour 00101001|00000110001000001 (5, 4, 3, 2, 1). After the crossover, new tours are (1 0 3 2 1) and (5 6 3 4 5). It is clear that cities 0 and 6 have appeared out of nowhere, the first tour is invalid as cities 4 or 5 is not seen, and city 1 is repeated. In the second tour, city 5 is repeated, and cities 1 and 2 are missing. The conclusion of this little experiment is not that it is impossible to use binary representation for the *TSP*, but that they must be taken in order of the dependencies, taking care of omissions, duplications, and some other problems. All of this adds to the computation time, and in the end will probably be unproductive.

Adjacency Representation - consists of putting down the tour in the form of a list of cities. A city *x* is listed in position *y* if the tour leads from city *x* to city *y*. This could be a bit confusing at first, but consider a straightforward example:

(3 8 4 9 2 1 6 7 5) represents tour 1 – 3 – 4 – 9 – 5 – 2 – 8 – 7 – 6.

Simple observation concludes that each tour has only one adjacency list, whilst constructing adjacency lists requires care, as some of them might represent illegal tours. Thus consider the following:

(3 8 4 9 1 2 6 7 5) represents tour 1 – 3 – 4 – 9 – 5 – 1.

A simple exchange of the 5th and 6th elements makes the above tour illegal. This also indicates that the classical crossover may not be a very good operator. Thus, a repair algorithm is required to use the classical crossover. The advantage of this representation is that it allows schemata analysis (allows specification of natural building blocks), edges in this case, and denotes all tours with edges (5 4) and (8 3). A disadvantage is the poor results the operators give for this representation.

Ordinary Representation - represents a tour of *n* cities such that the *x*-th city in the list is a number in the range from 1 to *n* – *x* + *1* (for some ordered list of cities *C* only serves as a reference point for tours), for example:

C = (1 2 3 4 5 6 7 8 9), then a tour T = 1 – 4 – 6 – 9 – 3 – 7 – 2 – 8 – 5 is represented as
R = (1 3 4 6 2 3 1 2 1), say another tour S = 2 – 4 – 6 – 8 – 1 – 3 – 5 – 7 – 9
is represented as P = (2 3 4 5 1 1 1 1 1).

The best part of this representation is that the classical crossover actually works. To see this concept in action place the cutting point say, after the 5th element in the list. So the offspring *A* and *B* after the crossover would be:

A = (1 3 4 6 2 1 1 1 1) which gives the tour TA = 1 – 4 – 6 – 9 – 3 – 2 – 5 – 7 – 8 and
B = (2 3 4 5 1 3 1 2 1) which gives the tour TB = 2 – 4 – 6 – 8 – 1 – 7 – 3 – 9 – 5.

Both of these tours are legal tours. However the experimental results associated with this representation and the classical crossover are not very encouraging, and experts deem it not a very appropriate representation method for the *TSP*.

Path Representation - is the simplest of all the representations seen so far. It is a natural list of how the tour reads, for example:

Tour T = 1 – 2 – 5 – 7 – 3 – 4 – 6 – 9 – 8 can be represented simply as R = (1 2 5 7 3 4 6 9 8).

The next section presents many powerful operators for the path representation.

C.4.2.1. Operators for the TSP

First consider the operators for the Adjacency Representation. There are three crossovers that will be presented here: *alternating edges, sub-tour chunks and heuristic crossovers.*

Alternating-Edges Crossover - builds an offspring by randomly choosing an edge from the first parent. Then selects an appropriate edge from the second parent. The operator extends the tour by selecting edges from alternating parents. If now a newly added edge were to introduce a partial tour for the current child, then the operator instead selects a random edge until the situation is corrected. An example of how a child might be created from two parents is thus: *P1 = (2 3 8 7 9 1 4 5 6) and P2 = (7 5 1 6 9 2 8 4 3),* then the offspring may be *O1 = (2 5 8 7 9 1 6 4 3).* The process started from the edge (1 2) from the first parent, with a note that a random edge was introduced during this process. The edge (7 6) was introduced instead of (7 8), to maintain a legal tour.

Sub-Tour Chunks Crossover - The name of this crossover suggests that the tour is built from small chunks of sub-tours, and this is the way it works. It randomly selects a subtour from one parent, then randomly selects another (random length) subtour from the other parent, and so on. If an illegal tour is to be introduced during this process, it randomly keeps selecting other edges from the remaining, until the situation is corrected.

Heuristic Crossover - This crossover builds the offspring by choosing a random city as the starting point for the resulting tour. It then compares the edges in the two parents leaving this city, and selects the shorter edge. In case of a conflict, a random edge is selected to correct the situation. It continues doing this until a valid tour is reached.

However, as already stated, the performance of these crossovers and this representation is hardly encouraging. According to Michalewicz [1995], in three experiments on 50, 100 and 200 cities, the system found tours within 25%, 16%, and 27% of the optimum, in approximately 15000, 20000, and 25000 generations respectively.

Next is presented the crossovers defined for the *path* representation. The crossovers presented briefly here are *partially mapped (PMX), order (OX), and cycle (CX).*

Partially-Mapped Crossover - Goldberg proposed this. Here the offspring is obtained by choosing a part from the tour. This part of the tour is the one between two randomly selected cutting points. These cutting points act as boundaries for swapping operations. Thus:

$$P1 = (1\ 2\ 3 \mid 4\ 7\ 8\ 9 \mid 5\ 6) \text{ and}$$
$$P2 = (4\ 7\ 5 \mid 1\ 9\ 3\ 2 \mid 8\ 6).$$

The above two parents are now considered. First the segments between the two cut points are swapped, then the rest of the tour marked with *X* can be read as unknown. Thus:

$$O1 = (X\ X\ X \mid 1\ 9\ 3\ 2 \mid X\ X)$$
$$O2 = (X\ X\ X \mid 4\ 7\ 8\ 9 \mid X\ X).$$

This also defines the mapping 1 to 4, 9 to 7, 3 to 8, and 2 to 9, and further cities are incorporated from the original parents for which there is no conflict, obtaining the following:

$$O1 = (X\ X\ X \mid 1\ 9\ 3\ 2 \mid 5\ 6)$$
$$O2 = (X\ X\ 5 \mid 4\ 7\ 8\ 9 \mid X\ 6).$$

The first step is to decide what will fit in, where the *Xs'* are in the offspring. It is very simple to take care of these remaining empty spots by employing mapping and to obtain the following offspring as a result:

$$O1 = (4\ 9\ 8 \mid 1\ 9\ 3\ 2 \mid 5\ 6)$$
$$O2 = (1\ 9\ 5 \mid 4\ 7\ 8\ 9 \mid 3\ 6).$$

Order Crossover - Here the offspring is built by choosing a sub-sequence of a tour from one parent and preserving the relative order of the cities from the other parent. Thus consider this systematically through an example:

$$P1 = (1\ 2\ 3 \mid 4\ 7\ 8\ 9 \mid 5\ 6)$$
$$P2 = (4\ 7\ 5 \mid 1\ 9\ 3\ 2 \mid 8\ 6).$$

Now consider how the offspring are produced. First, the segments between the cut points are copied into the offspring, thus:

$$O1 = (X\ X\ X \mid 4\ 7\ 8\ 9 \mid X\ X)$$
$$O2 = (X\ X\ X \mid 1\ 9\ 3\ 2 \mid X\ X).$$

Next for *O1*, start reading the cities in-order from P2, after the second cut, obtaining:

$$8 - 6 - 4 - 7 - 5 - 1 - 9 - 3 - 2\ .$$

Now ignoring the cities which are already present in *O1*, which reduces the above to:

$$6 - 5 - 1 - 3 - 2.$$

Now start inserting these cities in *O1* after the second cut point, and repeat the same procedure for the second offspring, thus getting:

$$O1 = (1\ 3\ 2 \mid 4\ 7\ 8\ 9 \mid 6\ 5) \text{ and} \qquad O2 = (4\ 7\ 8 \mid 1\ 9\ 3\ 2 \mid 5\ 6).$$

The order crossover exploits the property of the path representation, which states that the order of cities and not their positions are important, ie., the two identical tours are:

$$8 - 6 - 4 - 7 - 5 - 1 - 9 - 3 - 2 \text{ and} \qquad 4 - 7 - 5 - 1 - 9 - 3 - 2 - 8 - 6\ .$$

Cycle Crossover - builds the offspring in such a way that each city and its position comes from one of the parents. Thus, the *CX* works as follows:

$$P1 = (1\ 2\ 3 \mid 4\ 7\ 8\ 9 \mid 5\ 6) \quad \text{and} \quad P2 = (4\ 7\ 5 \mid 9\ 1\ 3\ 2 \mid 8\ 6),$$

and taking the first city from the first parent to form the first offspring thus:

$$O1 = (1\ X\ X\ X\ X\ X\ X\ X\ X\ ).$$

Since having to select cities from the two parents and from the same position, there is little choice but to first see what is in the first position of *P2* and fill that position up, thus:

$$O1 = (1\ X\ X\ 4\ X\ X\ X\ X\ X).$$

Similarly, 4 now means 9, which gives:

$$O1 = (1\ X\ X\ 4\ X\ X\ 9\ X\ X).$$

Continuing the similarly gives:

$$O1 = (1\ 2\ X\ 4\ 7\ X\ 8\ 9\ X\ X).$$

For the remaining cities just consider the other parent and fill it up as before, getting:

$$O1 = (1\ 2\ 5\ 4\ 7\ 3\ 9\ 8\ 6).$$

The second offspring is obtained by repeating this. Thus *CX* preserves the absolute position of the elements in the parent sequence. Most of the operators discussed so far take into account cities, ie. their positions and order, as opposed to edges (links between cities).

Edge-Recombination (ER) crossover - a relatively new crossover method. The general idea behind the *ER* crossover is that the objective function to be minimised is the total number of edges, which constitute a legal tour, and that the offspring contains edges present in both parents. This is done with the help of an edge list created from the parent tours. The two parents are first listed as:

$$P1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)\ and\quad P2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5).$$

Then an edge list is created, such that each city with all its edges is represented as:

City 1: 9 2 4. *City 2: 1 3 8.* *City 3: 2 4 9 5.* *City 4: 3 5 1.* *City 5: 4 6 3.*
City 6: 5 7 9. *City 7: 6 8.* *City 8: 7 9 2.* *and* *City 9: 8 1 6 3.*

During the construction of offspring, the initial city, the first city in the parents with the smallest number of edges, is selected. With the numbers being equal, a random selection is made and assume city 1 is selected. This city is directly connected to three other cities: 9, 2, and 4. The next choice is to be made from these three, thus 4 and 2 have three edges each, and 9 has four. A random choice between 2 and 4, say 4, is now made. Thus, city 4 now has 3, 5, and 1 in its list. City 1 is already in the tour and city 5 has lesser edges than 3, so city 5 is selected and the procedure continuous until the offspring *O1 = (1 4 5 6 7 8 2 3 9)* is obtained. This offspring is entirely composed of edges from both its parents.

Next is presented an example of sample instances from the travelling salesman library *(TSPLIB)* in ftp://softlib.es.rice/pub/tsplib of 30 cities, on which is applied the *GA* using an order-based representation. The program was written-developed in *MATLAB 5* and codes are attached in Appendix D (with all codes for *GA* presentation). The initial data (presented in graphical form in *Figure C.15)* is as follows:

Cities corresponding to initial tour, X – Y coordinates (km) = [82, 7; 91, 38; 83, 46; 71, 44; 64, 60; 68, 58; 83, 69; 87, 76; 74, 78; 71, 71; 58, 69; 54, 62; 54, 67; 37, 84; 41, 94; 2, 99; 7, 64; 22, 60; 25, 62;

Conclusion

This appendix has surveyed a wide variety of approaches to the *TSP*. The best ones are all based on local search in one form or another. Assuming one has enough time to run something more sophisticated than a simple tour construction heuristic, the first choice would probably be an efficient implementation of one of the classic local optimisation algorithms, such as the 2-Opt, 3-Opt, and Lin-Kernighan. The last algorithm comes within 1.5% of optimal for random Euclidean instances with as many as a million cities, and it is capable of doing almost as well for the real-world instances in *TSPLIB*. Within the running time bounds of these algorithms, no tabu search, simulated annealing, genetic, or neural net algorithm has yet been developed that provides comparably good tours. If shorter tours are wanted and significantly more time is available, both simulated annealing and genetic algorithms can, for many instances, find better tours than could be found in the same time by performing multiple independent runs of the Lin-Kernighan algorithm. These *SA* and *GA* approaches also have the advantage that one can use them without the detailed coding required to implement the Lin-Kernighan.

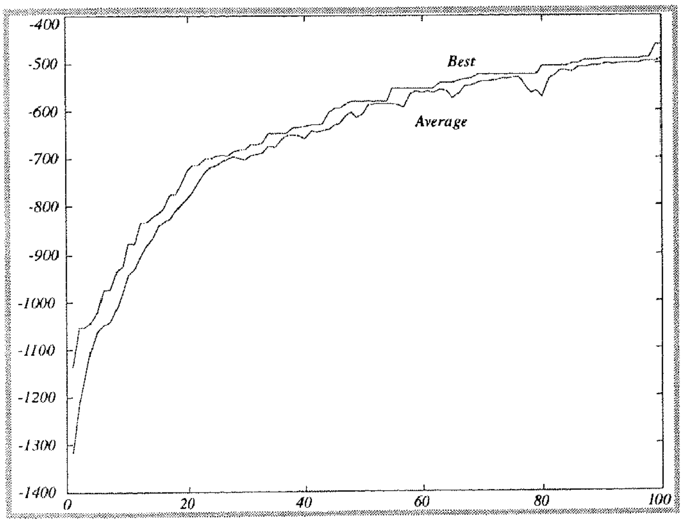


Figure C. 19 - The best and average function distribution across the GA generation

C.5. Background of the Simulated Annealing Algorithm

Simulated annealing, or *SA* for abbreviation, is a combinatorial optimisation method, where the aim is to find the best solution among a finite number of possible solutions. Simulated annealing is good at finding near optimal solutions with a reasonable amount of computing. However, it must be noted there is still uncertainty if the best solution found is the global optimum. This restricts the use of the algorithm to cases where a good local optimum is also acceptable.

C.5.1. Introduction

Simulated annealing is a technique for finding an optimal or near optimal solution for combinatorial optimisation problems, or problems which have discrete variables. It was proposed by Kirkpatrick 1983 and has been successfully applied to cell formation, *VLSI*, circuit partitioning, placement, and routing in the physical design of integrated circuits.

The goal of a combinatorial optimisation algorithm is to find the state of lowest cost (or energy) from a discrete space of admissible configurations *S*. For each problem, a cost function must be defined which maps each state to a real number denoting its cost. For many problems, the number of possible states grows exponentially with the size of the input. Optimising becomes the process of searching for the state of lowest cost in a hyper dimensional space. With a large number of possible states to visit the "brute force" method of visiting all configurations becomes impractical. Clearly, a search strategy is required to uncover the lowest cost solution in the jungle of states.

For many problems the states of the configuration space are related. A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to the subproblems. Either a "greedy" or a "dynamic" programming algorithm may solve these cases. In a greedy choice problem a globally optimal solution can be found by making a locally optimal (greedy) decision. The best choice is made at each moment, thus at each step the ramifications of the previous choice is solved. The choice made by a greedy algorithm can not depend on future decisions or solutions to subproblems. In dynamic programming a choice is made at each step that may depend on the solutions to the problems. Solving an optimal substructure problem will require a greedy or dynamic strategy depending on the nature of the problem.

C.5.2. Local Optimisation

The basic concepts of simulated annealing comes from local optimisation; also called an iterative improvement. A combinatorial optimisation problem can be characterised by a set of solutions and a cost function, which attaches a cost to every solution. Here the goal is to find the solution with minimum cost. (A maximisation problem can be turned into a minimisation problem easily by reversing the sign of the cost function).

In local optimisation one must first define a procedure to perturbate the solutions in order to get new ones. Perturbation is accomplished by small changes to the current solution. Solution *S'* is a neighbour of another solution *S*, if *S'* can be obtained from *S* by one such change. The neighbourhood of a given solution *S'* is the set of all neighbours of *S*.

The local optimisation algorithm takes an initial solution, which can be random or based on some heuristic procedure, and then starts to investigate its neighbourhood. If it finds a solution in the neighbourhood whose cost is lower than the present cost, it moves to that solution. This procedure is iterated until no further improvements can be made. The solution this algorithm finds is necessarily a local optimum, but it is usually not the global optimum. It is possible, of course, to apply the method many times with different initial solutions and hope that the best solution found is the global optimum.

C.5.3. Simulated Annealing

The term simulated annealing comes from a physical point of view. Annealing is a physical process where a crystal is cooled down from the liquid phase to the solid phase in a heat bath. If the cooling is done carefully enough (instead of rapid quenching), the energy state of the solid at the end of the cooling is at its minimum (or very near to it).

At the heart of the simulated annealing algorithm is the Metropolis Monte Carlo [Kirkpatrick 1983] procedure, which was introduced to provide an efficient simulation of a collection of atoms in equilibrium at a given temperature. The Metropolis procedure is the inner loop of the simulated annealing algorithm as shown in *Figure C.20.* Whilst the "greedy" algorithm forbids changes of state that increase the cost function, the Metropolis procedure allows moves to states that increases the cost function. Kirkpatrick suggested that the Metropolis Monte Carlo method could be used to simulate the physical annealing process and to solve combinatorial optimisation problems. They suggested adding an outer loop, which lowers the temperature in slow stages from a high melting temperature, until the system freezes, and no further changes occur. At each temperature the simulation must proceed long enough for the system to reach a steady state. The sequence of temperatures, and the method to reach equilibrium at each temperature, is known as an annealing schedule. They showed that this same technique could be applied to combinatorial optimisation problems if a cost function is used in place of the energy, and the temperature is used as a control parameter.

Simulation of the physical cooling can be done with the Metropolis algorithm [Kirkpatrick 1983] first proposed in the early days of computing. It generates sequences of states of the solid in the following way: given a current state i of the solid with energy E_i, then the next state j is generated by applying a small perturbation to the solid, for instance moving a particle. The energy of the next state is E_j. If the energy difference E_i - E_j is less than or equal to zero, the state j is accepted as the new current state. Otherwise the state j is accepted with a probability given by:

$$exp\left(-\frac{E_j - E_i}{k_b T}\right) \qquad \textit{Equation C. 14}$$

where:

T – is temperature of the solid, and

K_b – denotes the Boltzmann constant.

If the lowering of the temperature is done slowly enough, the solid reaches thermal equilibrium at each temperature. In the Metropolis algorithm this is achieved by applying a sufficiently large number of perturbations at each temperature.

Physical system	***Optimisation problem***
State	Feasible solution
Energy (Internal)	Cost (Objective function)
Atomic position	Decision Variables
Ground state	Optimal solution
Rapid quenching	Local search
Careful annealing	Simulated annealing
Cool into a stable , low energy state	Find a near optimal configuration

Table C. 4 - The analogy between the physical system and the optimisation problem.

The *SA* algorithm [Kirkpatrick 1983] can be viewed as a process analogous to the simulation of cooling down a solid by the Metropolis algorithm. The analogy between *SA* (combinatorial optimisation) and the physical situation (cooling down a solid) is presented in *Table C.4*. Using this analogy, the simulated annealing algorithm in *Figure C.20* can be produced. Thus:

```
1. Estimate an initial solution S and an initial temperature $T$.
2. If stop criterion not satisfied do the following:
        (a) While inner loop criterion not satisfied do the following:
                i. Select a neighbour $S'$ of S.
                ii. Let $\Delta = cost(S') - cost(S)$
                iii. $\leq 0$ (downhill move), set $S = S' = T$.
                iv. 0 (uphill move), set $S = S'$ with probability $e^{-\Delta/T}$
        (b) Reduce temperature $T$.
3. Return to S.
```

Figure C. 20 - Simulated annealing.

SA can be viewed as an extension to local optimisation. Local optimisation finds new solutions by making small transformations to the current solution in such a way that every new solution is better than, or equal to, the previous solution. In simulated annealing, even a move that makes a current solution worse is accepted with probability as follows:

$$P_{accept} = exp\left(\frac{-\Delta}{T}\right) = exp\left(-\frac{cost(S') - cost(S)}{T}\right) \qquad \text{Equation C. 15}$$

where:

cost(S) - denotes the cost function assigned to the current solution,

cost(S') - denotes the cost function assigned to the proposed next solution,

Δ - is the cost change, and

T - the "temperature" *T* is simply a control parameter in the same units as the cost function.

Ideally, when the local optimisation is trapped at a poor local optimum, simulated annealing can "climb" out of the poor local optimum.

The criterion for stopping the iteration (detecting when the process is "frozen") in step **2** in *Figure C.20* varies depending on the literature sources. One possible solution sourced is to give a required percentage of accepted moves. When the actual percentage of accepted moves is below this required level, for at least a given number of successive executions of steps ***2a*** and ***2b***, and no improvement of the best solution found so far is recorded, the process is considered frozen. Another possible solution is to stop when the cost function at the end of step ***2a*** is the same for a given number of consecutive temperatures.

In step ***2(a)i***, *S'* is usually selected randomly amongst all the neighbours of the current solution, with the same probability for all neighbours. However, with a complicated neighbourhood structure, a non-uniformly random selection might be appropriate.

A common inner loop criterion (step *2a*) is to perform the loop a constant *(L)* number of times. The value of *L* should be large enough so that the equilibrium is achieved before reducing the temperature. A rule of thumb is to take L equal to, or proportional to, the size of the neighbourhood [Aarts 1989]. The new temperature in step *2b* is usually calculated according to :

$$T' = r \times T \qquad \text{Equation C. 16}$$

where:

T' – is the new temperature, and

r – is typically in the range *0.95 – 0.99.*

Some other cooling schedules have also been studied (see for example [Johnson 1989]), but this exponential schedule *(Equation C.15)* seems to be not only one of the most efficient, but also very robust. It can be shown that simulated annealing converges asymptotically to the set of globally optimal solutions if some conditions are fulfilled [Aarts 1989, Chapter 2-3].

Unfortunately, pushing the probability of finding a global optimum provably to a given level, requires very slow cooling schedules, and long iterations at each temperature. Hence, it can take more time than what it would take to accomplish an exhaustive manual search in the solution space. Furthermore, it is still not known whether or not an actual global optima has been reached. Nevertheless, in practice, *SA* often finds very good local optima with a reasonable amount of computing. The main problems when implementing the simulated annealing algorithm are:

Selection of the cost function. Cost functions usually employ some heuristic related to the problem at hand. If there are difficulties whilst applying the algorithm, it is difficult to say whether the problem is too hard or whether the cost function is poor.

Efficiency. Perturbations and the corresponding cost change calculations should be simple enough to perform, so that the algorithm can perform iterations very fast.

It has been shown that the simulated annealing algorithm, when started in an arbitrary state and given an appropriate annealing schedule, will eventually converge to a global optimum. Although these results required an infinite amount of computation time to guarantee convergence, in practice, simulated annealing has been extremely successful when applied to circuit partitioning, placement problems, cell formation, chip placement and microprocessor layout. A substantial body of literature has focused on the *NP* – complete problems, which addresses the placement of chips on a microprocessor circuit board in order to optimise certain design parameters, such as wire density in channels, total wiring length, etc. (Kirkaptrick [1983], Darema [1987], Vidal [1993] and Casotto [1997]); it has outperformed all other known algorithms.

APPENDIX D

D.1. Maximum Undirected Spanning Tree

```
% Finds the maximum undirected spanning tree of an undirected graph. % function [Z_tree, cost] = maxtree(-C, Zin,
Zout). % input: % C Cost (distant) matrix where C = C' and C(i,i) = NaN. % Zin  Arcs forced in. Arc list. % Zout
Arcs forced out. Arc list. (could also be given as NaN in C). % output: % Z_tree The maximum undirected spanning
tree. Arc list. % cost Total cost.
function [Z_tree, cost] = maxtree(C1, Zin, Zout)
C=-C1;
if nargin < 3
  Zout = [];
  if nargin < 2
    Zin = [];
  End;  end;
for i = 1:size(Zout,1); % Exclude the arcs in Zout from C ;
  C(Zout(i,1),Zout(i,2)) = nan;
  C(Zout(i,2),Zout(i,1)) = nan;
end
m = size(C,1);  % Number of nodes
Z_tree = [];    % The maximum undirected spanning tree
cost = 0;       % The maximum undirected spanning tree cost
setlist = 1:m;  % Let all nodes be members of different sets
for i = 1:size(Zin,1); % Include the arcs in Zin in Z_tree;
  Z_tree = [Z_tree;Zin(i,:)];
 cost = cost+C(Zin(i,1),Zin(i,2)); % Update setlist. %newset= min(setlist(Zin(i,1)),setlist(Zin(i,2))); %index =;
find(max(setlist(Zin(i,1)),setlist(Zin(i,2)))==setlist);
  newset = setlist(Zin(i,1));
  index = find(setlist(Zin(i,2))==setlist);
  setlist(index) = newset*ones(1,length(index));
C(Zin(i,1),Zin(i,2)) = nan; % Exclude the arc from C ;
  C(Zin(i,2),Zin(i,1)) = nan;
end
while size(Z_tree,1) < m-1
[row,col]=find(C==min(C(finite(C)))); % Find the arc with minimum cost;
if setlist(row(1))~=setlist(col(1)) % Add arc k if it does not give a cycle (nodes belong to different sets);
    Z_tree = [Z_tree;[row(1) col(1)]];
    cost = cost+C(row(1),col(1)); % Update setlist.  %newset = min(setlist(row(1)),setlist(col(1)));
    newset = setlist(row(1)); %index = find(max(setlist(row(1)),setlist(col(1)))==setlist);
    index = find(setlist(col(1))==setlist);
    setlist(index) = newset*ones(1,length(index));
  end
C(row(1),col(1)) = nan; % Exclude the arc from C;
  C(col(1),row(1)) = nan;
End; cost=-cost;
```

D1.1. Output of Maximum Undirected Spanning Tree

```
» c=[0 0 0 8 0 0 0 0 0 0 0 0;... 0 0 0 0 1 0 0 0 0 0 0 0;...0 0 0 0 2 0 0 0 0 0 0 0;...0 0 0 0 0 4 14 11 0 0 0 0;...0 0 0 0 0
1 0 0 0 0 0 0;...0 0 0 0 0 0 0 0 0 0 0 0 ;...0 0 0 0 0 0 0 0 0 9 8 0;...0 0 0 0 0 0 0 0 9 0 0 0;...0 0 0 0 0 0 0 0 0 0 0 0;...0
0 0 0 0 0 0 0 0 0 0 0;...0 0 0 0 0 0 0 0 0 0 0 4;...0 0 0 0 0 0 0 0 0 0 0 0]
c =
   0   0   0   8   0   0   0   0   0   0   0   0
   0   0   0   0   1   0   0   0   0   0   0   0
   0   0   0   0   2   0   0   0   0   0   0   0
   0   0   0   0   0   4  14  11   0   0   0   0
   0   0   0   0   0   1   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   9   8   0
   0   0   0   0   0   0   0   0   9   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   4
   0   0   0   0   0   0   0   0   0   0   0   0
» maxtree(c)
ans =    4   7;   4   8;   8   9;   7   10;   1   4;   7   11;   4   6;   11   12;   3   5;   2   5;   5   6
» [z,cost]=maxtree(c)
z =   4   7;   4   8;   8   9;   7   10;   1   4;   7   11;   4   6;   11   12;   3   5;   2   5;   5   6
cost = -71
```

D.2. Minimum Undirected Spanning Tree

```
% Finds the minimum spanning tree of an undirected graph. % function [Z_tree, cost] = mintree(C, Zin, Zout); % input:% C Cost (distant) matrix where C = C' and C(i,i) = NaN; % Zin  Arcs forced in. Arc list.; % Zout Arcs forced out. Arc list. (could also be given as NaN in C); % output: % Z_tree The minimum undirected spanning tree. Arc list.; % cost       Total cost.
function [Z_tree, cost] = mintree(C, Zin, Zout)
if nargin < 3
  Zout = [];
  if nargin < 2
    Zin = [];
  End; end;
for i = 1:size(Zout,1); % Exclude the arcs in Zout from C ;
  C(Zout(i,1),Zout(i,2)) = nan;
  C(Zout(i,2),Zout(i,1)) = nan;
end
m = size(C,1);  % Number of nodes
Z_tree = [];    % The minimum undirected spanning tree
cost = 0;       % The minimum undirected spanning tree cost
setlist = 1:m;  % Let all nodes be members of different sets
for i = 1:size(Zin,1); % Include the arcs in Zin in Z_tree;
  Z_tree = [Z_tree;Zin(i,:)];
  cost = cost+C(Zin(i,1),Zin(i,2)); % Update setlist; %newset = min(setlist(Zin(i,1)),setlist(Zin(i,2)));
  newset = setlist(Zin(i,1)); %index = find(max(setlist(Zin(i,1)),setlist(Zin(i,2)))==setlist);
  index = find(setlist(Zin(i,2))==setlist);
  setlist(index) = newset*ones(1,length(index));
C(Zin(i,1),Zin(i,2)) = nan; % Exclude the arc from C;
  C(Zin(i,2),Zin(i,1)) = nan;
end
while size(Z_tree,1) < m-1
[row,col]=find(C==min(C(finite(C)))); % Find the arc with minimum cost;
if setlist(row(1))~=setlist(col(1)); % Add arc k if it doesnot give a cycle (nodes belong to different sets);
    Z_tree = [Z_tree;[row(1) col(1)]];
    cost = cost+C(row(1),col(1)); % Update setlist; %newset = min(setlist(row(1)),setlist(col(1)));
    newset = setlist(row(1)); %index = find(max(setlist(row(1)),setlist(col(1)))==setlist);
    index = find(setlist(col(1))==setlist);
    setlist(index) = newset*ones(1,length(index));
  end;
C(row(1),col(1)) = nan; % Exclude the arc from C ;
```

```
  C(col(1),row(1)) = nan;
End.
```

D.3. Maximum Weighted Directed Spanning Tree

```
function [arcs,tree] = mwdrst(A); %Maximum weighted directed spanning tree ; % [arcs,tree] = mwdrst(A); % A -
input directed graph in form of a matrix; % A(i,j) = 5 designates an arc from vertex; % i to vertex j, with cost 5; %
tree - weighted directed spanning tree of a directed graph; % Step 1 - ; % List all arcs
[nrows ncols] = size(A);
if nrows ~= ncols, error(’input matrix must be square’);end
if nrows < 3, error(’trivial case, less than 3 vertices’);end
narcs = 0;
for j=1:ncols
  for i=1:nrows
    if A(i,j) ~= 0
      narcs = narcs + 1;
      arcs(narcs,1) = i;              % source of arc
      arcs(narcs,2) = j;              % terminal vertex
      arcs(narcs,3) = A(i,j);         % cost
    end;    end; end;
rx_vertex = arcs(1,2);
tx_vertex = arcs(1,1);
max_cost  = arcs(1,3);
cnt = 0;
for i=2:mwdrst
  if (arcs(i,2) ~= rx_vertex)
    cnt = cnt + 1;
    tree(cnt,1) = tx_vertex;
    tree(cnt,2) = rx_vertex;
    tree(cnt,3) = max_cost;
    rx_vertex = arcs(i,2);
    tx_vertex = arcs(i,1);
    max_cost = arcs(i,3);
  else
    if arcs(i,3) > max_cost
      rx_vertex = arcs(i,2);
      tx_vertex = arcs(i,1);
      max_cost = arcs(i,3);
    end;  end;  end
cnt = cnt + 1;
tree(cnt,1) = tx_vertex;
tree(cnt,2) = rx_vertex;
tree(cnt,3) = max_cost;
if (cnt < nrows); % Step 2;%error(’this graph has no weighted directed spanning tree);
else
  [stree,ind] = sort(tree(:,3));
  tree = tree(ind(1:nrows-1),:);
end
loops = check_loops(tree,nrows); % Step 3; % check if there are loops;
if not(isempty(loops))
  [a b] = size(loops);
  fprintf(’There are %d loops\n’,b)
end
function [loops] = check_loops(atree,nrows)
atree = [atree zeros(size(1:nrows-1))’];
nloops = 0;
loops = [];
while sum(atree(:,4)) < nrows-1
  start = [];
  i = 0;
  while isempty(start)
    i=i+1;
    if atree(i,4)==0
      start = atree(i,1);
      next = atree(i,2);
      atree(i,4) = 1;            % done with this arc in tree
      loop = [];
      loop(1) = start;
      loop(2) = next;
      lp_cnt = 2;
    end;  end
  if (sum(atree(:,4)) < nrows-1 & i+1 <= nrows-1)
    repeat = 1;
    while repeat
      repeat = 0;
      for j=i+1:nrows-1
        if (atree(j,1) == next)
          atree(j,4)=1;
          next = atree(j,2);
          if (next == start)
            m = length(loop);
            loop = [loop zeros(size(1:nrows-m))]';
            loops = [loops loop];
          else
            lp_cnt = lp_cnt + 1;
            loop(lp_cnt) = next;
            repeat = 1;
          end
        end;       end;    end;  end;    end.
```

D.3.1. Output of a MWDRST

```
» C = [0 62 0 0 0 0 0 0 3 0;.0 0 38 0 0 0 24 0 0 0;.0 0 0 13 0 4 14 0 0 7;.0 0 0 0 6  0 0 15 0 0;.0 0 0 8 0 65 0 7 0 0;.0
0 0 0 0 0 0 0 0 4;.0 0 0 0 15 0 0 7 6 10;.0 0 0 0 29 0 0 0 0 9;.0 0 0 0 0 0 0 9 0 0;.0 0 0 0 30 0 0 0 0 0];
» save C;
» [arcs,tree] = mwdrst(C);
» tree
tree =1   1   0;   2   2   0;   1   3   0;   1   4   0;   1   5   0;   1   6   0;   1   7   0;   1   8  0;   2  9  0.
» C
C =
   0   62    0    0    0    0    0    0    3    0
   0    0   38    0    0    0   24    0    0    0
   0    0    0   13    0    4   14    0    0    7
   0    0    0    0    6    0    0   15    0    0
   0    0    0    8    0   65    0    7    0    0
   0    0    0    0    0    0    0    0    0    4
   0    0    0    0   15    0    0    7    6   10
   0    0    0    0   29    0    0    0    0    9
   0    0    0    0    0    0    0    9    0    0
   0    0    0    0   30    0    0    0    0    0
» [arcs,tree] = mwdrst(C);
» tree
tree =         1    2   62
               2    3   38
               3    4   13
              10    5   30
               5    6   65
               2    7   24
               4    8   15
               7    9    6
               7   10   10
```

D.4. Dijkastra Algorithm

```
% function [pred,dist] = dijkstra(s,P,Z,c);% input parameters;%s: The starting node s;%P: Pointer vector to start
```

each node in Z-matrix; %Z: Arcs outgoing from the nodes in increasing order;% Z(:,1) Tail. Z(:,2) Head.; %c: Costs related to the arcs in the Z-matrix; % output parameters;%pred: pred(j) = Predecessor of node j;%dist:distance from each node to s;

```
function [pred,dist] = dijkstra(s,P,Z,c)
fprintf('dijkstra: Start node %5.0f \n',s);
n = size(Z,1); % m = number of nodes. n = number of arcs.
m = length(P) - 1;
dist = Inf*ones(m,1);
pred = zeros(m,1);
dist(s) = 0;
S = s;
B = [1:s-1 s+1:m];
for j = P(s):P(s+1)-1
  node = Z(j,2);
  dist(node) = c(j);
  pred(node) = s;
  xprint(dist,'dist:',' %6.2f',10)
end
while length(S) ~= m
[min_dist k] = min(dist(B)); % Node selection;
  i = B(k);
  S = [S i];
  B = [B(1:k-1) B(k+1:length(B))];
  for arc = P(i):P(i+1)-1
    node = Z(arc,2);
    if dist(node) > dist(i)+c(arc)
      dist(node) = dist(i)+c(arc);
      pred(node) = i;
    end
    fprintf('(Node i, Node j, Arc) =%3d %3d %3d\n',i,node,arc)
    xprint(dist,'dist:',' %6.2f',10)
  end
end % while.
```

D.5. *QAP* Algorithm Codes – *SDPI* Solution

function[TC,a]=sdpi(a0,W,D,C);Steepest Descent Pairwise Interchange Heuristic.%[TC,a]=sdpi(a0,W,D,C); % Determines final assignment vector a with minimum total cost TC;% M = number of activities;% N = number of sites;% a0 = 1 x N initial assignment vector;% W = M x M weight matrix;% D = N x N site distance matrix;% C = M x N fixed cost matrix (optional);% Requires that M = N;% error checking

```
error(nargchk(3,4,nargin));
[a0c,N] = size(a0);
if a0c ~= 1  error('Argument a0 not a row vector.');
end
if any(sort(a0) ~= [1:N])  error('Argument a0 not a feasible assignment.');
end
[Dr,Dc] = size(D);
if Dr ~= Dc  error('Argument D not N x N square matrix.');
end
if Dr ~= N  error('N in argument D not the same as N in a0.');
end
[M,Wc] = size(W);
if M ~= Wc  error('Argument W not M x M square matrix.');
end
if N ~= M  error('M in argument W not the same as N in a0 and D.');
end
if nargin == 4
  if any(size(C) ~= [M,N])
    error('Argument C does not have same M and/or N as a0, W, and D.');
  end; end
```

% internal variables;% ba = best a;% ca = current a;% cba = current best a found;% bTC = best TC;% kswitch = will make an interchange;% si = i index of interchange;% sj = j index of interchange;% main

```
if nargin == 3
  bTC = sum(sum(W(a0,a0) .* D));
else
  D = D + diag(diag(ones(N)));
  bTC = sum(sum((W(a0,a0) + diag(diag(C(a0,:)))) .* D));
end
if nargout == 0
  disp(' ');
  disp('Initial TC =');disp(bTC);
  disp('Initial  a =');disp(a0);
end
ba = a0;
kswitch = 1;
TCM = zeros(M,M);
while kswitch == 1
  kswitch = 0;
  for i = 1:M
    for j = i+1:M
      ca = ba([1:i-1 j i+1:j-1 i j+1:M]);
      if nargin == 3
        cTC = sum(sum(W(ca,ca) .* D));
      else
        cTC = sum(sum((W(ca,ca) + diag(diag(C(ca,:)))) .* D));
      end
      TCM(i,j) = cTC;
      if cTC < bTC
        bTC = cTC;
        cba = ca;
        kswitch = 1;
        si = i; sj = j;
      end;     end;   end;
 if kswitch == 1; % TCM;
     ba = cba;
     if nargout == 0
       disp(sprintf('Interchange %.0f and %.0f:',si,sj));
       disp('  Current TC =');disp(bTC);
       disp('  Current  a =');disp(ba);
     end;  end;   end;
 if nargout == 0; % output final results;
   disp('Final TC =');disp(bTC);
   disp('Final  a =');disp(ba);
 else
   TC = bTC;
   a = ba;   end.
```

D.5.1. *SDPI* Output

```
» d = [0 1 2 1 2 3;1 0 1 2 1 2; 2 1 0 3 2 1;1 2 3 0 1 2;2 1 2 1 0 1;3 2 1 2 1 0]
d =         0   1   2   1   2   3
            1   0   1   2   1   2
            2   1   0   3   2   1
            1   2   3   0   1   2
            2   1   2   1   0   1
            3   2   1   2   1   0
» w = [0 4 6 2 4 4;4 0 4 2 2 8; 6 4 0 2 2 6;2 2 2 0 6 2;4 2 2 6 0 10;4 8 6 2 10 0]
w =         0   4   6   2   4   4
            4   0   4   2   2   8
            6   4   0   2   2   6
            2   2   2   0   6   2
            4   2   2   6   0  10
            4   8   6   2  10   0
» a0 = [2 4 5 3 1 6]
a0 = 2    4    5    3    1    6
```

```
» cc=zeros(6)
cc =        0   0   0   0   0   0
            0   0   0   0   0   0
            0   0   0   0   0   0
            0   0   0   0   0   0
            0   0   0   0   0   0
            0   0   0   0   0   0
» [tc,a]=sdpi(a0,w,d,cc);
» tc
tc = 184
» a
a = 2   6   5   3   1   4
» [tc,a]=sdpi(a0,w,d);
» tc
tc =  184
```

D.6. *TSP* Using Branch and Bound Algorithm

```
% function [Tour, f_tour, OneTree, f_tree, w_max, my_max, optPar] = ;% salesman(C, Zin, Zout, my f_BestTour, optPar); % input:% C Cost (distance) matrix where C = C' and C(i,i) = NaN;% Zin  Arcs forced in. Arc list.;% Zout Arcs forced out. Arc list. (could also be given as NaN in C);% my Lagrange multipliers;% f_BestTour Cost (total distance) of a known tour ; % optPar  User specified parameters (see goptions.m);% salesman.m is using:% optPar(1) Print level : <= 5 No output; >5 Convergence results; % >6 Output every iteration; (<=4 could be used in a B&B-rutin;% calling salesman as a subroutine);% optPar(15) Maximal duality gap in percent of best dual objective;% optPar(25)  Fail tolerance for dual objective ;% optPar(18) Step length parameter eps; % optPar(21) Maximal # of iterations the dual objective can fail to ;% increase until eps is reduced ;% optPar(27) Maximal # of iterations the dual objective can fail to ;% increase until routine terminates ;% optPar(14) Maximal # of iterations. max(10*dim(x),100) is default.;% optPar(24) Wait flag, pause each iteration if set true;% output:;% Tour  Best tour found. Arc list.;% f_tour Cost (total distance) of best tour found;% OneTree Best 1-tree found. Arc list. ;% f_tree Cost of best 1-tree found;% w_max Best dual objective; % my_max Lagrange multipliers at w_max; % optPar(28) Exit flag :;% = 0  => OK;%  = 1 => Max # of iterations. ;%  = 2  => Infeasible problem, no tour exists.
function [Tour, f_tour, OneTree, f_tree, w_max, my_max, optPar] = salesman(C, Zin, Zout, my, f_BestTour, optPar)
if nargin < 6
  optPar=[];
  if nargin < 5
    f_BestTour=[];
    if nargin < 4
      my=[];
      if nargin < 3
        Zout=[];
        if nargin < 2
          Zin=[];
        End;      end;     end;  end;   end;
m = size(C,1);      % Number of nodes
if isempty(f_BestTour), f_BestTour = inf; end
if isempty(my)        , my = zeros(m,1) ; end
if isempty(optPar)
  optPar=lpDef; % Must be corrected!
  optPar(1) = 7;
  optPar(14) = max(10*m*(m-1),100);
  optPar(15) = 0.0001;
  optPar(18) = 0.1; % Should be near 2 if f_upper was better
  optPar(21) = 3;
  optPar(24) = 0;
  optPar(25) = 0.01;
  optPar(27) = 5;
end
PriLev = optPar(1); % Print level
Wait = optPar(24);  % Pause flag
C = C+diag(nan*ones(m,1)); % Check so C is OK
differ = C-C';
if max(max(abs(differ(finite(differ))))) > 0
```

```
  if PriLev>5
    fprintf('\n--- Algorithm terminates.\n')
    fprintf('   Matrix C is not symmetric, no solution found.\n')
  end
  optPar(28)=2;
  return
end
for i = 1:size(Zout,1); % Exclude the arcs in Zout from C
  C(Zout(i,1),Zout(i,2)) = nan;
  C(Zout(i,2),Zout(i,1)) = nan;
end
nodedegree = sum(finite(C)); % Number of arcs connected to each node
ndeg_in = zeros(1,m); % Number of arcs connected to each node in Zin
if size(Zin,1)>0
  for i = 1:m
    ndeg_in(i) = length(find(Zin==i));
  end;  end;
setlist = 1:m; % Let all nodes be members of different sets; % Are the arcs in Zin giving a cycle?
cycle = 0;    % Cycle flag
for i = 1:size(Zin,1)
  if setlist(Zin(i,1))~=setlist(Zin(i,2)); % Update setlist if the nodes belongs to different sets
    newset = min(setlist(Zin(i,1)),setlist(Zin(i,2)));
    index = find(max(setlist(Zin(i,1)),setlist(Zin(i,2)))==setlist);
    setlist(index) = newset*ones(1,length(index));
  else
    if i==m&i==size(Zin,1)
      cycle = 2; % The m'th and last arc gave a cycle, i.e. a tour
    else
      cycle = 1;
    end
    break % A cycle is found
  end; end
if min(nodedegree)<2|max(ndeg_in)>2|cycle==1
  Tour = [];
  f_tour = inf;
  OneTree = [];
  f_tree = inf;
  w_max = -inf;
  my_max= [];
  if PriLev>5
    fprintf('\n--- Algorithm terminates.\n')
    fprintf('   Problem infeasible, no solution found.\n')
    if min(nodedegree)<2
      fprintf('   There is a node with less than two arcs (caused by Zout?).\n')
    elseif max(ndeg_in)>2
      fprintf('   More than two arcs are connected to the same node in Zin.\n')
    elseif cycle==1
      fprintf('   The arcs in Zin gave a cycle.\n')
    end;  end
  optPar(28)=2;
  return
elseif cycle==2
  Tour = Zin;
  f_tour = 0;
  for i = 1:m
    f_tour = f_tour+C(Tour(i,1),Tour(i,2));
  end
  OneTree = Tour;
  f_tree = f_tour;
  w_max = -inf;
  my_max= [];
  if PriLev>5
```

```
    fprintf('\n--- Algorithm terminates.\n')
    fprintf('    The arcs in Zin is a tour with the cost %.6f.\n',f_tour)
  end
  optPar(28)=0;
  return
end
w_max = -inf;          % Best dual objective found so far
iter = 0;              % Number of iterations since w_max was found
my_max = [];           % Lagrange multipliers at w_max
Tour = [];             % Best tour found so far
f_tour = inf;          % Best tour cost found so far
OneTree = [];          % Best 1-tree found so far
f_tree = inf;          % Best 1-tree cost found so far
max_iter = optPar(27);  % Maximal # of iterations the dual can fail to incr.
max_loop = optPar(14);  % Maximal # of iterations
w_last = inf*ones(1,max_iter); % Last # of w
DualGap = optPar(15);  % Maximal duality gap in percent of w_best
DObjTol = optPar(25);  % Fail tolerance for dual objective
eps = optPar(18);      % Step length parameter in the dual problem
RedEps = optPar(21);   % Reduce eps if w_best fails to incr. in RedEps iter
f_upper = 0; % Best tour upper bound.
not_jet = ones(1,m);
for i=1:size(Zin,1)
  f_upper = f_upper+C(Zin(i,1),Zin(i,2));
  not_jet(Zin(i,1)) = 0;
end
for i=1:m
  if not_jet(i)==1
    f_upper = f_upper+max(C(finite(C(:,i)),i));
  end;  end;
if size(Zin,1)>0
  index = find(Zin(:,1)~=m&Zin(:,2)~=m);
  Zinm_1 = Zin(index,:);
else
  Zinm_1 = [];
end
if size(Zout,1)>0
  index = find(Zout(:,1)~=m&Zout(:,2)~=m);
  Zoutm_1 = Zout(index,:);
else
  Zoutm_1 = [];
end
loop = 0;
while loop < max_loop % Main loop starts
  loop = loop+1;
C_hat = C;
  for i = 1:m
    C_hat(:,i) = C_hat(:,i)-my(i)*ones(m,1);
    C_hat(i,:) = C_hat(i,:)-my(i)*ones(1,m);
  end
[Z_tree,c_tree] = mintree(C_hat(1:m-1,1:m-1),Zinm_1,Zoutm_1);
Zm = C_hat(:,m);
  if size(Zin,1)>0
    index = find(Zin(:,1)==m|Zin(:,2)==m); % These arcs must be in
    Zm(min(Zin(index,:)')) = -inf*ones(length(index),1);
  end
  [Zm_sort,index] = sort(Zm);
  Z_1tree = [Z_tree;[index(1) m];[index(2) m]];
  c_1tree = c_tree+C_hat(index(1),m)+C_hat(index(2),m);
 for i = 1:m
    deg(i) = length(find(Z_1tree==i));
  end
  deg=deg(:);
f_org = 0;
  for i=1:m
    f_org = f_org+C(Z_1tree(i,1),Z_1tree(i,2));
  end
if f_org<f_tree
    f_tree = f_org;
    OneTree = Z_1tree;
  end
if min(deg)==2&max(deg)==2&f_org<f_tour
    f_tour = f_org;
    Tour = Z_1tree;
    f_upper = f_org;
  end
w = 2*sum(my)+c_1tree; % The dual objective
  w_last = [w w_last(1:max_iter-1)];
  if w>w_max & iter>0
    w_max = w;
    my_max = my;
    iter = 1;
  else
    iter = iter+1;
  end
g = -(deg-2*ones(m,1)); % Subgradient
  g_norm = norm(g);
  if g_norm>0.0001
    t = eps*(f_upper-w)/g_norm; % Should t be resticted to prevent big steps.
    my = my+t*g/g_norm;
  else
    my = my+g; % Take a one step, just to prevent dividing by zero.
  end
if iter >= RedEps
    eps = eps/2;
  end
if (max(w_last)-min(w_last))<=DObjTol | w_max>=f_BestTour | ...
     (((f_upper-w)<=DualGap*w_max)&iter>=2)
    if PriLev>5
      fprintf(\n--- Algorithm terminates. ITER = %d \n’,loop)
      if finite(f_tour)
        fprintf(’   Total cost of best tour found is %.6f.\n’,f_tour)
        fprintf(\n   Best dual objective found is %.6f.\n’,w_max)
        fprintf(’   The duality gap is %.3f percent’,(f_upper-w)/w_max)
        fprintf(’ of the best dual objective.\n’)
        fprintf(\n   Best tour found is : \n’)
        mPrint(Tour’,Tour’,’ %4.0f’,15);
      else
        fprintf(’   No tour is found.\n’)
        fprintf(’   Best dual objective found is %.6f.\n’,w_max)
      end;   end
    optPar(28)=0;
    return
  end
if PriLev>6
    fprintf(\n Iteration %d \n’,loop)
    fprintf(’  Dual objective : %.6f. ’,w)
    if finite(f_tour)
      fprintf(\n   Total cost of best tour found is %.6f.\n’,f_tour)
    else
      fprintf(’   No tour is found.\n’)
    end;  end
if Wait&PriLev>6
    pause
  end;  end
if PriLev>5
  fprintf('\n--- Algorithm terminates.\n')
  fprintf('   Too many iterations. ITER = %d \n',loop)
  if finite(f_tour)
    fprintf('   Total cost of best tour found is %.6f.\n',f_tour)
    fprintf('   Best dual objective found is %.6f.\n\n',w_max)
    fprintf('   The duality gap is %.3f percent',(f_upper-w)/w_max)
    fprintf(' of the best dual objective.\n')
    fprintf('\n    Best tour found is : \n')
    mPrint(Tour',Tour',' %4.0f',15);
  else
    fprintf('   No tour is found.\n')
    fprintf('   Best dual objective found is %.6f.\n',w_max)
  end;  end;
optPar(28)=1;
```

D.6.2. *TSP* Branch and Bound Routine (Balas Method)

% Branch & Bound algorithm for binary IP using Balas method.;% Solving binary IP in the form: min c'*x subject to A x <= b, x 0/1.;% The elements in A, b and c are restricted to be integers.;% The first meq= optPar(13) are equalities.; % function [x, optPar] = balas(A, b, c, optPar); % input: % OptPar is the standard parameter vector defined by lpDef and goptions;% If optPar is empty, lpDef is called. ;% balas is using:% optPar(1) Print level: =0 No output; >0 Convergence results; % >1 Output every iteration;% optPar(13) meq = Number of equality constraints (the first meq equations);% optPar(14) Maximal number of iterations. max(10*dim(x),100) is default.;% optPar(24) Wait flag, pause each iteration if set true ;% output:% x Solution x;% optPar(28) Exit flag.;% = 0 => OK;% = 1 => Max # of iterations. No solution found.;% = 2 => Problem infeasible.;% optPar(8) Objective c'*x at optimum x (or last iterate x if no convergence)

```
function [x, optPar] = balas(A, b, c, optPar)
if nargin < 4
  optPar = [];
end
b = b(:); c = c(:);
if is empty(optPar),  optPar = lpDef;            end;
if length(optPar)<29,optPar = lpDef([],[],optPar); end ;
PriLev = optPar(1);
wait = optPar(24);
meq = optPar(13); % The first meq constraints are equalities
A = [-A(1:meq,:);A];
b = [-b(1:meq,:);b];
n = length(c);
m = length(b);
max_iter = optPar(14);
if max_iter==0, max_iter = max(10*n,100); end
f_best = inf;     % Best feasible solution found so far, objective value
x = [];           % Best feasible solution found so far
pred = 0;         % Predictor (=0: root of branch tree)
problem = [0 nan]; % Variable number "0" is in this subproblem set to "nan"
L = 1;            % List of nodes still to branch
No1 = 0;          % Number of variables fixed to one at L
AA = [c';A];
bb = [f_best-1;b]; % Look for a PFS that is better
m = m+1;
for iter=1:max_iter % Main loop starts
if PriLev > 1
    fprintf('\n= branch =   Iteration %d\n\n',iter);
  end
if length(L)=0
    if f_best=inf % Empty feasible set
      optPar(28)=2;
      if PriLev > 0
        fprintf(' No Feasible Solution to IP Problem\n')
      end
    else
      optPar(28)=0;
      if PriLev > 0
        fprintf('\n--- Optimal Solution Found! ITER = %d \n',iter)
        fprintf('\n   Optimal Objective = %d \n',f_best)
        xprint(x,' x = ','%6.1f',10);
      end;     end
    return % Terminate program
  end
[maxones,index] = max(No1);
  j = L(index);
  L = [L(1:index-1);L(index+1:length(L))];
  No1 = [No1(1:index-1);No1(index+1:length(No1))];
  jj = j;  % This problem
  J0 = []; % Variables set to zero
  J1 = []; % Variables set to one
  while pred(j) > 0
    if problem(j,2)==0
      J0 = [J0 problem(j,1)];
    elseif problem(j,2)==1
      J1 = [J1 problem(j,1)];
    end
    j = pred(j);
  end
bb(1) = f_best-1;
  LHS = -bb;  % Left Hand Side: should be less than zero when feasible
  Jfree = []; % Variables not yet fixed
  for i=1:n
    if any([J0,0]==i)
      % Add zero to LHS
    elseif any([J1,0]==i)
      LHS = LHS+AA(:,i);
    else
      LHS = LHS+min([zeros(m,1)';AA(:,i)'])';
      Jfree = [Jfree i];
    End;   end
 if PriLev > 1
    if ~isempty(J0)
      xprint(J0,'Index of Variables Fixed to Zero:','%6.0f',6);
    else
      fprintf('No Variables Fixed to Zero.\n')
    end
    if ~isempty(J1)
      xprint(J1,'Index of Variables Fixed to One: ','%6.0f',6);
    else
      fprintf('No Variables Fixed to One.\n')
    end;  end
 ENDTREE = 0; % Flag : end of branch tree
   if max(LHS)>optPar(4) % Subproblem is infeasible
     ENDTREE = 1;
 else % Subproblem is feasible
     if length(Jfree)==0 % All variables are fixed
       x_k = zeros(n,1);
       x_k(J1) = ones(length(J1),1);
       if c'*x_k<f_best % Is this the best solution found so far?
         f_best = c'*x_k;
         optPar(8) = f_best;
         x = x_k;
         if PriLev > 1
           fprintf('Update Best PFS Found: %d\n',f_best)
           xprint(x,' x = ','%6.1f',10);
         end;       end;       ENDTREE = 1;
```

```
else % there are still variables not fixed
BRANCH = 1; % Flag : shall we branch?
      for i=1:length(Jfree)
        [maxviolation,ii]=max(LHS+abs(AA(:,Jfree(i))));
        if maxviolation>optPar(4) % Construct one new subproblem
          if is empty(L)
            L = [size(problem,1)+1];
          else
            L = [L;size(problem,1)+1];
          end
          if is empty(No1)
            No1 = max ones;
          else
            No1 = [No1;max ones];
          end
          if AA(ii,Jfree(i))>0
            problem = [problem;[Jfree(i) 0]]; % Variable set to zero
            pred = [pred;jj];
          else
            problem = [problem;[Jfree(i) 1]]; % Variable set to one
            pred = [pred;jj];
          end
          BRANCH = 0;
          break
        end;       end
if BRANCH % construct two new subproblems
fixvar = Jfree(1);
        if is empty(L)
          L = [size(problem,1)+1;size(problem,1)+2];
        else
          L = [L;size(problem,1)+1;size(problem,1)+2];
        end
        if is empty(No1)
          No1 = [max ones;max ones+1];
        else
          No1 = [No1;max ones;max ones+1];
        end
        problem = [problem;[fixvar 0];[fixvar 1]];
        pred = [pred;jj;jj];
      end;     end;   end
if PriLev>1 & ENDTREE
    fprintf(' End of Tree.\n')
  end
  if PriLev>2
    if ~is empty(L)
      xprint(L(:),' New List L:  ','%6.0f',6);
    else
      fprintf(' List L is Empty. \n\n')
    end;   end
if wait & PriLev>1
    pause
  end;   end % Main loop
if PriLev>0
  fprintf('\n--- TOO MANY ITERATIONS. ITER = %d \n',iter)
end;
optPar(28)=1;
```

D.7. *TSP SA* Baseline Codes

```
function [S,cost] = tsp_sa(x,stop_crit); % Travelling salesmen problem - SA; % S = tsp_sa(x); % x - cities; % S -
sequence of cities; % cost
n = length(x);
```

```
S = [1:n];
cost = cities_cost(S,x);
incr = 1e10;
i = 0;
dzaba = 0;
while (incr > stop_crit & dzaba < 70)
  S1 = gen_seq(S);
  cost1 = cities_cost(S1,x);
  if (rand < 0.35 & cost1 < cost)
    incr = (cost - cost1);
    S = S1;
    cost = cost1;
    dzaba = 0;
  else
    dzaba = dzaba + 1;
  end
  i = i +1;
  fprintf('%d cost = %f incr = %f\n',i,cost,incr);
end
plot(x(2,:),x(3,:),'o');
hold on;
for i=1:n-1
  ind1 = S(i);
  ind2 = S(i+1);
  plot([x(2,ind1),x(2,ind2)],[x(3,ind1),x(3,ind2)]);
end
hold off;
function cost = cities_cost(S,x);
% cost = cities_cost(S,x); % S - sequence of cities; % x - id, coordinates of cities;
n = length(S);
total_dist = 0;
for i=1:n-1
id1 = S(i);
id2 = S(i+1);
dist = sqrt((x(2,id1) - x(2,id2))^2 + (x(3,id1)-x(3,id2))^2);
  total_dist = total_dist + dist;
end
cost = total_dist;
function S = gen_seq(S0);
S = S0; % S = gen_seq(S0);% from input sequence S0 generate;% another slightly different S1;
n = length(S0);
n_swaps = n/10+1;
for i=1:n_swaps
  ind1 = round(rand*n) + 1;
  if (ind1 > n) ind1 = n; end
  ind2 = round(rand*n) + 1;
  while ind1 == ind2
    ind2 = round(rand*n) + 1;
  end
  if (ind2 > n) ind2 = n; end
  S1 = S;
  Y = S1(ind1);
  S1(ind1) = S1(ind2);
  S1(ind2) = Y;
  S = S1;
End.
```

D.7.1. Improved *SA - TSP* with 2-opt

function Best_tour_length= tspsiman(EUC_2D);% A Symmetric 2D Euclidian TSP; % Nearest Neighbour tour construction + 2-Opt local search + SA/Metropolis test;% This function makes use of the following function/s and data file/s:;% tourdist; % Input;% Weights or lengths must be given in x-y coordinates (EUC_2D). Data file must be saved as a text file for it to be processed by the MATLAB program. ;% Output;% Best_tour_length; %

```
Temperature_of_best_tour_length;% Solution_count; % Search_stop_temperature;% Elapsed_time % In seconds; %
Solutions_generated - number of solutions generated;% Floating_point_operations; % Graphs;% Temperature vs.
Tour Length;% Number of solutions vs. Tour Length.
flops(0) ;
t0= clock ;
xy= EUC_2D ;
n_cities= length(xy) ;
rand('state',sum(100*clock));
distance_matrix = zeros(n_cities) ;
for n_cities_x = 1: n_cities,
    for n_cities_y = 1:n_cities_x
        x = xy(n_cities_x, 1) ;
        y = xy(n_cities_x, 2) ;
        xx = xy(n_cities_y, 1) ;
        yy = xy(n_cities_y, 2) ;
        distance_matrix(n_cities_x, n_cities_y)= ceil(sqrt((x - xx)^2 + (y - yy)^2)) ;
        distance_matrix(n_cities_y, n_cities_x)= distance_matrix(n_cities_x, n_cities_y) ;
            end;   end
lenbestNN= inf ;
pbestNN= [] ;
prand= randperm(n_cities) ; % A random selection of the starting city
                    f=find(prand==1) ;
                    prand(f)= prand(1) ;
            p= [1 prand(2)] ;
            i= prand(3) ;
            count= 3 ;
            while count <= n_cities
            NNdist= inf ;
            pp= i ;
            for j= 1: n_cities
            if (distance_matrix(i, j) < NNdist) & (j~=i) & ((j~=p) == ones(1,length(p)))
             NNdist= distance_matrix(i, j) ;
             pp= j ;
            end;                end;
            p= [p pp ] ;
     i= pp ;
            count= count + 1 ;
            end
            len= tourdist(p, distance_matrix) ;
            if len < lenbestNN
                    lenbestNN= len ;
                    pbestNN= p ;
            end
solnn= [] ;
lenn= [] ; temp= [] ;
soln= 1 ;
lencurr= lenbestNN;
Best_tour_length= lenbestNN
pcurr= pbestNN ;
pbest= pbestNN ;
restart= 1 ;
Tstart= 30 ; % Start temperature
Tend= 1 ; % Stop temperature
Tred= 0.97 ;
T= Tstart ;
Nochange= 2 ; % If after Nochange neighborhood searches, no improvements break search.
lenn= [lenn lencurr] ;
temp= [temp T] ;
solnn= [solnn soln] ;
bb= 0 ;
while T >= Tend
            big= n_cities - 1 ;
```

```
   while big >= 3
            small= big - 2 ;
            while small >= 1
   curropt= distance_matrix(pcurr(big),pcurr(big+1)) + distance_matrix(pcurr(small),pcurr(small+1)) ;
    swap2= distance_matrix(pcurr(small),pcurr(big)) + distance_matrix(pcurr(small+1),pcurr(big+1)) ;
                    soln= soln + 1 ;
                    if swap2 < curropt
                            order2= 1: n_cities ;
                            order2=[1:small big:-1:small+1 big+1:n_cities] ;
                            pcurr= pcurr(order2) ;
                            lencurr= tourdist(pcurr, distance_matrix) ;
                            lenn= [lenn lencurr] ;
                            temp= [temp T] ;
                            solnn= [solnn soln] ;
                            if lencurr < Best_tour_length
                                    Best_tour_length= lencurr
                                    pbest= pcurr ;
                                    Temperature_of_best_tour_length= T
                                    Solution_count= soln
                                    T= Tred * T ;
                                    if T <= 3
                                            T= 50 ;
                                    End;    end
                             Tcurr= T ;
                    bb= 0 ;
                             big= n_cities - 1 ;
                             small= big - 1 ;
                             if T <= 3
                                     T= 10 ;
                             end
                             if T <= Tend ;
                                     big= 2.9 ;
                                     break
                             end
                    else if swap2 > curropt
   %r= abs(randn) ;
   r= rand; % where r ranges from 0.0 to 1.0
   diff= swap2 - curropt ;
   %if r < exp(-(diff) / T)
            if r <= exp(-(diff) / T)
                                     order2= 1: n_cities ;
                                     order2=[1:small big:-1:small+1 big+1:n_cities] ;
                                     pcurr= pcurr(order2) ;
                     lencurr= tour dist(pcurr, distance_matrix) ;
                                     T= Tred * T ;
                                     bb= 0 ;
                             end;    end
            small= small - 1 ;
            end
            big= big - 1 ;
    end
    bb= bb + 1 ;
    if T <= Tend | bb > No change ;
            clc
            Best_tour_length
            Best tour= [pbest -pbest(1)]
            Temperature_of_best_tour_length
            Solution_count
            Search_stop_temperature= T
            Elapsed_time= etime(clock, t0) % In seconds
Solutions_generated= soln
Floating_point_operations = flops
```

```
                if bb > No change
                        No_change= bb
                end
                disp('Press ENTER to display plot (Temperature vs. Tour Length) or Ctrl^C to end search.')
                pause
                clc
                plot(temp, lenn)
                title('Simulated Annealing w/ 2-Opt local search')
                xlabel('Temperature (not scaled)')
                ylabel('Tour Lengths/Costs')
                grid
        disp('Press ENTER to display plot (Number of Solutions vs. Tour Length) or Ctrl^C to end search.')
                pause
                clc
                plot(solnn, lenn)
                title('Simulated Annealing w/ 2-Opt local search')
                xlabel('Number of Solutions')
                ylabel('Tour Lengths/Costs')
                grid
                disp('Press ENTER to restart search (if var restart > 0) or Ctrl^C to end search.')
                pause
                if restart > 0
                        clc
                        T= Tstart ; bb= 0 ;
                        solnn= []; lenn= []; temp= [] ;
                        % This time randomly generate tours and restart annealing
                        prand= randperm(n_cities) ;
                                f=find(prand==1) ;
                                prand(f)= prand(1) ; prand(1)= 1 ;
                  lencurr= Best_tour_length        pcurr= pbest ;
                end ;     end; end; % End of local search
function tour_distance = tour dist(tourvec, distance_matrix);
n_cities = length(tourvec);
city = 1;
tour_distance = 0;
while city <= (n_cities - 1),
  tour_distance = tour_distance + distance_matrix(tourvec(city), tourvec(city+1));
  city = city + 1;
end; tour_distance = tour_distance + distance_matrix(tourvec(n_cities), tourvec(1));
```

D.8. Genetic Algorithm – *TSP* Order Based Example Routine

```
% This routine code shows how to use the ga using an order-based representation.; % Setting the seed to the same
for binary
rand('seed',156789)
% 6 city problem
t=[ 92.6112 59.0801; 49.1155 50.0000; 12.5000 57.9436; 75.0000 19.3703; 8.6504 13.7113; 36.8786 92.9628];
t=100*rand(15,2);
sz=size(t,1);
distMatrix=dists(t,t);
% Order-based Representation Crossover Operators;% cyclicXover; % erXover; % enhancederXover; %
linerorderXover; % orderbasedXover; % partmapXover; % singleptXover; % uniformXover;
xFns = 'cyclicXover uniformXover partmapXover orderbasedXover '
xFns =[xFns,'singleptXover linerorderXover'];
% xFns = [xFns,'enhancederXover linerorderXover']; % xFns = [xFns,'linerorderXover singleptXover']
xOpts = [2;2;2;2;2;2];% 2; 2; 2; 2; 2; 2];
% Order-based Mutation Operators;% inversionMutation;% adjswapMutation; % shiftMutation; % swapMutation %
threeswapMutation;
mFns = 'inversionMutation adjswapMutation shiftMutation swapMutation threeswapMutation';
mOpts = [2;2;2;2;2];
termFns = 'maxGenTerm'; % Termination Operators;
termOps = [100]; % 200 Generations
```

```
selectFn = 'normGeomSelect'; % Selection Function;
selectOps = [0.08];
evalFn = 'tspEval'; % Evaluation Function;
evalOps = [];
type tspEval
bounds = [sz]; % Bounds on the number of cities in the TSP;
gaOpts=[1e-6 1 1]; % GA Options [epsilon float/binar display];
startPop = initializeoga(80,bounds,'tspEval',[1e-6 1]); % Generate an initialise population of size 20
pause; % Lets run the GA; %Hit a return to continue;
[x endPop bestPop trace]=ga(bounds,evalFn,evalOps,startPop,gaOpts,...
   termFns,termOps,selectFn,selectOps,xFns,xOpts,mFns,mOpts);
x; % x is the best solution found;
pause; % endPop is the ending population; endPop; %Hit a return to continue; %pause; % bestPop is the best
solution tracked over generations
bestPop
pause
trace; % trace is a trace of the best value and average value of generations;
pause
clf; % Plot the best over time;
plot(trace(:,1),trace(:,2));
pause; % Add the average to the graph
hold on
plot(trace(:,1),trace(:,3)); %Hit a return to continue;
pause; % figure(2)
clf
A=ones(sz,sz);
A= xor(triu(A),tril(A));
[xg yg]=gplot(A,t);
clf
plot(xg,yg,'b:','MarkerSize',24);
h=gca;
hold on
ap=x;
plot(t(x(1:sz),1),t(x(1:(sz)),2),'r-')
plot(t([x(1),x(sz)],1),t([x(1),x(sz)],2),'r-')
plot(xg,yg,'b.','MarkerSize',24);
j=1;
for i=1:sz
  str=sprintf('C-%d',j);
  if t(i,1)<50
   j=j+1;
   text(t(i,1)-7,t(i,2),str);
  else
   j=j+1;
   text(t(i,1)+2,t(i,2),str);
  end;   end;
 legend('Path','Best Found Path')
```

D.9. *SA* Layout Placement 1

```
function [mach,cost] = place(arcs,area); % Automatic placement via Simulated Annealing; % Input: arcs - array of
arcs described by ; %   * start vertex (machine); %   * end vertex (machine); %* flow; % * cost; % area  - a x b
(assumption is that each machine is; %  2x2 and therefore area must be greater; % or equal the total area of all
machines; % Output: mach  - location (placement) of vertices (machines); % example; % load(arcs); % loc =
place(arcs,[10 10]); % list first all vertices (machines)
[A n] = size(arcs);
if (n ~= 4) error('wrong input arcs');end
nmach = 2;
mach(1,1) = arcs(1,1);
mach(2,1) = -1;
mach(1,2) = arcs(1,2);
mach(2,2) = -1;
```

```
for a=2:A
  for ind=1:2
    m = arcs(a,ind); % check if this machine is already in mach array
flag = 0;
    for n = 1:nmach
      if (mach(1,n) == m), flag =1; end
    end
    if (flag == 0)
      nmach = nmach + 1;
      mach(1,nmach) = m;
      mach(2,nmach) = -1;  % initial placement at 0,0
    end;    end; end; % check if area big enough;
given_area = area(1)*area(2);
if given_area < nmach, error('area too small');end; % Initial placement
i=0;
j=0;
while (sum(mach(2,:)== -1) > 0)
  a = fix(rand*(nmach-1) + 1.5);
  if (mach(2,a) == -1)
    mach(3,a) = i;  % y axis
    mach(2,a) = j;  % x axis
    if (j < area(2)-1)
      j = j + 1;
    else
      if (i < area(1)-1),
        i = i + 1;
        j = 0;
      end;      end;  end;  end
cost = placement_cost(mach,nmach,arcs,A);
new_mach = swap(mach,nmach);
cost1 = placement_cost(new_mach,nmach,arcs,A);
Tstart = 5*abs(cost-cost1); % Temperature
if (Tstart == 0), Tstart = 1;end;
Tend = Tstart/500;
Tred = 0.91;
T = Tstart;
l=0;
nsol = 0;
while (T >= Tend)
  l = l + 1;
  uphill_cnt = 0;
  for k=1:10*nmach/(T+2)
    if (rand < 0.75)
      new_mach = swap(mach,nmach);
    else
      new_mach = move(mach,nmach,area);
    end
    nsol  = nsol + 1;
    new_cost = placement_cost(new_mach,nmach,arcs,A);
    if new_cost < cost
      mach = new_mach;
      cost = new_cost;
    else
      r = rand;
      diff = new_cost - cost; % fprintf('      r = %f  e = %f\n',r,exp(-diff/T));
 if (r < exp(-(diff)/T))
        uphill_cnt = uphill_cnt + 1;
        mach = new_mach;
        cost = new_cost;
      end;     end;   end
  T = Tred * T;
  fprintf('Temp = %f  cost = %f uphill_cnt = %d\n',T,cost,...
    uphill_cnt);
  Temp(l) = T;
  Cost(l) = cost;
end
plot(Temp,Cost,'o-');grid
xlabel('Temperature');
ylabel('Placement cost');
function cost = placement_cost(mach,nmach,arcs,narcs);
cost = 0;
for i=1:narcs
m1 = arcs(i,1); % distance;
  ind1 = find(mach(1,:) == m1);
  p1x = mach(2,ind1);
  p1y = mach(3,ind1);
  m2 = arcs(i,2);
  ind2 = find(mach(1,:) == m2);
  p2x = mach(2,ind2);
  p2y = mach(3,ind2);
  d = abs(p1x - p2x) + abs(p1y - p2y);
  cost = cost + arcs(i,3)*arcs(i,4)*d;
end
function new_mach = swap(mach,nmach);
a = fix(rand*(nmach-1) + 1.5);
b = fix(rand*(nmach-1) + 1.5);
while (b == a)
  b = fix(rand*(nmach-1) + 1.5);
end
new_mach = mach;
new_mach(2,a) = mach(2,b);
new_mach(3,a) = mach(3,b);
new_mach(2,b) = mach(2,a);
new_mach(3,b) = mach(3,a);
function new_mach = move(mach,nmach,area);
a = fix(rand*(nmach-1) + 1.5);
nonempty = zeros(area(1)*area(2),2);
for n=1:nmach
  i = mach(2,n);
  j = mach(3,n);
nonempty(i+1,j+1) = 1; % fprintf('%d %d n=%d\n',i+1,j+1,n);
end
[ind1 ind2] = find(nonempty == 0);
L = length(ind1);
if L > 1
  k = fix(rand*(L-1) + 1.5);
else
  k=1;
end
new_mach = mach;
new_mach(2,a) = ind1(k)-1;
new_mach(3,a) = ind2(k)-1;
```

D.9.1. *SA* Layout Placement 1 - Output Example

```
example
 load(arcs);
 loc = place(arcs,[10 10]);
 » load arcs
» loc = place(arcs,[4 4])
Tstart =    20
Temp = 18.000000  cost = 27.000000 uphill_cnt = 85
Temp = 16.200000  cost = 36.000000 uphill_cnt = 84
Temp = 14.580000  cost = 35.000000 uphill_cnt = 88
.................................................................
```

```
.........................................................
Temp = 0.075142  cost = 14.000000 uphill_cnt = 0
Temp = 0.044371  cost = 14.000000 uphill_cnt = 1
loc =  Columns 1 through 14
  1   2   3   4   5   6   7   8   9  11  10  12  13  14
  1   1   2   0   1   2   3   0   1   3   2   3   0   2
  0   1   2   1   2   1   2   2   3   1   0   3   3   3
» loc'
ans =        1   1   0
             2   1   1
             3   2   2
             4   0   1
             5   1   2
             6   2   1
             7   3   2
             8   0   2
             9   1   3
            11   3   1
            10   2   0
            12   3   3
            13   0   3
            14   2   3
» arcs =     1   2   1   1
             2   3   1   1
             2   4   1   1
             2   5   1   1
             3   6   1   1
             3   7   1   1
             4   8   1   1
             5   9   1   1
             6  11   1   1
             6  10   1   1
             7  12   1   1
             8  13   1   1
             9  14   1   1
```

D.10. Improved *SA* Layout Placement

```
function [mach,cost] = placev2(nodes1,arcs,vacant_area);% [loc,cost] = place(nodes,arcs,area);% Automatic
placement via SA;% Input: nodes - array of nodes (machines) to be arranged;% on a rectilinear grid. Consists of;% *
node number;% * width;% * height;% arcs - array of arcs described by ;% * start vertex (machine);% * end vertex
(machine);% * flow;% * cost;% vacant_area - this is a 2D array which defines;% the area to be used for the layout;%
NB - vacant fields are denoted by 1,;% while forbidden fields are set to 0.;% Output: mach - location (placement)
of vertices (machines);% example;% load nodes.dat; % load arcs.dat; % area = ones(10,8); % loc =
place(nodes,arcs,area);
global nodes;
tt=cputime;
nodes = nodes1;
[nmach dummy] = size(nodes);
if (dummy ~= 3), error('wrong nodes array');end
 [A n] = size(arcs); % list first all vertices (machines)
if (n ~= 4) error('wrong input arcs');end
mach = zeros(3,nmach);
mach(1,:) = nodes(:,1)';
mach(2,:) = -ones(1,nmach); % initial placement (x)
mach(3,:) = -ones(1); % initial placement (y)
required_area = sum(nodes(:,2) .* nodes(:,3));
given_area = sum(sum(vacant_area)); % check if area big enough
if given_area < required_area, error('area too small');end
 [area1 area2] = size(vacant_area);
num_forb_fields = sum(sum(vacant_area==0));
floorplan = (-1)*(1-vacant_area); % occupancy of the area; % Initial placement
```

```
cnt=0;
while (sum(mach(2,:)== -1) > 0)
  cnt = cnt    +1;
  if cnt == nmach*100,
    error('Area too small for initial placement!!!');
  end
  a = fix(rand*(nmach-1) + 1.5);  % random node index
  if (mach(2,a) == -1)
    Width = nodes(a,2);
    Height = nodes(a,3);
    xloc = fix(rand*(area1-1) + 0.5);
    yloc = fix(rand*(area2-1) + 0.5);
    if (xloc+ Width <= area1& yloc+Height <= area2)
      vacant = 1;  % TRUE
      for g=1:Width
        for h=1:Height
          if not(floorplan(xloc+g,yloc+h)==0)
            vacant = 0;  % FALSE
            break
         end;        end;        end
      if (vacant)
        mach(3,a) = yloc;      % y axis
        mach(2,a) = xloc;      % x axis
        for g=1:Width
          for h=1:Height
            floorplan(xloc+g,yloc+h) = a;
          end;        end; %floorplan;    %report(mach,floorplan,vacant_area,[],[]);pause; end;   end;  end;end
cost = placement_cost(mach,nmach,arcs,A); %floorplan; %report(mach,floorplan,vacant_area,[],[]);pause
[new_mach,new_floorplan] = swap(mach,nmach,floorplan);
cost1 = placement_cost(new_mach,nmach,arcs,A);
%new_floorplan; %report(new_mach,new_floorplan,vacant_area,[],[]);pause; % -- Temperature Control --
Tstart = 4*abs(cost-cost1);
if (Tstart == 0), Tstart = 1;end;
Tend = Tstart/800;
Tred = 0.91;
T = Tstart;
l=1;
Temp(l) = T;
Cost(l) = cost;
nsol = 0;
while (T >= Tend)
  l = l + 1;
  uphill_cnt = 0;
  for k=1:20*nmach/(T+2)
    if (rand < 0.25); %fprintf('Checking SWAP\n');    %floorplan;
[new_mach,new_floorplan] = swap(mach,nmach,floorplan);
occup_area = sum(sum(new_floorplan ~= 0)); %new_floorplan;    %pause
      if (occup_area ~= num_forb_fields + required_area),
        error('bad floorplan- (swap)');
 end;  else; %fprintf('Checking MOVE\n');  %floorplan;  %report(mach,floorplan,vacant_area,[],[]);pause
      [new_mach,new_floorplan] = move(mach,nmach,floorplan); %new_floorplan; %pause;
    occup_area = sum(sum(new_floorplan ~= 0)); %report(new_mach,new_floorplan,vacant_area,[],[]); pause;
      if (occup_area ~= num_forb_fields + required_area),
        error('bad floorplan- (move)');
      end;     end
    nsol  = nsol + 1;
    new_cost = placement_cost(new_mach,nmach,arcs,A);
    if new_cost < cost
      mach = new_mach;
      cost = new_cost;
      floorplan = new_floorplan;
    else
```

```
      r = rand;
      diff = new_cost - cost; % fprintf('      r = %f  e = %f\n',r,exp(-diff/T));
if (r < exp(-(diff)/T))
        uphill_cnt = uphill_cnt + 1;
        mach = new_mach;
        cost = new_cost;
        floorplan = new_floorplan;
      end;     end;  end
  T = Tred * T;
  fprintf('Temp = %f  cost = %f uphill_cnt = %d\n',T,cost,...
    uphill_cnt);
  Temp(l) = T;
  Cost(l) = cost;
end
total_cpu_time = cputime-tt;
fprintf('CPU time (sec): %f\n',total_cpu_time);
floorplan
report(mach,floorplan,vacant_area,Temp,Cost);
function cost = placement_cost(mach,nmach,arcs,narcs);
global nodes;
cost = 0;
for i=1:narcs
m1 = arcs(i,1); % distance
  ind1 = find(mach(1,:) == m1); % central position of first machine;
p1x = mach(2,ind1) + nodes(ind1,2)/2;
  p1y = mach(3,ind1) + nodes(ind1,3)/2;
  m2 = arcs(i,2);
  ind2 = find(mach(1,:) == m2);
  p2x = mach(2,ind2) + nodes(ind2,2)/2;
  p2y = mach(3,ind2) + nodes(ind2,3)/2;
  d = abs(p1x - p2x) + abs(p1y - p2y); %fprintf('from %d %d costs %6.2f\n',ind1,ind2,d);
cost = cost + arcs(i,3)*arcs(i,4)*d;
end
function [new_mach, new_floorplan] = swap(mach,nmach,floorplan);
global nodes;
new_mach = [];
[area1 area2] = size(floorplan);
while is empty(new_mach)
  a = fix(rand*(nmach-1) + 1.5);
  b = fix(rand*(nmach-1) + 1.5);
  while (b == a)
    b = fix(rand*(nmach-1) + 1.5);
  end
Width_a = nodes(a,2);
 Height_a = nodes(a,3);
 xloc_a = mach(2,a);
 yloc_a = mach(3,a);
 Width_b = nodes(b,2);
 Height_b = nodes(b,3);
 xloc_b = mach(2,b);
 yloc_b = mach(3,b);
if (Width_a == Width_b & Height_a == Height_b); % Check sizes; % Identical; %fprintf('identical\n');
new_floorplan = floorplan;
      new_mach = mach;
      new_mach(2,a) = mach(2,b);
      new_mach(3,a) = mach(3,b);
      new_mach(2,b) = mach(2,a);
      new_mach(3,b) = mach(3,a);
      mask = (floorplan == b);
      new_floorplan = floorplan - (b-a)*mask;
      mask = (floorplan == a);
      new_floorplan = new_floorplan - (a-b)*mask;
 else;
if (Width_a == Height_b & Height_a == Width_b); % Need to rotate by 90 deg;
        %fprintf('rotate\n');
        new_floorplan = floorplan;
        new_mach = mach;         %
        new_mach(2,a) = mach(2,b);
        new_mach(3,a) = mach(3,b);
        new_mach(2,b) = mach(2,a);
        new_mach(3,b) = mach(3,a);
        temp = nodes(a,2:3);
        nodes(a,2:3) = nodes(b,2:3);
        nodes(b,2:3) = temp;
        mask = (floorplan == b);
        new_floorplan = floorplan - (b-a)*mask;
        mask = (floorplan == a);
        new_floorplan = new_floorplan - (a-b)*mask;
      end;    end; end
function [new_mach,new_floorplan] = move(mach,nmach,floorplan);
global nodes;
new_mach = [];
[area1 area2] = size(floorplan);
while is empty(new_mach)
a = fix(rand*(nmach-1) + 1.5); % select randomly a machine to be moved;
 Width = nodes(a,2);
 Height = nodes(a,3);
[ind1 ind2] = find(floorplan == 0); % select randomly a new position;
 L = length(ind1);
 for Y=1:10    % try 10 times to find a vacant block;
if L > 1
      k = fix(rand*(L-1) + 1.5);
    else
      k=1;
    end
    vacant = 1;
    if (ind1(k)-1+Width <= area1 & ind2(k)-1+Height <= area2)
      for g=1:Width
        for h=1:Height
          if not(floorplan(ind1(k)-1+g,ind2(k)-1+h)==0)
            vacant = 0;
          end;        end;      end;
 else
      vacant = 0;
   end
   if vacant, break;end
  end
  if vacant
   new_mach = mach;
   new_floorplan = floorplan;
   new_mach(2,a) = ind1(k)-1;
   new_mach(3,a) = ind2(k)-1;
   mask = (floorplan == a);
   new_floorplan = floorplan - a*mask;
   for g=1:Width
     for h=1:Height
       new_floorplan(ind1(k)-1+g,ind2(k)-1+h) = a;
     end;   end;  end;  end
 function report(mach,floorplan,vacant,Temp,Cost)
  [dummy nmach] = size(mach);
 figure(1);
 set(1,'Position', [432 233 365 326]);
 aux = ones(size(vacant));
 v = nmach*vacant - aux;
```

```
colormap(gray);
imagesc(floorplan + v);
title('Black=restricted, Gray=unoccupied');
axis('ij');axis('on');grid on
figure(2);
set(2,'Position', [59 232 365 326]);
semilogx(Temp,Cost,'o-');grid
axis('square');
xlabel('Temperature');
ylabel('Placement cost');
```

D.10.1. Improved *SA* Layout Placement – Output Example

```
» load('arcs.dat');
» arcs
arcs =       1   2   1   1
             2   3   1   1
             2   4   1   1
             2   5   1   1
             3   6   1   1
             3   7   1   1
             4   8   1   1
             5   9   1   1
             6  11   1   1
             6  10   1   1
             7  12   1   1
             8  13   1   1
             9  14   1   1
            14  15   1   1
» load nodes.dat;
» nodes
nodes =      1   1   2
             2   2   2
             3   2   1
             4   2   3
             5   1   1
             6   2   2
             7   1   2
             8   2   2
             9   1   1
            10   3   2
            11   1   2
            12   1   3
            13   1   2
            14   2   2
            15   1   1
» area = ones(12,10);
» area =     1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
» area(5,3)=0;
» area =     1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
             1   1   1   1   1   1   1   1   1   1
```

```
              1   1   1   1   1   1   1   1   1   1
              1   1   0   1   1   1   1   1   1   1
              1   1   1   1   1   1   1   1   1   1
              1   1   1   1   1   1   1   1   1   1
              1   1   1   1   1   1   1   1   1   1
              1   1   1   1   1   1   1   1   1   1
              1   1   1   1   1   1   1   1   1   1
              1   1   1   1   1   1   1   1   1   1
              1   1   1   1   1   1   1   1   1   1
» loc=placev2(nodes,arcs,area);
Temp = 3.640000  cost = 81.500000 uphill_cnt = 11
Temp = 3.312400  cost = 70.500000 uphill_cnt = 16
Temp = 3.014284  cost = 63.500000 uphill_cnt = 9
.........................................................
.........................................................
Temp = 0.005432  cost = 28.500000 uphill_cnt = 0
Temp = 0.004943  cost = 28.500000 uphill_cnt = 0
CPU time (sec): 44.120000
floorplan =
   0    0    0    0    0    0    0   10   10    0
   0    0    0    0   12   12   12   10   10    0
   0    0    0    0    0    7    7   10   10    0
   0    0    0    0    0    0    3    3    6    6
  13   13   -1    4    4    2    2    0    6    6
   8    8    0    4    4    2    2    5   11   11
   8    8    0    4    4    1    1    9   14   14
   0    0    0    0    0    0    0    0   14   14
   0    0    0    0    0    0    0    0   15    0
   0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0
```